Sonar based localization for the PSUBOT: a computer assisted motorized wheelchair for the disabled capable of operating in a mapped environment.

Kevin Stanton[†]

Portland State University
Department of Electrical Engineering
PO Box 751
Portland, OR 97207
(503) 725-3807

ABSTRACT

This is the abstract

TABLE OF CONTENTS

[†] (kevins@ee.pdx.edu)

I. Introduction

This introduction has yet to be written.

II. General System Overview of the PSUBOT

The PSUBOT system is broken into four subsystems: User Interface, Motion Planning & control, External Sensors, and the Operating System. The following sections discuss these fundamental components of the PSUBOT.

A. User Interface

Due to the constricted motion of the target user of the PSUBOT, the conventional hand-operated mechanical interface (joystick) was deemed unacceptable. Other existing methods include machine observation of: breath, eye movement, and linkages connected to the chin for those able to move their head freely. However, with current and future technology we believe it to be possible to use voice input without introducing excessive delays into the control loop or requiring prohibitively expensive hardware for speech processing by using a limited vocabulary and speaker independent recognition.

The input language is well structured and hierarchal, further simplifying the task of recognition, since only a subset of all possible words need be considered at a particular position in the input statement. The highest level of the command tree has three categories: ‘Assist’, ‘Utility’, and ‘Move’.

The utility and assist subtrees will be discussed only briefly, while the motion commands will be addressed in more detail as they relate more to this work.

Utility-

The computer will have a large amount of state information, some of which may be of interest to the rider. For example, the PSUBOT will know the time and date, battery life remaining, and perhaps even the weather conditions outside. All this and other more will be accessible by the rider from the utility subsystem. The computer will also be able to store phone numbers and appointments, play a game, and even play the radio. All this is for the purpose of making the PSUBOT as friendly as possible and attempt to make it act more like an able companion than a machine.

Information-

> It is also our goal to make the rider feel like an intelligent partner in the operation of the PSUBOT rather than simply an extension of a machine. To this end (among others) the rider will be given the opportunity to provide information to the PSUBOT to help it make decisions. This may be done voluntarily (from the ‘information’ subtree) or in response to a request for information by the PSUBOT. While additional information may add to the performance of the overall system, it must not be necessary; i.e. the PSUBOT must have the ability to navigate without human involvement, yet critical decisions may well be aided by the cooperation of something (someone) significantly more perceptive than itself (and many many times more complex).

As an example, consider the PSUBOT and rider traveling down a corridor when they meet a pile of boxes nearly obstructing the hallway. The PSUBOT could ask the rider whether he/she suggest they try going around to the left or the right. Not only could the rider likely make a decision in a glance (faster than the PSUBOT), but the rider could probably take more factors into account than could a machine algorithm.

To sum up, it would seem a terrible waste to limit the PSUBOT to internal decision making strategies only when a wonderfully complex and intelligent human is available for ‘comment’

Move-

Voice commanded motion is available using high and low level constructs. The low level commands simply replace and extend the basic joystick capabilities. These include, but are not limited to: forward, stop, turn, right, left, rotate, slower, and faster. These capabilities allow the user fine grained control over wheelchair motions where the necessary higher level commands are absent or found to perform unsatisfactorally in a particular situation.

High level commands extend the capabilities of the wheelchair to an autonomous mobile robot. Commands include: ‘go to room 312’, ‘find the drinking fountain’, ‘go to my office’. Clearly the execution of these commands requires an intelligent navigation system (see chapter 3).

Natural language systems are currently in place for user/computer interaction within restricted application domains. The requirement of a narrow domain is currently necessary to simplify the number of words which must be searched and limit the context in which those words may be used in order to eliminate many of the ambiguities which would otherwise result. Although the PSUBOT would likely have a small enough vocabulary for a natural language interface, we choose a more rigid grammar for the current version of the PSUBOT, as such refinement is not the major thrust of this work.

A simplified grammar is shown below for some low-level commands:
‘forward’ {dist} {‘at’ speed} ‘go’
‘rotate’ {x ’degrees’} ‘go’
The only high-level command considered thus far is ‘goto’ (and also the most difficult to implement) as it serves testing and motion verification purposes.

Speaker Dependence

The voice input system is speaker independent; i.e. the computer is trained to recognize the riders voice, though another person’s voice will also be recognized if they pronounce words similarly and use similar inflections. Speaker independent recognition (designed to recognize anyone’s voice without modification) is an unnecessary luxury since there will likely be only one primary user of a production PSUBOT, and thus only one speaker. Speaker dependence also adds a small amount of security since a non-rider shouting commands to the PSUBOT would need to sound similar to the rightful owner for the PSUBOT to respond. Voice fingerprinting is a possible extension to the current voice system. In this way the characteristics of the user’s voice would serve as a password to inhibit foul play.

B. Motor Control

Motion of the PSUBOT is accomplished by two large rear wheels each of which is powered by a permanent magnet motor and belt. The motors are Pulse-Width Modulated (PWM) to provide variable speed by a solid-state control module mounted on the back of the wheelchair. Normally, the input to the module is a joystick positioned for easy access by the rider’s right hand. The two potentiometers within the joystick are connected at right angles and oriented in an ‘X’ so that a forward thrust on the joystick displaces each potentiometer by the same angle. The outputs for each axis (each wheel) is a push/pull pair with min/max values of 2.8V and 3.6V respectively. In other words, the voltages V_{L1} and V_{L2} are related by the equations:

$$V_{L1} + V_{L2} = 3.2V$$

and

$$2.8V < V_L < 3.6V$$

where V_{L1} and V_{L2} are the push/pull control voltages for the left wheel. Of course the same relationship is true for the right wheel controller. It is then the task of the computer to generate these signals in place of the joystick. This is accomplished with the PSUBOT motor interface of appendix A which employs a pair of Digital to Analog Converters (DAC’s).

Feedback

As with any feedback control system, the PSUBOT has sensors to measure the controlled quantity (speed). This is accomplished by a pair of optical encoders connected to each wheel by a chain and sprocket. The encoders require +5V and ground, and output two TTL-level square waves which are proportional to the angular rotation, and are 90° out of phase. From the gear ratios, encoder resolution and tire size we can calculate a distance relative to each cycle of the square waves. From their phases we can determine direction. See figure 1

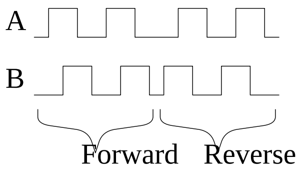


FIGURE 1.

The feedback board counts up or down depending on the phase so that an aggregate distance is available to the computer each time the control loop is executed.

Control Algorithm

A PID (Proportional + Integral + Derivative) feedback control algorithm is used to maintain accurate wheel velocities. The control loop performs one loop (sample, calculation and output) and exits, allowing other functions to be performed on the (currently) uniprocessor PSUBOT (See section on the PSU OS).

C. Sensors

In addition to the feedback sensor, several maneuver-based sensor systems assist with localization and obstacle avoidance. A SONAR device is located at a height of approximately six feet above the floor on a stepper motor to provide rotation. The output of the sonar device is a distance an an angle. The distance represents the number of feet to the nearest obstruction in the direction which the SONAR transducer points. This data is used in the localization system described in chapter 5.

There will also be SONAR transducers located one foot above the ground around the perimeter of the PSUBOT for use with close proximity (less than ten feet) obstacle avoidance (see chapter 4).

A flux-gate compass is another of the contributing sensors to the localization system. Global heading is thus determined to a resolution of eight bits (0-255). The compass was a commercial automobile product sold by Radio Shack and modified by us to allow the computer to digitize and process the necessary analog signals and extract heading information. Again, the reader is referred to chapter 5 for additional details.

D. Software Systems

It's Object Oriented!

All the major software subsystems were designed using an object oriented approach as described in [Kerth88a]. This easily translated into classes and member functions when implemented using C++. It is stressed that a good object oriented software solution comes from a good object oriented design (OOD).

A good OOD comes by asking four questions:

- what are this object's responsibilities'
- what hidden information does it hold
- how does it let others view its hidden data
- how does it allow others to modify its hidden data (if at all)

instead of

- what does this function receive as parameters
- what global constructs does it access
- what does it do to the data
- what does it return

Resultant from this type of thinking about the problem: a larger picture of the overall system emerges, related functions are grouped together, and a well-defined interface between classes (objects) is defined and enforced. It is the authors view that a bad OOD will probably yield worse program code than its corresponding functional design, but a good OOD lends itself to cleaner code and fewer chances for programmer error as well. Cox in [Cox91a] also draws out some of the specific advantages of using an object oriented language such as C++ to write programs which deal with physical systems, especially robotics.

1. Framework

The software for the PSUBOT is described in terms of an object diagram as seen in figure 2. The boxes represent the objects which perform a particular role, and the arrows indicate the direction in which messages are initiated. For example, the Pilot may send a message to the Wheels telling them to increase their speed, or may send a message asking how fast the left wheel is turning. So, the flow of information is actually bidirectional but the pilot is in control.

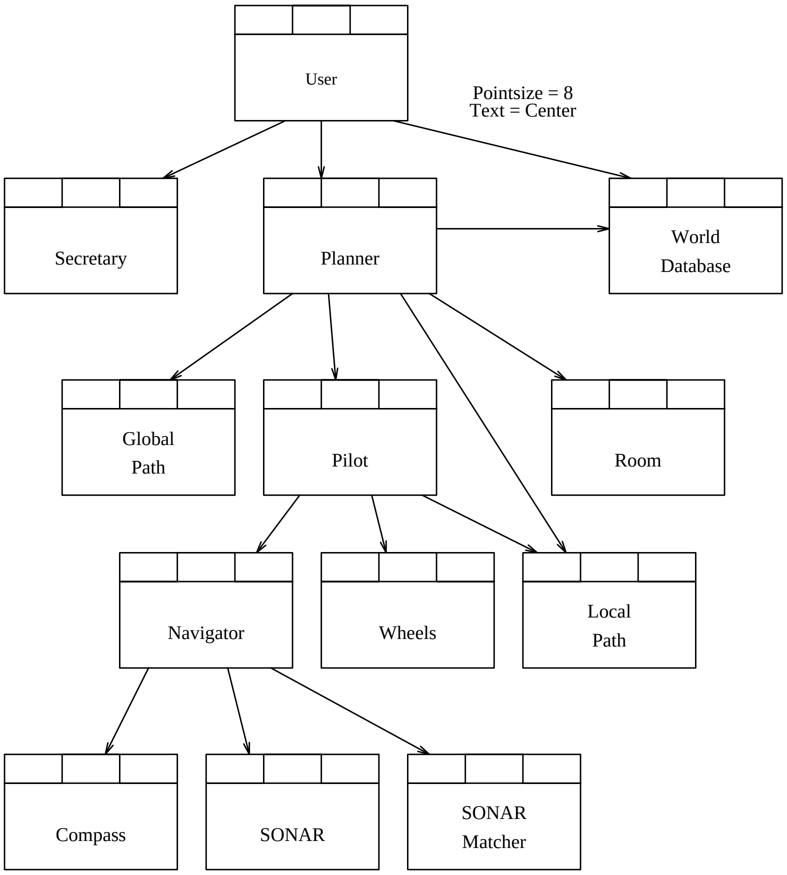


FIGURE 2.

A more detailed picture would include all the message names, parameters, and the return values. Such a diagram is included in the Appendix for all objects implemented.

Execution starts with the User object which interacts with the rider through voice communication. Depending on the type of command or question stated one of the next three Objects will interpret and perform the desired operation. If the command is of the ‘motion’ type, the Planner takes over and begins the process of path planning and execution. The Planner first asks for a Road Map (sequence of rooms to the goal) from the World Database, and then generates a Local Path through the current Room (also provided by the World Database).

Next, the Pilot begins by asking the Navigator where it is and commanding the Wheels as it follows the local path. It is the task of the Pilot to reach the local goal in the room while avoiding collisions.

The Wheels implement the PID controller which controls the two differentially controlled wheels of the wheelchair.

The Navigator is the main emphasis of this work, as it is responsible for localizing (finding the current location) based on sensor data. It also gives the Pilot information on impending collisions while drawing on information from the SONAR and Compass interface objects.

If the command received by the User object was one of the accepted utility commands or questions, the Secretary takes the appropriate actions.

It can be noted that if the User is able to receive commands while the Pilot is working, there must be some level of multitasking present in the structure. The current solution to this problem is to have a passive custom scheduler which, when asked, tells the object whether it may execute or not. So, the execution sequence in the tree would begin with the user, pass to the Secretary (which may or may not try to do anything useful), then to the Planner, to the Pilot (if applicable), and so on until all the objects are given the chance to execute. This in turn requires the ability for the objects to save state information so that each time a module is told to work it can begin where it left off. More information on the specifics of this scheme may be found in the Appendix.

III. Path Planning/Obstacle Avoidance

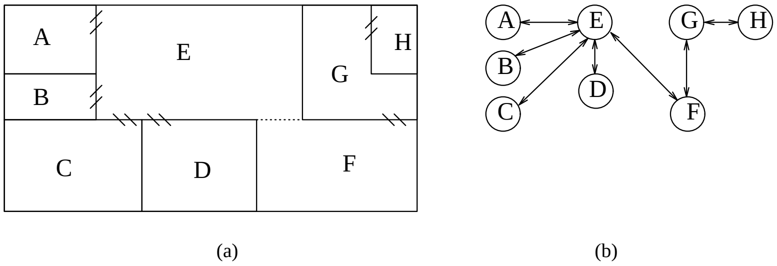
For a robot to deal intelligently with the real world it must be able to deal with unexpected events. Often these events are obstacles in its path due to displaced furniture, people walking down the hall, or disorientation on the part of the robot. This chapter describes what a path is, methods of generating it, how to follow it, and what to do when something is in the way.

1. Overview of Path Planning

What is a path? A path is something which shows the robot where to go so as to reach the goal. Ideally, a path for the PSUBOT would be free from obstacles and would minimize the distance traveled. There are two fundamental prerequisites of a path: (1) an initial position, and (2) a final position (Goal). The initial position of the PSUBOT is derived from its surroundings (see localization in Chapter IV) and the goal is supplied by the user. A path, then, should serve to connect these two points in such a way that the application of the path to a pilot will result in the robot achieving the goal state. Given the layout of the building and the location and sizes of known obstacles, the problem may be reduced to a graph search.

However, for an entire campus the number of nodes could number in the thousands requiring significant computation to find a workable solution. This excess in complexity is unnecessary with a little hierarchal decomposition of the world. First a graph is generated containing all the rooms of the current building as nodes, openings between them as arcs. This graph describes all possible routes in the building. A tree searching algorithm may then be used to produce the set of nodes (rooms) through which the distance (or some other pertinent variable) to the goal is minimized. This procedure may then be repeated for buildings and their interconnections; See figure 3.

Note that some rooms (corridors) may be split into two (‘E’ and ‘F’ in the example) to allow for a more regular room description. This search would contain about as many nodes as there are rooms and corridors in the building - in the order of tens or hundreds. An example path from room ‘B’ to room ‘G’ is {B,E,F,G}. (Actually the global decomposition starts at a higher level - buildings, floors, rooms - but is simplified for this discussion. See [Mayi91a] for a complete description of global planning for the PSUBOT.)



(a) (b)

FIGURE 3. Sample Building Layout

Once a sequence of rooms is obtained, a local path is generated for the first room which terminates at the entrance to the next room in the global path. The pilot then attempts to follow the path. If successful, a path through the next room is planned, and so on until the ultimate goal is reached. In this way we can greatly reduce the complexity of the local planner, since much tree-pruning is done at the global level.

There are two major ways to represent the world in research today. The first (both historically and in my presentation) is the spacial representation where the environment is laid out on a 2 (or 3) dimensional grid. The resulting algorithms for operating within such a framework are inherently numerical in nature. The second method of representing the environment is more symbolic in nature and is based more on artificial intelligence techniques than equations and highly processed sensor data. But first present an overview of several dominant methods in the geometric path planning relm.

1.1. The visibility graph method

Lozano Perez first formalized the visibility graph approach in his landmark paper: [Lozan79a]. It begins with all obstacles modeled as rectangles, each of which has four vertices. Then, for any given set of obstacles, a visibility graph is generated in the following way:

```
for each vertex υ
    connect υ with an arc to every other vertex ω which is ’visible’ from υ.
```

where two nodes are ’visible’ if the line connecting them does not intersect any other obstacles. By definition, each vertex is visible from the two adjacent vertices of the same obstacle. See figure 4.

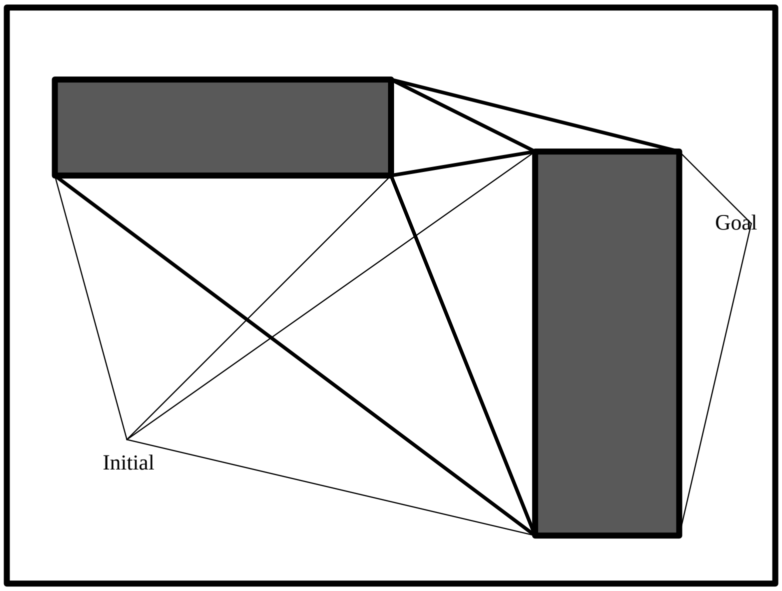


FIGURE 4. Visibility Graph

Once the graph is generated (only as often as the room changes), the initial and goal nodes are added into the graph with arcs to each ’visible’ node. Note that if there is a simple path between the initial and goal nodes (a straight line), that path will be part of the whole visibility graph. The graph may then be searched for a path meeting the minimization criterion.

In all but the trivial case, the path will bring the robot zero distance from the obstacles. There is no problem if the robot is a point, but for real systems there must be another step to keep the robot a safe distance from the obstacles. The process proposed in the above paper is called ’obstacle growing’ and involves increasing the dimensions of each obstacle artificially so that the real robot (when touching the grown, virtual obstacle) will still be a safe distance from the real obstacle.

1.2. cell decomposition

This method (also known as ’quad-tree’) differs from the visibility graph method in several respects. First, obstacles may be of arbitrary shape. Second, a path need not follow the outline of obstacles which are blocking the path, but may take a more natural curving path. See figure 5.

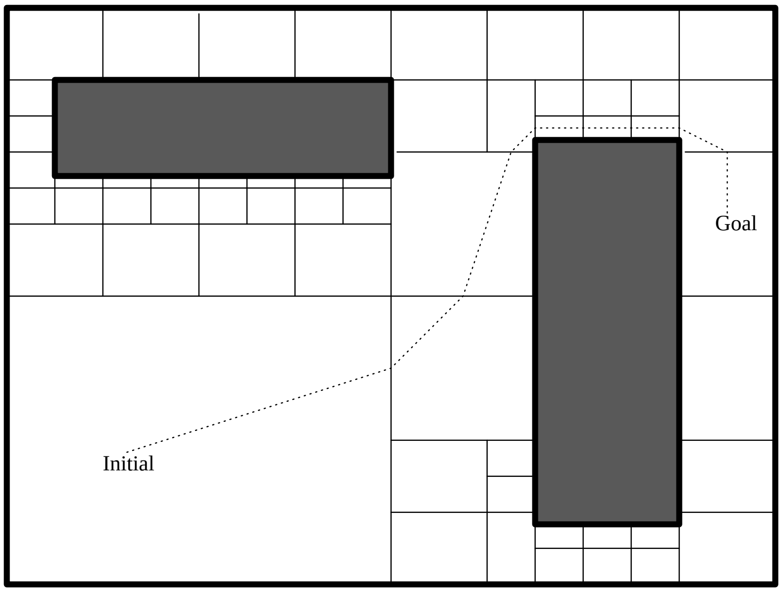


FIGURE 5. Cell Decomposition

To perform cell decomposition the space is broken into several rectangular pieces and labeled ’empty’ or ’full’ depending on whether no obstacles are with the boarders or at least one obstacle is within the boarders of the ’cell’ respectively. Those cells which were labeled ’full’ are decomposed further by dividing each of them into smaller areas, each of which are labeled ’full’ or ’empty’. The process continues until some minimum size cell is reached. Then all cells become nodes in a graph, and adjoining sides become arcs between the nodes. The arcs commonly are weighted with the relative distance between cells. The graph (as in the visibility graph method) may then be searched for a sequence of cells from the cell containing the initial position to the cell containing the goal. The actual path may then be constructed by connecting the midpoints of the sides of adjacent cells in the sequence. [Lozan81a],

A disadvantage of this approach is the complexity involved when there are many obstacles, since the number of nodes in the graph may grow large very quickly. On advantage is that gross planning may be performed with little computation if the minimum sized cell is sufficiently large. A side affect of this strategy is that a path which might exist in practice may remain unfound due to a lack in resolution.

Although the path is smoother, it is still piece-wise linear which yields jerky movements. The next method, voronoi diagrams, generate smooth paths which are equidistant from near obstacles.

A variation of the cell decomposition method is called occupancy grids [Elfes87a]. As with the above approach, space is divided into rectangular (or square) cells. However, the cells themselves are given weights depending on how certain the robot is that they are occupied. This information is generally deduced from a range sensor. So, as the robot moves through space it continuously scans the area and makes modifications to the relative ’occupancy’ of each cell in the grid. Planning is then accomplished by finding the sequence of cells to the goal which corresponds to the lowest probability of colliding with an obstacle. A nice characteristic is that raw sonar data is used directly in determining where obstacles are, and the information has a history effect in that the results of multiple readings are combined allow more reliable conclusions.

This method does not require a priori knowledge of the environment, and it is therefore more difficult to specify a goal in absolute terms. However, its ability to adapt quickly to changes in the structure of the environment make it useful in unstructured environments and outdoors as in the CMU Mobile robotics lab [??].

1.3. Potential Fields

Think of the robot and all obstacles as negatively charged particles and the goal as a positively charged particle with the robot being the only one able to move, and one will have a general understanding of what the potential fields method is all about [Khati86a]. A mathematical model is generated to describe the strengths of the repelling fields generated by the obstacles, and the attracting field generated by the goal, so that if the robot is allowed to travel along the gradient of the curve it would eventually find itself at the global minimum (the goal). That is, if there are no **local** minimums to become trapped in.

Local minimums are the major problem with potential fields approaches to path planning, and only specially designed algorithms to detect and eliminate such null areas are able to spur the robot on to the ultimate goal. Another problem with this approach is the numerical computation required to generate and search the potential fields functions for the solution path. For an overview of the potential fields method of path planning, see [Kodit89a].

1.4. Voronoi Diagrams

Another approach which has been used to generate paths through a known set of arbitrary shaped obstacles is known as the voronoi diagram [O*’82a]. With its origins heavily in mathematics the voronoi diagram has found its way into numerous real world applications like the simulation of crystal growth and locating a business as far as possible from any of its competitors.

A voronoi diagram is the locus of points equidistant from the nearest two obstacle surfaces. See figure 6. Two additional segments are added to connect the initial and final points to the voronoi curve and then the whole curve is searched for the optimum path. If the obstacles are all piecewise linear then the diagram will consist of line segments and parabolic arcs [Latom91a].

1.5. Symbolic Representations

A major and controversial research direction coming from the MIT Artificial Intelligence Laboratory is the subsumption approach to robot planning and motion. Using this approach, the robot knows nothing of the geometry and layout of a building in space, but rather moves from one place to another by finding, labeling, and moving between landmarks as in the robot Toto [Matar90a] which consists of several simple distributed processors. The philosophy of much of the subsumption architecture is that of assimilating the operation of simple biological systems such as the ant, bat, cockroach, etc. [Brooka]. Though resulting

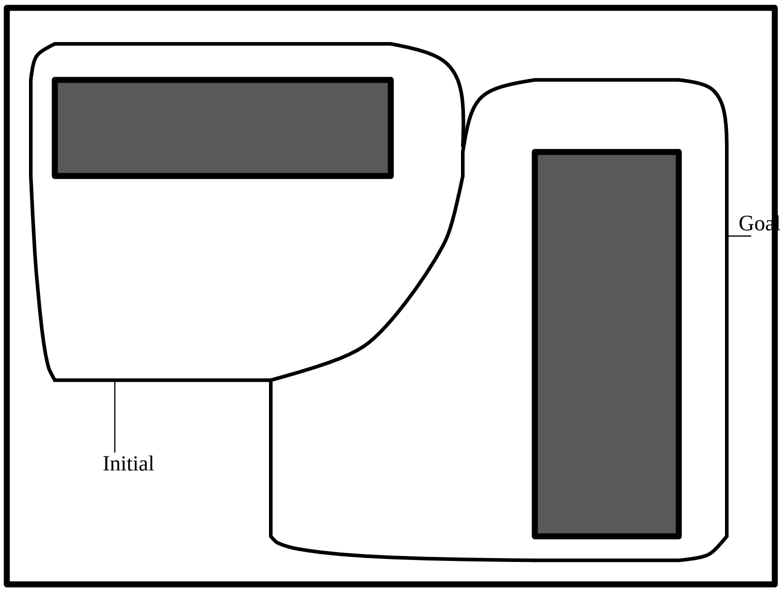


FIGURE 6. Voronoi Diagram

robots have received much media attention for their innovation and very small hardware requirements (one robot 'Squirt', though only slightly larger than one cubic inch, is able to hide in dark corners and run from loud noises [Flynn89a]) some question the utility of such simple and reflex-oriented robots. It is, however, difficult to discount their ability to operate and adapt to unknown and changing environments.

2. Path Representation for the PSUBOT

The geometric method of representing an environment is particularly attractive to a project such as the PSUBOT where the structure of the building is known beforehand. For a thorough theoretical description of the above methods and others the reader is referred to [Latom91a].

A problem with many geometry-based methods of path planning is that they are very rigid (made up of line segments and arcs). This means that if an obstacle of non-zero width is found in the path, replanning of the *entire* remaining path is required - which often is very computationally expensive. We decided to look for a method which would be flexible, and at the same time simple to compute.

3. Not lines, Channels

The path description chosen for the PSUBOT is a combination of the visibility graph method and a construct of our own, called 'quad channels' (not to be confused with 'quad trees'). The basic notion of a path is hence expanded from one dimension to two. See figure 7.

A channel is an obstacle-free region in which the robot may move in its trek to the local goal (the goal point in the current room). The channel is made up of a sequence of 'quads' each of which have one or two of their four sides in common depending on whether they are at the beginning, middle, or end of the channel. A Quad Channel may be simplified to a piece-wise linear path by connecting the midpoints of the common sides of the quads to each other and to the initial and goal points. The advantage of the channel over the resulting linear path is that the robot may attempt to maneuver around an unexpected (unmapped) obstacle without replanning. This obstacle avoidance is allowed as long as it does not move outside the channel. This provides flexibility for the robot while ensuring that it doesn't wander off in an attempt to maneuver around an unforseen obstacle.

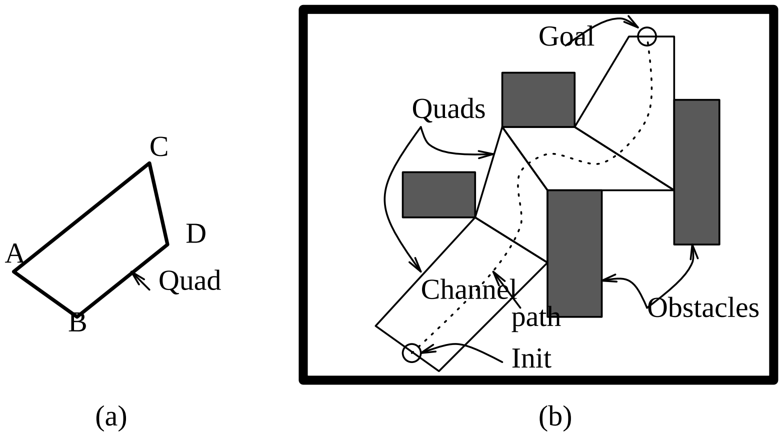


FIGURE 7. (a) Quad (b) Channel within a room

4. Path generation

A modified visibility graph algorithm would seem the best prospect for generating channels since the boarders of obstacles make natural boundaries for the channel, and all obstacles are to be represented as rectangles. If needed, a three sided quad could be generated by setting the length of one side to zero. In this case all other sides would need to be non-zero.

The input to the channel generator (path planner) is the dimensions of the room, all obstacles approximated by rectangles, and the initial and goal positions within the room. The output is an ordered collection of quads which make up the channel.

The actual algorithm for generating the channel is left as a remaining research topic and is not within the scope of this research. It is included only as a part of the overall framework for the PSUBOT developed by the author.

5. Following the Path

To solve the real world path-following problem, two fundamental questions must be answered: (1) Where are we, and (2) Where should we go now? If we cannot answer question number one we would never know when the goal has been reached or be able to navigate intelligently around known obstacles. The importance of the second question is obvious. We now discuss localization (where are we) followed by navigation (where to go now).

5.1. Localization

The first question is known as the ‘Localization Problem’ and has been studied in recent years by [Cox91a] , [Mayi91a] , [Crowl89a]. It if formulated as follows: Given the world as perceived by available sensors (often with conflicting conclusions), what is the best estimate of the robot’s position and orientation. In our case we have only sonar (distance,ϕ) and a compass (heading). If we intend to navigate within a channel consisting of *quads*, it is clear that we must have the ability to deduce our current position.

Localization is the major emphasis of this work, and is discussed extensively in chapter 4.

5.2. Commanding Motion

Once we have the ability to determine the current position in a room, the next function the robot will require is the ability to follow a path described in terms of quads as explained earlier. A simple algorithm for following a path can be imagined where the robot identifies an immediate goal at the midpoint of the opposite side of the quad and reaches it through one rotation and one forward movement (assuming no unexpected obstacles). In reality, however, rooms often contain people and other unmapped obstacles. Therefore, to perform acceptably the robot must have the ability to deal with obstacles and/or an incorrectly specified path (one which even contains *known* obstacles). Lumelsky describes a method for maneuvering around unexpected obstacles. He names the algorithms ‘bug1’ and ‘bug2’ [Lumel87a]. bug1 is shown to have a better **worst case time** complexity while bug2 has a lower **average time** complexity. Both guarantee a solution when one exists. We choose to implement the bug2 algorithm since its average time of execution is less than bug1, and since it takes an artificially contrived example to demonstrate better performance with bug1.

A robot controlled by the bug2 algorithm moves directly toward the goal until the it is reached, or until an obstacle is encountered. If the goal is reached, the algorithm terminates. However, if an obstacle is encountered the robot turns and follows the perimeter of the obstacle until the line from the starting point to the goal is reached. Then it resumes motion along the original straight-line path until the goal is reached or until another obstacle is encountered, and so on (see figure 8). This algorithm requires - in addition to localization - two capabilities: Goal Seeking, and Wall Tracking. These two can be thought of as fundamental behaviors which the pilot can switch between as necessary so that one of the two is controlling the robot at all times during navigation.

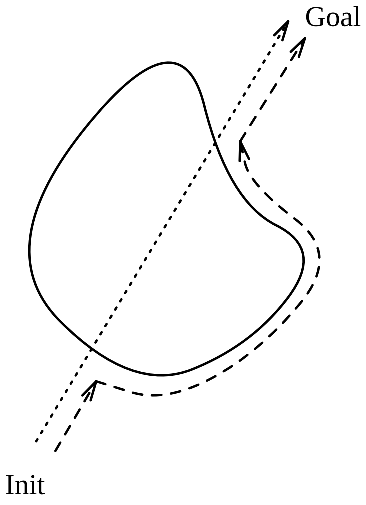


FIGURE 8. Typical bug2 Path Trace

5.2.1. Goal Seeking

Goal seeking consists of following the line which goes between the initial and final positions and consists of three steps: Heading the robot toward the goal, moving forward until the goal is reached (or an obstacle), and a final rotation once the goal is reached. The maneuver may be characterized by two rotations: α_1 and α_2, and the distance between the points: *D* (see figure 9).

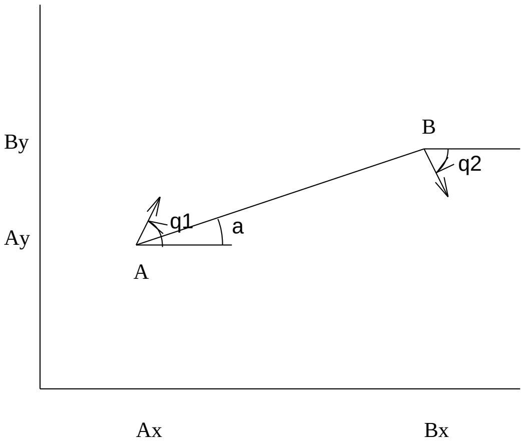


FIGURE 9. Goal Seeking Parameters

These are found in terms of the initial and final points (A and B) and the initial and desired final headings (θ_1 and θ_2):

$$\alpha_1 = \tan^{-1}\left(\frac{y_b - y_a}{x_b - x_a}\right) - \theta_1$$

$$D = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$$

$$\alpha_2 = \theta_2 - \alpha_1$$

5.2.2. Wall Tracking

If an obstacle is passable, then tracking its exterior will generally result in reaching the other side. Yet, the question remains: when an obstacle is encountered, should the robot turn left or right? Our solution is to turn right first. If tracking the obstacle on the right leads the robot outside the ’channel’ then it turns around and tries the other direction.

If the obstacle obstructs the entire channel, the pilot gives up, allowing the channel generator (path planner) to generate a less restrictive channel.

To track a wall (i.e. the boundary of an obstacle) the robot must periodically receive readings from at least 2 sensors: one on the side (sensing the distance to the wall), and one to the front (preventing collisions with a concave obstacle). Using these the robot may maintain a constant distance from the obstacle and ensure that it doesn’t collide with the same or a different obstacle.

If, for example, the distance to the right side of the robot was found to jump suddenly to a large number, the robot would deduce that the edge of the obstacle had just been passed (see figure 10).

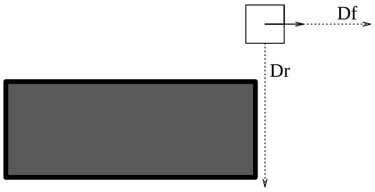


FIGURE 10. Robot Reaching Edge of Obstacle

6. Coordinating the Behaviors (The Pilot)

The objective of the Pilot is to direct the robot to the local goal (within the room). While carrying out this task it may become necessary to maneuver around obstacles found in the path. One could characterize the obstacle for inclusion in the obstacle set and replan the path taking the new obstacle into consideration, or simply try to go around it. We choose the latter because it avoids the complexity of object characterization and the time necessary to replan the path yet provides the flexibility of the more adaptive ‘behavior’ of wall tracking.

The Pilot should also be expandable so that new behaviors could be added as found necessary. Such things as moving through doors (a special kind of maneuver), moving into and operating an elevator, and approaching a charging dock are natural candidates for specialized behaviors which could be incorporated into the Pilot.

Following is the Pilot algorithm showing how it would use the goal seeking and wall tracking behaviors to move to a goal.

```
FOR each quad
    A  Turn toward local-goal (endpoint on far side of quad)
    Move forward UNTIL
        a. The goal is reached
            CONTINUE procedure for next quad.
        b. Obstacle is found obstructing path
            Turn Right
            B  Invoke Track_Wall behavior.
            UNTIL
                a. Local Goal is reached
                    REPEAT for NEXT quad
                b. Obstacle no longer blocking Goal_Seeking
                    Go To A
                c. At quad boundary
                    IF Second time quad boundary reached
                        give up.
                    ELSE
                        Rotate 180°
                        Go To B
```

IV Localization

If a robot is to be given a goal in terms of global coordinates, then it must know where it is before it can begin moving toward the goal.

I. The Need for Localization

Before the robot takes the first step in moving toward a goal it must first know where it is as was seen in the previous chapter. The process of determining ones location relative to some reference frame is called *localization.* For our purposes location will be represented by a pair (x,y) relative to the fixed frame of the origin (0,0) of the building, and a heading parameter (θ) which is defined to be zero when aligned with the x axis (see figure 11).

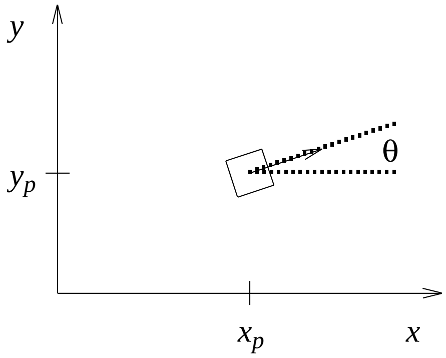


FIGURE 11.

For the robot to possess this information it must acquire data from the environment, whether it be from the surrounding walls or the earths magnetic field. This raw data must then be processed to yield an approximation of the current position. Multiple approximations (from more than one sensor) must somehow be combined into a representation which can be used for planning motions. There are nearly as many methods for processing the sonar data as there have been robots, since methods are heavily dependent on the types of sensors available to the robot.

Early techniques were rather crude with magnetic or radioactive strips embedded in the floors of the building, creating a sort of ‘virtual railroad track‘ over which the robot would be allowed to travel. The restrictions of such an approach are as inflexible as they are numerous.

Newer methods use everything from precision distance sensors to satellite positioning systems and vision in their attempt to approximate the current location of the robot.

Intended use of the robot is another influence on the methods used for localization. If the robot is to explore new territory on Mars then matching sensor data with stored images makes little sense, while a robot which is to operate within a structured environment such as an office building or factory floor can gain much information by comparing sensor readings to a stored model.

II. A Few Localization Techniques

Any method for the localization of a mobile robot may be divided into two major categories: zero drift estimation, and non-zero drift estimation. Put simply, if you navigate twice some distance with a non-zero drift method, the associated error will likewise be twice in magnitude. We first give an intuitive example and later present a more rigorous definition.

An analogy can be drawn between a robot maneuvering and driving your car on a trip. If the odometer on your car looses 0.1 mile in 100 miles, it will likely loose 0.2 miles after accumulating 200 miles - odometry is a non-zero drift location estimator. If, however, you look at the city names on mailboxes as you pass them, you likely could circle the continent several times and know which city you were in just as well after years of traveling as you did when you began - this method of localization has the property of zero-drift (though the resolution of the later is not nearly as good).

Mathematically, each component of the posture may be modeled independently by:

$$\Psi_n = \Psi_{n-1} + \Phi + \chi$$

where Ψ_n is the estimate of the parameter Ψ at time n, Φ is the true change of the quantity which Ψ is estimating since time $n-1$, and χ is a random variable. If $E(\chi)$ (the expected value of χ) is zero, the estimator is said to have zero drift. If $|E(\chi)| > 0$, the estimator is said to have non-zero drift. The $E(\chi)$ and $E^2(\chi)$ are intrinsic characteristics of any given type of localization.

We note one additional observation before describing several localization methods: Just as odometry in your car can be trusted almost whole heartedly in the short term, so other non-zero drift localization methods often have much smaller variances then their zero-drift counterparts. It will be to our best interests to use this fact when integrating several sources of location estimation.

1. Odometry
    - This method is available in any wheeled robot where the wheels are driven using feedback control. Often odometry is implemented with optical encoders mechanically connected to a turning shaft (motor, gears, or the drive wheel itself) where a direct relationship between the rotation of the encoder and the distance traveled by the wheel may be derived. One variant on this method is to place a passive measurement wheel directly on the floor behind the robot. Most robots utilize odometry based localization since the data is readily available and it often provides an accurate and fast short term estimation of the robots posture. The major problem with odometry, as with any dead-reckoning system, is the inherent drift associated with it; i.e. the error grows without bound. It is therefore necessary to provide additional means by which to determine position if trips longer than of a few feet are required.
2. Beacons
    - These devices come in two flavors: active, and passive. With active beacons, each send out unique code streams which are received by the robot and used to find their position relative to the devices through triangulation. Passive beacons are often reflectors or bar-coded strips which the robot can scan (with a light source) to determine the identity of the beacon. It then can estimate its location in a similar manner. With more passive beacons visible, a better estimation is possible. There are two requirements for the use of beacons for localization: (1) The location of all beacons must be known by the robot beforehand and (2) each building in which the robot is to navigate must be prepared with the placement of beacons before localization using this method is possible. It is because of the second of these requirements that we do not plan to use beacons for localization. See previous section (not in this document) on design assumptions of the PSUBOT.

3. Vision
   - Work has been done at Portland State University toward the incorporation of vision into the PSUBOT system [Espin1.a]. Our vision system uses the notion of ‘features’: Those attributes of an image which are deemed important and unique to a particular ‘scene’. Features from an picture taken beforehand are stored in a database along with the location of the robot when the picture was taken. Then periodically during normal navigation the vision system takes a picture, matches it to the database of stored ‘scenes’, and determines the corresponding posture of the robot. However, due to its slow speed and limited usefulness in the current narrower scope localization strategies investigated in the current version of the PSUBOT, vision is not used. Faster algorithms and hardware, and more understanding of how to incorporate different estimations of position into a single representation will be necessary before vision is of more than minimal use to the PSUBOT for localization.
4. Range Finders
   - These devices return the pair (r, θ) where r is the distance to the closest object of some minimum normal size and reflection coefficient, and θ is the angle relative to the devices’ platform on the robot. Range finders are usually optical or acoustic, but the general principles of operation are the same. A wave (sound or electromagnetic) is sent from a source in the direction θ. The wave is reflected off of an object (objects) at distance r and returns to a receiver (sometimes the same device as the source). Auxiliary circuitry then calculates the distance to the reflecting object. Time-of-flight is used for acoustic ranging devices ($\text{speed}_{\text{sound}} \cdot \text{time}_{\text{flight}}$), while laser/inferred rangers measure the phase difference between the outgoing and incoming beam. The range of the acoustic devices is often limited by reflection off walls, ceilings, and floors because of its wide cone (20 degrees), whereas laser/inferred rangers are limited by the effect of aliasing when there is a phase shift of more then 2π occurs between the source and returning signals over long distances. For a good survey of many of the ranging systems available today, see a work by CDR Everett [Evere89a].

In addition to collision avoidance, distance transducers are used to produce a scan of the area for use in navigation. Usually techniques match the generated map to a model in an attempt to normalize accumulating odometry errors.

There are two main matching strategies. One matches lines to lines ([Crowl89a], and [Elfes86a]). The second matches points to lines [Cox89a]. We describe both methods after a cursory description of the models they usually operate on.

Most models are represented by line segments. This is a natural approach since building interiors are generally made up of straight, flat walls. A curving wall may be approximated by a partial *n*-degree regular polygon with better approximations using larger values of *n*. This type of model simplifies both matching and planning algorithms in terms of computational complexity.

The first matching strategy [Elfes86a] uses the sensor data to generate lines. These lines are then compared to the model in an attempt to find a consistent offset and rotation pair representing the accumoated position and orientation error introduced by the odometry. I call this method the ‘compare abstract data to model’ method since the sensors do not return lines, but points which are abstracted into lines.

A nice feature of this method is that the model may be updated directly from the generated line segments. This feature brings up the question of ‘why begin with a model at all? Why not generate the model from scratch?’. Although implemented for the robot Neptune [Elfes86a], there were several points to consider before using such a method in the PSUBOT.

Since map-building must use some reference in determining distance, odometry is used. One would have to assume the mean error of the odometry over time to be zero in order for the generated dimensions of the building (the model) to correspond to actual measurements. This condition would be true if the robot moved about randomly for a long time to ensure that errors cancel. If, however, the robot often traveled the same direction down a hallway, it would likely have an incorrect measure of its length due to odometry drift. This ‘pulling one up by ones bootstraps’ is mentioned in [Elfes86a].

Whether or not this matters would be dependent on the desired application, and could be rectified by periodically varying which path the robot took.

The second matching strategy has been investigated by [Cox89a] and consists of matching the raw data points (distance measurements) to the line segments of the model itself. An advantage is that one needn’t be concerned with how to generate lines from sparse and spurious sensor data. This yields a simpler process.

It does not, however, facilitate updating the world model since it does not compare like types. We choose this method for the PSUBOT, however, since we assume there is a map of the building available to the computer (it is restricted to navigation in mapped environments) and since the algorithm is simpler - which is an important consideration as all computation occurs onboard.

5. Global Positioning Systems

- A GPS can calculate the modules’ global, worldwide position through the triangulation of signals from orbiting satellites. This is an example of active beacon-based localization where the beacons are sattelites orbiting the earth. Accuracy can be relatively good (within ten feet), and they are not prohibitively expensive. Even within this remarkable range, however, better methods will still be required for providing a more fine grain estimate of position. Future versions of the PSUBOT may be equipped with GPS.

6. Compass

- This device is related to (GPS) in its global nature (will work reasonably well almost anywhere in the world), but is much simpler and has a smaller average error (if such a comparison is possible). Using the orientation of the earth’s magnetic field a compass can give the heading of the robot. Magnetic compasses are plagued with errors when placed near ferrous metals, or other induced magnetic fields. The electronic flux-gate compass, however, is more immune to these perturbations and thus provides a more accurate estimate of heading. A compass is used in the PSUBOT since it is both inexpensive (< \$40) and provides a low-noise, zero-drift estimate of the θ parameter.

Comparison of localization techniques

We now demonstrate compare and contrast several parameters indicative of various localization techniques. First we compare flexibilty and complexity. See figure 12.

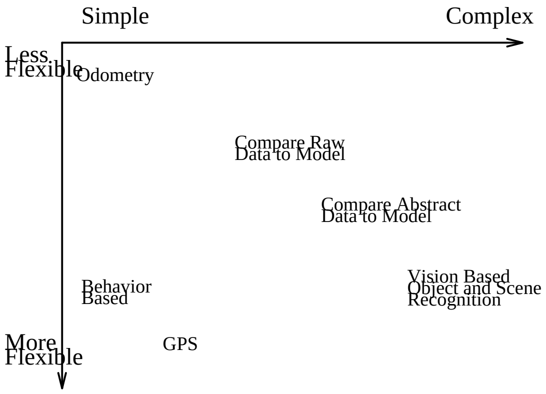


FIGURE 12. Flexability vs. Complexity

Flexability is a measure of how well the robot deals with inaccuracies in the model, sensor noise, and extra or missing obstacles.

Behavior based robots tend to be very good at operating in unstructured environments since they generally have little knowledge about the geometric or spacial layout of the building anyway. They simply obey a set of rules and ‘learn’ about the environment as they move along. Some simply respond to hard-coded reflexes such as that of ‘‘hide in dark corners and venturing out in the direction of last heard noises, only moving after the noises are long gone’’ (from the abstract of [Flynn89a]). This simplicity is also reflected by the amount of memory necessary to operate the robot: Toto from MIT (a fairly capable robot with SONAR and a compass) uses only 51K bytes for a moderate sized run [Matar90a].

Although this approach is flexible, it lacks the ability required by the PSUBOT to find a ‘best path’ from one room to another.

Global methods (GPS, compass, etc.) have slightly higher complexity (at least in terms of hardware), but are much more flexible than any other type of localization scheme presented since they need know nothing about their surroundings or where they have been. The first reading at a particular point is as reliable as the 50th, and it doesn’t matter where it was located before coming to that point.

Vision-based localization and navigation systems are very complex. Simply the sheer quantity of data requires one or more dedicated processors in order to deduce useful position information from an image. However, the potential for using vision as a navigation aid (especially stereo vision) is easily seen by realizing that much of the natural kingdom uses binacular vision to navigate.

The two localization techniques near the center of the chart use range data to ‘see’ the neighboring walls much like a bat uses SONAR to determine the shape, location, and speed of its prey.

A big advantage of using this type of sensor is that the amount of data is limited and can be reduced further by lowering the resolution of the rotating device; without severe degredation of interpreted results.

Although not the most flexible approach, various preprocessing filters can help to remove noise which would otherwise introduce errors into the resulting estimation. In chapter XXXX we demostrate the ability of our localization algorithm to deal with people moving freely around the room in which the robot is operating and find that small changes in the model make little or no change in the performance of the matcher, while large changes in the room cause the system to go unstable and give incorrect results.

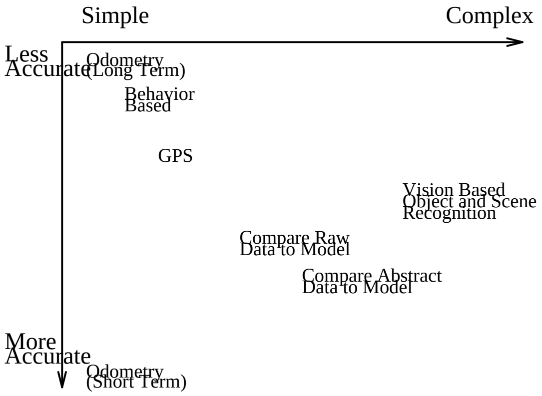


FIGURE 13. Accuracy vs. Complexity

III. Localization for the PSUBOT

The current version of the PSUBOT uses odometry, compass, and ultrasonic ranging to determine its posture within a known building. Each estimates one or more element of the triple (x_p, y_p, θ), each of which will be combined in a manner which provides the statistically optimal estimate of the current position (see section on integration of sensor results).

7. Odometry

Position estimation through odometry for a robot with two drive wheels has been studied by [Iijim81a] cited in [Wang88a] ((need to acquire this reference for myself)). The results of these will be used without modification, and are restated here for completeness.

We must know the distances traveled by the left and right wheels; ΔD_l and ΔD_r respectively. Fortunately, these values are available directly from the feedback hardware and from them we can derive the distance traveled (ΔD)and the change of heading ($\Delta\theta$). See figure 14 for the model of motion where A and C are the initial wheel locations, and A′ and C′ are the final wheel positions. B denotes the center of the rear axle whose width is L. From these values we find the distance traveled by the robot ΔD_n and the change in heading θ_n since the last sample period (at time $n-1$).

$$\Delta D_n = \frac{\left(\Delta D_r + \Delta D_l\right)}{2}$$

$$\Delta\theta_n = \frac{\left(\Delta D_r - \Delta D_l\right)}{L}$$

The new position and heading can then be calculated by adding to the parameter estimates at time $n-1$:

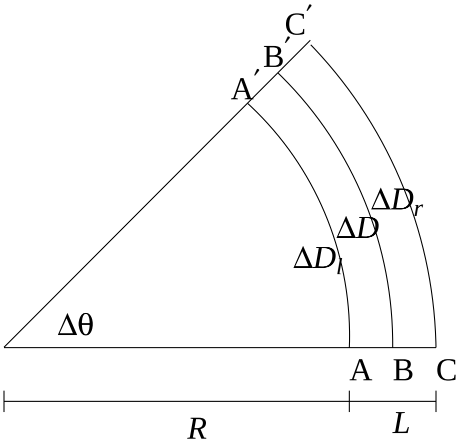


FIGURE 14.

$$x_n = x_{n-1} + \frac{\sin\left(\frac{\Delta\theta_n}{2}\right)}{\frac{\Delta\theta_n}{2}} \cdot \Delta D_n \cdot \cos\left(\theta_{n-1} - \frac{\Delta\theta_n}{2}\right)$$

$$y_n = y_{n-1} + \frac{\sin\left(\frac{\Delta\theta_n}{2}\right)}{\frac{\Delta\theta_n}{2}} \cdot \Delta D_n \cdot \sin\left(\theta_{n-1} + \frac{\Delta\theta_n}{2}\right)$$

$$\theta_n = \theta_{n-1} + \Delta\theta_n$$

Thus we find our posture at time *n* to be:

$$P_n = (x_n, y_n, \theta_n)$$

Since the measurements of $\Delta D_{r,l}$ are obtained through odometry and have an associated error which does not have a mean of zero. The recurrence relations defined above rely on odometry alone which implies that the cumulative positional error will grow without bound.

8. Compass

The flux-gate compass in use on the PSUBOT was manufactured for public use in automobiles and modified to allow acquisition of signals through a digital computer. Two signals from the compass are sampled by the computer and will be called A and B (See figure 15).

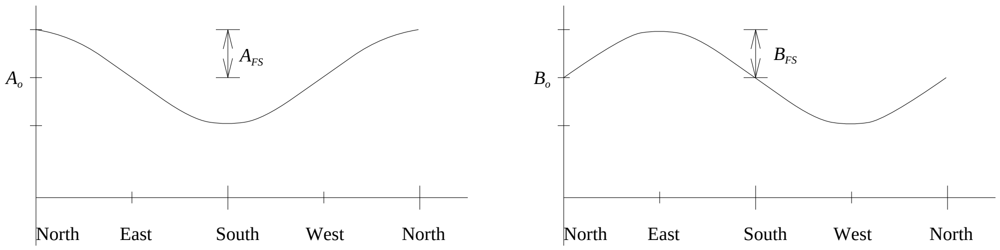


FIGURE 15.

The signals are modeled as trigonometric functions: $A = A_{FS}\sin(\theta_A) + A_o$ and $B = B_{FS}\cos(\theta_B) + B_o$ and are both needed to determine the heading of the robot since neither function is reflexive. Solving for $\sin(\theta)$ and $\cos(\theta)$ and dividing we have the tangent from which we can then find the heading

$$\theta = \tan^{-1}\left[\frac{\left[\frac{A - A_o}{A_{FS}}\right]}{\left[\frac{B - B_o}{B_{FS}}\right]}\right]$$

The parameters A_0 and A_{FS} are determined experimentally during configuration of the robot to a particular environment. There is currently no method of updating these values during the course of navigation, though a method of doing so should be rather simple.

So, the output of the compass is the angular heading of the robot with 0° mapped to the direction of the x axis and positive headings measured counter-clockwise.

9. SONAR Matcher

Of primary interest in this work is the reduction of odometry error by matching a set of range points (d, α) described earlier to a model of the world.

9.1. World Model

The World is modeled by a collection of line segments $\overline{A_iB_i}$ where A_i and B_i are points represented as column vectors $A=[A_x,A_y]^t$. We assume that $A_i \neq B_i$. See figure 16 for a possible map.

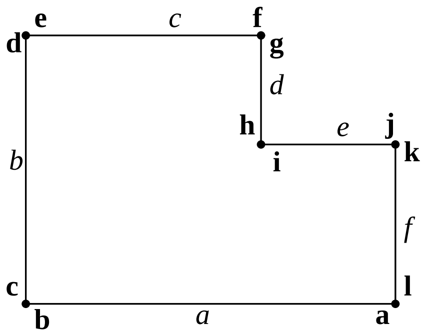


FIGURE 16.

A more complex representation of the world is allowed, provided it can be decomposed into a collection of line segments as described above.

9.2. SONAR System

Distance measurements are achieved through the use of SONAR (SOund Navigation And Ranging). Since the frequence of the sound is higher than is generally audible (> 20,000 Hz) we will also call it an ‘ultrasonic ranging system’. A Polaroid [Polaroid corp] ultrasonic transducer is used which both sends and receives signals. The system also incorporates a stepper motor for rotation, external counting/measuring circuitry, and a PC interface. The value returned from the SONAR control software is a distance, angle pair (d, α). Collecting two or more SONAR-derived points at different angles is called a SONAR *scan*. Scans are performed periodically. Range samples can be taken while the robot is in motion by using the robot's current estimation of its posture to transform each point into global coordinates at the time it is received. Once the desired scan is completed, the matcher has the responsibility of finding a correction vector which matches the points to the line segment model. The correction represents the matcher's estimation of the accumulated posture error up to the current time and is used to hold the ever-growing odometry position error to an acceptable level.

9.3. Matcher Algorithm

A common matching procedure involves minimizing the sum of the squared errors (SSE). The quantity to be minimized is demonstrated by a simple example:

Consider the robot in the following room with its estimated position shown ($P = x, y, \theta$). The robot performs a scan consisting of 5 points (see figure 17).

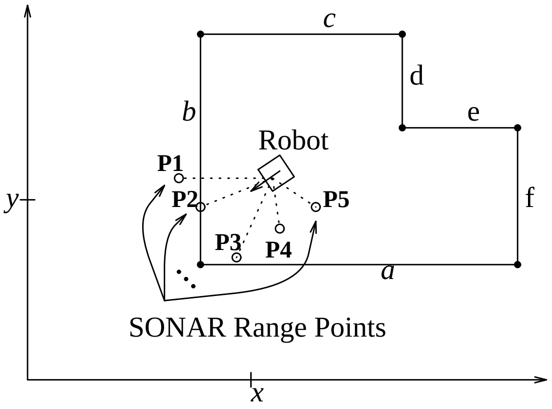


FIGURE 17.

Based on the sonar data, we can see that the robot is closer to the bottom of the image and is heading differently than it thinks. Now keeping the model constant and treating the robot and sonar scan points as a rigid body (a translation of the points moves the robot by the same amount), apply a shift in the x and y directions and a rotation such that the sum of the distances between each point and the closest line segment (the **target line** of the point) squared is as small as possible (Sum of Squared Errors). Thus $\mathbf{P}_1$ and $\mathbf{P}_2$ have b as their target line while a is the target of $\mathbf{P}_3$, $\mathbf{P}_4$, and $\mathbf{P}_5$. The other lines in the image are ignored.

9.3.1. Mathematical Fundamentals of SONAR Matcher

We now develop some of the mathematics necessary to accomplish the above task. The following method is based on a work by Cox [Nelso88a], and [Cox91a].

It will be our convention to use a bold symbol vector (**p**), lower case italic for a line segment (*l*), and an upper-case italic character for the corresponding infinite line (*L*) which coincides with the segment *l*. Scalars are in italics and the meaning will be clear from the context in which they are used. Also, since the vectors are nearly almost always of dimension 2x1 and carry physical *x/y* coordinate information, an *x* or *y* subscript will refer to the first or second element respectively of the vector.

Point:

The point[†] **p** is represented as a vector of the form: $[x,y\,]^t = \begin{bmatrix} x \\ y \end{bmatrix}$.

Line Segment:

A line segment *l* is defined by two endpoints **a** and **b** and is assumed to be in the first quadrant.

Infinite Line:

Any line segment *l* has exactly one infinite line *L* which also passes through the endpoints of *l*. Such a line can be represented by the pair: $(\mathbf{u},r)$ where **u** is the unit vector orthogonal to the infinite line *L* and the point described by $r{\cdot}\mathbf{u}$ is on *L*. (see Figure 18).

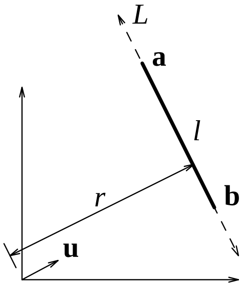


FIGURE 18.

To aid in the derivations, let us first transform *L* into the general equation of a line:

$$Ax + By + C = 0$$

The coefficients A,B,C of *L* can be calculated in terms of **a** and **b** as follows:

$$A = -(\mathbf{b}_y - \mathbf{a}_y)$$

$$B = \mathbf{b}_x - \mathbf{a}_x$$

$$C = \mathbf{a}_x \cdot (\mathbf{b}_y - \mathbf{a}_{y)}) - \mathbf{a}_y \cdot (\mathbf{b}_x - \mathbf{a}_{x)}$$

from which the following relationships can be seen:

If *L* is Horizontal	$A = 0$
If *L* is Vertical	$B = 0$
If *L* passes through (0,0)	$C = 0$

Then from geometry we find that the components of **u** are:

$$\mathbf{u}_x = \pm \frac{A}{\sqrt{A^2+B^2}}$$

$$\mathbf{u}_y = \pm \frac{B}{\sqrt{A^2+B^2}}$$

$$r = \pm \left(\frac{-C}{\sqrt{A^2+B^2}} \right)$$

[†] The terms *point* and *vector* will be used interchangeably.

where the ± sign is chosen opposite to C. If $C=0$, use the sign of B.

Distance to Infinite Line:

The distance from a point **p** to an infinite line L represented by $(\mathbf{u},r)$ is

$$\text{Distance}=(\mathbf{p}^t\cdot\mathbf{u}-r)$$

Distance to Line Segment:

The distance from a point **p** to a line segment l (defined by points **a** and **b**) is more complicated, since there are three cases. Consider the line which is perpendicular to the infinite line L, passes through the point **p** and intersects l at point **q** (see Figure 19).

/264.00p)’’

FIGURE 19.

Which of the three cases is valid can be determined by testing the order of the points **a**, **b**, and **q** along the line L.

Case 1: **qab** $\quad \text{dist}=\text{abs}\left(\,||\mathbf{a}|| - ||\mathbf{p}||\,\right) = \sqrt{(\mathbf{p}_x-\mathbf{a}_x)^2+(\mathbf{p}_y-\mathbf{a}_y)^2}$

Case 2: **aqb** $\quad \text{dist}=\text{abs}(\mathbf{p}'\cdot\mathbf{u}-r)$

Case 3: **abq** $\quad \text{dist}=\text{abs}\left(\,||\mathbf{b}|| - ||\mathbf{p}||\,\right) = \sqrt{(\mathbf{p}_x-\mathbf{b}_x)^2+(\mathbf{p}_y-\mathbf{b}_y)^2}$

where abs denotes absolute value. If **q** is equal to with either **a** or **b**, Case 2 is used. Each point in the sonar scan has as its target line the line segment of minimum distance. It is the sum of the squares of these distances which we are trying to minimize with the matcher.

Point Rotation:

It is well known from geometry that to rotate a point **p** around the origin by θ, one simply multiples it by the following orthogonal transformation matrix $R_o(\theta)$:

$$R_o(\theta)=\begin{bmatrix}\cos(\theta) & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)\end{bmatrix}$$

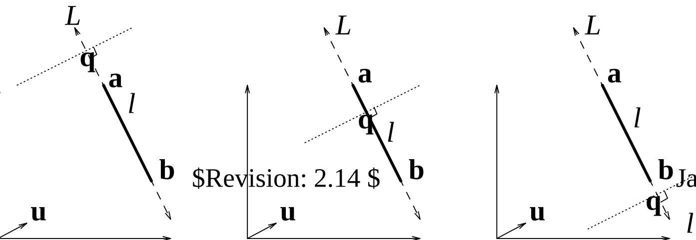
To rotate **p** around **c** which is not necessarily at the origin, we must first shift the axes and then multiply:

$$\mathbf{p}' = R_o(\theta)\cdot(\mathbf{p}-\mathbf{c})$$

Pseudo-Rotation

For reasons to be explained later, it proves useful to approximate the true rotation $R_o(\theta)$ by the first term of its series expansion:

$$R_p(\theta)=\begin{bmatrix}1 & -\theta\\ \theta & 1\end{bmatrix}$$



A pseudo-rotation is equivalent to moving points along the line tangent to the arc rather than along the arc itself. Note also, that for very small θ, $R_p(\theta) \approx R_o(\theta)$.

Transforming a correction to another reference frame

Since the solution vector we will derive is based on rotation around the centroid of points **c**, a transformation is necessary to find the correction vector relative to the robot's position. Rotation cannot be substituted by translation, so the θ parameter of each correction will be the same.

Each point $\mathbf{p}_i$ is rotated around **c** and shifted by **t**.

$$\sum_{i=1}^{n}\mathbf{q}_i = \sum_{i=1}^{n}R_o(\theta)(\mathbf{p}_i-\mathbf{c})+(\mathbf{c}+\mathbf{t})$$

We wish to find a new vector $\mathbf{t}'$ such that

$$\sum_{i=1}^{n}\mathbf{q}_i = \sum_{i=1}^{n}R_o(\theta)(\mathbf{p}_i-\mathbf{R})+(\mathbf{R}+\mathbf{t}')$$

where **R** is the current position of the robot before correction. Subtracting the above equations and rearranging yields the expression for $\mathbf{t}'$:

$$\mathbf{t}' = \mathbf{t}-R(\theta)(\mathbf{c}-\mathbf{R})+(\mathbf{c}-\mathbf{R})$$

The updated posture of the robot can then be found:

$$P(n)=(x_{n-1}+\mathbf{t}'_x, y_{n-1}+\mathbf{t}'_y, \theta_{n-1}+\theta)$$

9.3.2. Minimize Squared Error

We are now ready to derive expressions for the correction $(\mathbf{t},\theta)$ which, when applied to the points of the sonar scan image, matches them to the line segment model in the mean squared sense.

Define S as the sum of the squared distances between the points, each rotated about the centroid and translated by **t**, and the target lines (not line segments):

$$S=\sum_{i=1}^{n}\left([R_o(\theta)(\mathbf{p}_i-\mathbf{c})+(\mathbf{c}+\mathbf{t})]^t\mathbf{u}_i-r_i\right)^2$$

where there are n scan points $\mathbf{p}_i$ in the image whose centroid is **c**. $\mathbf{u}_i$ and r_i represent the target line of the point $\mathbf{p}_i$, and $R_o(\theta)$ is the rotation matrix.

Removing the $\sum$ and the square operator, the above equation can be seen in the form:

$$\text{Distance} = (\mathbf{P}^t\cdot\mathbf{u}-r)$$

where **P** is the point **p** rotated by θ about the centroid **c** and then displaced by the translation vector **t**. Since S would be linear without the trigonometric functions of $R_o(\theta)$, the pseudo-rotation matrix $R_p(\theta)$ is used to simplify the solution. However, since $R_p(\theta)$ is only an approximation of the true congruence, it will be necessary to iterate for the solution.

Setting the partial derivatives to zero:

$$\frac{\partial\mathbf{t}_x}{\partial S}=0 \qquad \frac{\partial\mathbf{t}_y}{\partial S}=0 \qquad \frac{\partial\theta}{\partial S}=0$$

yields a third order system of linear equations of the form **Ax** = **B**:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} \mathbf{t}_x \\ \mathbf{t}_y \\ \theta \end{bmatrix} = \begin{bmatrix} B_{11} \\ B_{21} \\ B_{31} \end{bmatrix}$$

where

A =

$$\left| \begin{array}{lll} \sum_{i=1}^{n} \mathbf{u}_{i_x}{}^2 & \sum_{i=1}^{n} \mathbf{u}_{i_x}\mathbf{u}_{i_y} & \sum_{i=1}^{n} \mathbf{c}_y\mathbf{u}_{i_x}{}^2 - \mathbf{c}_x\mathbf{u}_{i_x}\mathbf{u}_{i_y} + \mathbf{u}_{i_x}\mathbf{u}_{i_y}\mathbf{v}_{i_x} - \mathbf{u}_{i_x}{}^2\mathbf{v}_{i_y} \\ \sum_{i=1}^{n} \mathbf{u}_{i_x}\mathbf{u}_{i_y} & \sum_{i=1}^{n} \mathbf{u}_{i_y}{}^2 & \sum_{i=1}^{n} \mathbf{c}_y\mathbf{u}_{i_x}\mathbf{u}_{i_y}\mathbf{c}_x\mathbf{u}_{i_y}{}^2 + \mathbf{u}_{i_y}{}^2\mathbf{v}_{i_x} - \mathbf{u}_{i_x}\mathbf{u}_{i_y}\mathbf{v}_{i_y} \\ \sum_{i=1}^{n} \mathbf{c}_y\mathbf{u}_{i_x}{}^2 - \mathbf{c}_x\mathbf{u}_{i_x}\mathbf{u}_{i_y} + & \sum_{i=1}^{n} \mathbf{c}_y\mathbf{u}_{i_x}\mathbf{u}_{i_y} - \mathbf{c}_x\mathbf{u}_{i_y}{}^2 + & \sum_{i=1}^{n} (\mathbf{c}_y\mathbf{u}_{i_x} - \mathbf{c}_x\mathbf{u}_{i_y} + \mathbf{u}_{i_y}\mathbf{v}_{i_x} - \mathbf{u}_{i_x}\mathbf{v}_{i_y})^2 \\ \quad \mathbf{u}_{i_x}\mathbf{u}_{i_y}\mathbf{v}_{i_x} - \mathbf{v}_{i_y}\mathbf{u}_{i_x}{}^2 & \quad \mathbf{u}_{i_y}{}^2\mathbf{v}_{i_x} - \mathbf{u}_{i_x}\mathbf{u}_{i_y}\mathbf{v}_{i_y} & \end{array} \right|$$

B =

$$\begin{bmatrix} \sum_{i=1}^{n} r\mathbf{u}_{i_x} - \mathbf{u}_{i_x}{}^2\mathbf{v}_{i_x} - \mathbf{u}_{i_x}\mathbf{u}_{i_y}\mathbf{v}_{i_y} \\ \sum_{i=1}^{n} r\mathbf{u}_{i_y} - \mathbf{u}_{i_x}\mathbf{u}_{i_y}\mathbf{v}_{i_x} - \mathbf{u}_{i_y}{}^2\mathbf{v}_{i_y} \\ \sum_{i=1}^{n} (\mathbf{u}_{i_x}\mathbf{v}_{i_y} - \mathbf{u}_{i_y}\mathbf{v}_{i_x} + \mathbf{c}_x\mathbf{u}_{i_y} - \mathbf{c}_y\mathbf{u}_{i_x}) \cdot (\mathbf{u}_{i_x}\mathbf{v}_{i_x} + \mathbf{u}_{i_y}\mathbf{v}_{i_y} - r) \end{bmatrix}$$

Gaussian elimination produces the following results:

$$\mathbf{t}_x = \frac{-A_{23}\cdot A_{32}\cdot B_{11} + A_{22}\cdot A_{33}\cdot B_{11} + A_{13}\cdot A_{32}\cdot B_{21} - A_{12}\cdot A_{33}\cdot B_{21} - A_{13}\cdot A_{22}\cdot B_{31} + A_{12}\cdot A_{23}\cdot B_{31}}{-A_{13}\cdot A_{22}\cdot A_{31} + A_{12}\cdot A_{23}\cdot A_{31} + A_{13}\cdot A_{21}\cdot A_{32} - A_{11}\cdot A_{23}\cdot A_{32} - A_{12}\cdot A_{21}\cdot A_{33} + A_{11}\cdot A_{22}\cdot A_{33}}$$

$$\mathbf{t}_y = \frac{A_{23}\cdot A_{31}\cdot B_{11} - A_{21}\cdot A_{33}\cdot B_{11} - A_{13}\cdot A_{31}\cdot B_{21} + A_{11}\cdot A_{33}\cdot B_{21} + A_{13}\cdot A_{21}\cdot B_{31} - A_{11}\cdot A_{23}\cdot B_{31}}{-A_{13}\cdot A_{22}\cdot A_{31} + A_{12}\cdot A_{23}\cdot A_{31} + A_{13}\cdot A_{21}\cdot A_{32} - A_{11}\cdot A_{23}\cdot A_{32} - A_{12}\cdot A_{21}\cdot A_{33} + A_{11}\cdot A_{22}\cdot A_{33}}$$

$$\theta = \frac{-A_{22}\cdot A_{31}\cdot B_{11} + A_{21}\cdot A_{32}\cdot B_{11} + A_{12}\cdot A_{31}\cdot B_{21} - A_{11}\cdot A_{32}\cdot B_{21} - A_{12}\cdot A_{21}\cdot B_{31} + A_{11}\cdot A_{22}\cdot B_{31}}{-A_{13}\cdot A_{22}\cdot A_{31} + A_{12}\cdot A_{23}\cdot A_{31} + A_{13}\cdot A_{21}\cdot A_{32} - A_{11}\cdot A_{23}\cdot A_{32} - A_{12}\cdot A_{21}\cdot A_{33} + A_{11}\cdot A_{22}\cdot A_{33}}$$

We now review the matching process:

1. A SONAR scan produces a set of data points in polar form (d_i, α_i) which are transformed into the Cartesian world coordinates - $\mathbf{p}_i = (x_i, y_i)$ - based on the believed posture of the robot (the posture which is to be corrected).
2. A target line L_j is assigned to each point p_i by finding the line segment l_j whose distance from the point is minimum.
3. Repeat the following until $(\mathbf{t}, \theta)$ is very small.
    - 3.1 Find $(\mathbf{t}, \theta)$ which minimizes the mean squared distance between points and their targets using pseudo-rotation.

3.2 Apply ($\mathbf{t}, \theta$) to the SONAR data points using *true* rotation.

4. Add up all the corrections applied to the points under step 3. This is the solution vector ($\mathbf{t}, \theta$).
5. Find translation and rotation (based at the current position of the robot) which yields the same translation and rotation based at the centroid of SONAR data points.
6. Apply the solution vector to the old posture to calculate a new matcher-corrected posture of the robot.

9.4. Comments on the Matcher

The algorithm as described is subject to several downfalls due to the straight ahead mathematical approach, so let us take several steps back for a moment to expose these problems and their solutions.

9.4.1. Finding the correction vector

Notice that all correction vectors of step 3 are summed in step 4 to yield the final correction vector. We are able to do this because of the property of points rotated about their centroid. To translate and rotate a set of points by $\mathbf{t}_i, \theta_i$ about their centroid a number of times is equivalent to translating by $\sum \mathbf{t}_i$ and rotating by $\sum \theta_i$ This property is not true unless rotation occurs around the centroid of points.

9.4.2. Removal of Outliers

Due to the noisy nature of the SONAR range data [sonar is noisy reference], it was necessary to remove certain data points to allow closer matches. The rationale used here is that some points are not simply ‘‘inaccurate’’, but rather are wrong (the data doesn’t correspond to any physical distance we are interested in measuring). There are several causes for such extraneous data. One is the affect of echos, especially near corners where the sound waves may reflect off of several walls before returning to the transducer. Echoing leads to misleading readings of long distances and should be removed.

Another cause of erroneous range data is perhaps illustrated best with an example: Suppose the robot is in a corridor having a width of 8 feet and a height of 12 feet (not unlike the PCAT building in which testing occurred). Additionally assume that the SONAR device is pointing in the direction of a wall at the end of the corridor which is out of range. Clearly a finite distance reading will be incorrect, yet often occurs due to reflections of the ultrasonic sound wave off the walls, ceiling, and floor. Again, it is emphasized that we are not trying to remove data points which are slightly inaccurate, but those which have no meaningful physical relationship to the world.

The method employed by [Cox91a] is straight forward: remove any points whose distance from their target line segment is greater than some pre-defined constant. The assumption which makes the above method work is that of ‘small step’; the current estimated posture is *close* to the real posture. If the small step assumption were not true for a particular matching session, valid and possibly critical points could be removed while extraneous, noisy points would likely be retained. A more adaptive algorithm seemed to us a better choice.

We compute the average μ and standard deviation (σ) of the errors:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} \text{error}_i$$

$$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^{n} (\text{error}_i - \mu)^2$$

and remove those data points whose distance from their target line segment outside the range: $(mu - F\sigma, \mu + F\sigma)$ where F is determined experimentally to reflect the fraction of erroneous data points. It would perhaps be possible to find some correlation between the shape of the current room (line segment world model) and this quantity. We have not implemented such a scheme, but use the constant value of $F = 2$. This value of F retains about 95% of the points (assuming the distribution of the errors is Gaussian [Guttm82a].

9.4.3. Subtle implementation/intention discrepancy

Recall the meaning of the error term which is minimized by the matcher: The distance between a point and its *infinite* length target line. In other words, while the target of a point is determined by its distance from the line segments (as expected), the actual minimization equation effectively extends the target line segment to infinity in both directions and then finds the distance from the point to *this* line which causes points to be drawn toward *phantom* lines. The desired and calculated distances are different only if the distance from the point **p** to its target line segment l does not lie on the perpendicular through **p** to L. However, since target lines are determined based on distances to line *segments,* the affect of this undesirable side effect is reduced when outliers are removed. In a typical continuous line segment world model, such troublesome points occur mainly at convex (outside) corners. An algorithm could be imagined which would hunt down and eliminate these data points.

9.4.4. Indeterminate Results

The system matrix in section 3.3.2 is singular ($\frac{0}{0}$ results) when all target lines in the calculation of the correction vector are parallel, or nearly parallel. See Figure 20.

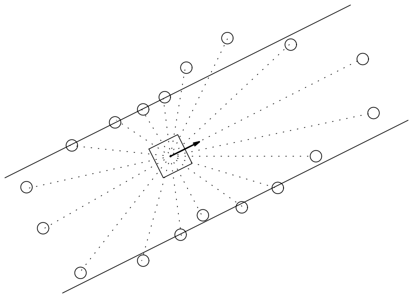


FIGURE 20.

Such a situation would not be uncommon with a robot traveling down a corridor. $\frac{0}{0}$ implies an infinite number of solutions. This makes intuitive sense when the robot is too far from the ends of a corridor to determine its lateral position. Clearly the matcher should be capable ot determining the correction perpendicular to the walls, but the position along the hallway cannot be determined from the available SONAR data points. Cox’s solution to this problem is to make modifications to the system matrix and introduce predefined, experimentally determined constants to increase the system stability. Again we attempt a more rigorous approach. Our modification of the procedure is the topic of the next section.

IV. Modifications for Parallel Target Lines

As stated earlier, an assumption which is made is that most points are near the line segment of the model they represent in the real world. This is generally a valid assumption since the purpose of the sonar matcher is to provide an update to the dead reckoning so the posture errors do not grow without bound as they would otherwise. However, even when the small step assumption is satisfied, the matching algorithm described in the previous section yields an infinite number of solutions if all the target liens are parallel. Consider a simple example shown in figure 21.

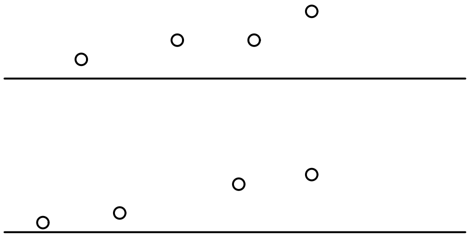


FIGURE 21.

Clearly the data points need to move in the negative *y* direction and rotate. Any value for *x*, however, yields a minimum error term and the system matrix *A* is found to be singular.

We identify three reasons why such a situation could appear in practice:

1. Data points are distant from the model (the robot is lost). In this case there is often one closest line segment which is the target of all the points.
2. The robot is traveling down a corridor: Long, narrow, parallel line segments are the targets of the points.
3. Only a partial scan was taken. If the robot were following a wall in a very large room, it may decide to acquire data only where it expects to find it valid. As in 1. above, all data points would likely have the same line segment as their target.

Since **t** in the solution vector is under-specified, we reduce the dimension of the problem from three to two; i.e. we require the data points to move only perpendicular to their target lines. Rotation, of course, is also performed.

The vector **v** is assigned to be perpendicular to the (parallel) target lines, and *K* becomes the multiplier by which the points are moved along **v**. The sum of the least squares equation becomes:

$$S = \sum_{i=1}^{n} \left([R(\theta)(\mathbf{p}_i - \mathbf{c}) + (\mathbf{c} + K\mathbf{v})]^t \mathbf{u}_i - r_i \right)^2$$

Following the same procedure as before, we find the values for *K* and θ which minimizes *S* by setting the partial derivatives to zero:

$$\frac{\partial K}{\partial S} = 0 \qquad \frac{\partial \theta}{\partial S} = 0$$

and solving the second order system of linear equations:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} K \\ \theta \end{bmatrix} = \begin{bmatrix} B_{11} \\ B_{21} \end{bmatrix}$$

where

A =

$$\begin{vmatrix} \sum_{i=1}^{n} (\mathbf{u}_{i_x} px + \mathbf{u}_{i_y} py)^2 & \sum_{i=1}^{n} (px\mathbf{u}_{i_x} + py\mathbf{u}_{i_y})(\mathbf{c}_y \mathbf{u}_{i_x} - \mathbf{c}_x \mathbf{u}_{i_y} + \mathbf{u}_{i_y} \mathbf{v}_{i_x} - \mathbf{u}_{i_x} \mathbf{v}_{i_y} \\ \sum_{i=1}^{n} (\mathbf{u}_{i_x} px + \mathbf{u}_{i_y} py)(-\mathbf{u}_{i_x} \mathbf{v}_{i_y} + \mathbf{u}_{i_x} \mathbf{c}_y + \mathbf{u}_{i_y} \mathbf{v}_{i_x} - \mathbf{u}_{i_y} \mathbf{c}_x) & \sum_{i=1}^{n} (-ux\mathbf{v}_{i_y} + \mathbf{u}_{i_x} \mathbf{c}_y + \mathbf{u}_{i_y} \mathbf{v}_{i_x} - \mathbf{u}_{i_y} \mathbf{c}_x)^2 \end{vmatrix}$$

B =

$$\begin{vmatrix} -(px\mathbf{u}_{i_x} + py\mathbf{u}_{i_y})(-r + \mathbf{u}_{i_x} \mathbf{v}_{i_x} + \mathbf{u}_{i_y} \mathbf{v}_{i_y}) \\ -(\mathbf{u}_{i_x} \mathbf{v}_{i_x} + \mathbf{u}_{i_y} \mathbf{v}_{i_y} - r)(-\mathbf{u}_{i_x} \mathbf{v}_{i_y} + \mathbf{u}_{i_x} \mathbf{c}_y + \mathbf{u}_{i_y} \mathbf{v}_{i_x} - \mathbf{u}_{i_y} \mathbf{c}_x) \end{vmatrix}$$

The solution vector may then be determined:

$$K = \frac{B_{21} - \frac{B_{11}}{A_{12}} A_{22}}{A_{21} - \frac{A_{11}}{A_{12}} A_{22}}$$

$$\theta = \frac{B_{21} - \frac{A_{21}(B_{21} - \frac{B_{11}}{A_{12}} A_{22})}{A_{21} - \frac{A_{11}}{A_{12}} A_{22}}}{A_{22}}$$

and the correction is:

$$P_n(n) = \left(x_{n-1} + K\mathbf{v}_x \text{ , } y_{n-1} + K\mathbf{v}_y \text{ , } \theta_{n-1} + \theta \right)$$

where the subscript of *n* denotes the current matching session.

If all target lines are found to be parallel, matcher2 is used. However, since the problem may be of type (1) above, moving the data points closer to the model may make the ‘small step’ assumption valid as the targets become non-parallel. It is for this reason that we test for an impending singularity (parallel targets) before each iteration of the algorithm.

V. Experiment

The formal testing of the sonar matching algorithm was accomplished by mapping a room, gathering sonar data, corrupting it, applying the matcher, and analyzing the results. From these results we hoped to characterize the performance of the matching algorithm under different conditions, in different rooms, and under bounded but random odometry errors.

Figure 22 illustrates the steps used to test the matcher algorithm.

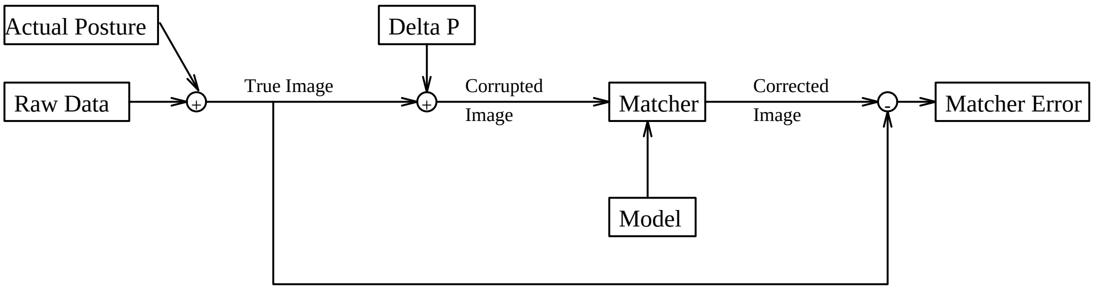


FIGURE 22.

First, a blueprint was secured from the architecture department with the physical plant of PSU. The origin was arbitrarily chosen as the NW corner of the building with zero degrees pointing south. Next the walls of a section of the building was turned into line segments by finding the coordinates of their endpoints. The segments representing a room is called the model. It was found that some walls on the blueprint were inaccurate and others appeared to be simply in the wrong place. This showed that with a building with many temporary partitions and multiple reconstructions, drawing cannot be relied upon wholeheartedly and must be verified using a tape measure and a friend.

Since the scope of this work did not include generating, searching, or retrieving line segment models from the 'world database', applicable lines were manually chosen and entered into a file for each scan location.

The next step was to move the robot to a known location and perform a sonar scan. The location is called the 'actual posture'. These scans were then transferred from the PC to the Unix system for analysis.

To analyze the algorithm we needed to induce positional errors into the data to see how the matcher performed in spite of them. Since all the points of a particular scan were taken from the same place (taken without the robot moving), a simple shift and rotation of the image did just that. The amount of shift in x and y and the amount of rotation (θ) about the robot's location is called the induced Δ 'Delta'. We define its components as: $\Delta P = (\Delta x, \Delta y)$ and the change in θ as $\Delta\theta$.

The matching algorithm may then be employed on the corrupted data in an attempt to negate the induced Δ. The difference between the result of the matcher and the actual posture is called the 'error' of that matching session. It should be noted that a Δ of (0,0,0) does not, in general, yield an error or (0,0,0).

Figure 23 shows a ΔP grid with an assigned $\Delta\theta$ of $20^{(de}$ and how an image is corrupted (shifted and rotated) by adding a particular ΔP and rotating by $\Delta\theta$.

Matching each corrupted image from each setting of each room produces a seeming myriad of relationships between which relationships could be shown. For example: How does the Δx (or Δy or $\Delta\theta$) affect the ability of the matcher to converge; how does the number (or closeness or length or one-

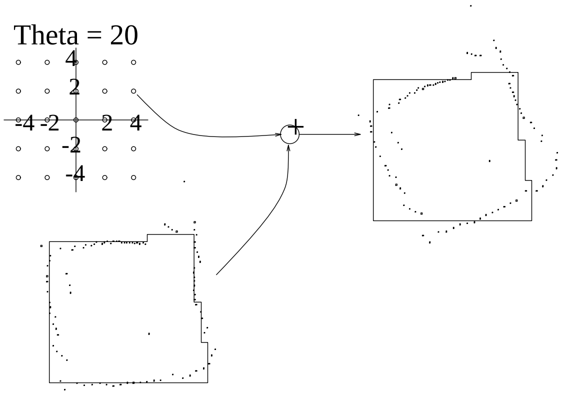


FIGURE 23.

sidedness) of the lines affect the convergence (or average *x*,*y* or θ errors) of the matcher; how does the change in the parameter *F* in the noise-removal module affect the convergence (or *x*,*y*, or θ errors) of the matcher; and so on. Worse yet, once we tabulated and graphed these relationships, how would we interpret the results? Fundamentally, however, the question at hand is whether the matcher is capable of keeping the robot on course; its ability to converge on the correct minimum in the presence of odometry-based errors.

So to draw conclusions we analyze the data using two methods: We look at the maximum and average errors versus the arc length of Δ*P* for various Δθ's; and we look to see where the algorithm converges in a particular room for various Δθ's. Finally, we analyze the complexity of the algorithm and perform some benchmarks.

The final and ultimate test of the matcher is to combine it with a pilot algorithm on the robot and observe how it maneuvers within the building using its odometry and occasional matching for navigation.

Expected Results

Since the matching algorithm works by converging on the nearest minimum sum of mean squared errors, we expected to see a small final error when inducing small Δ*P*'s, and large errors for large Δ*P*'s. The relationship was not expected to be linear but binary: either the matcher converges on the correct local minimum or it converges on the wrong one (yielding a larger error). The expected graph of absolute error versus Δ*P* is shown in figure 24. Actual results are shown in a later section.

Types of Rooms

Four types of rooms were used in our analysis of the matcher: small rectangular, large rectangular, corridor (two parallel walls), and a combination of the above. Each type has associated good and bad points which are discussed below.

A small rectangular room is one in which the matcher performs the best. Its small size allows the sonar device to locate all four walls, yielding a well-founded match. It was found that the matcher often converged correctly despite very large translations. A problem with rectangular rooms is that a rotation of the image by 45° or more often caused the matcher to converge on a correction whose heading was off by

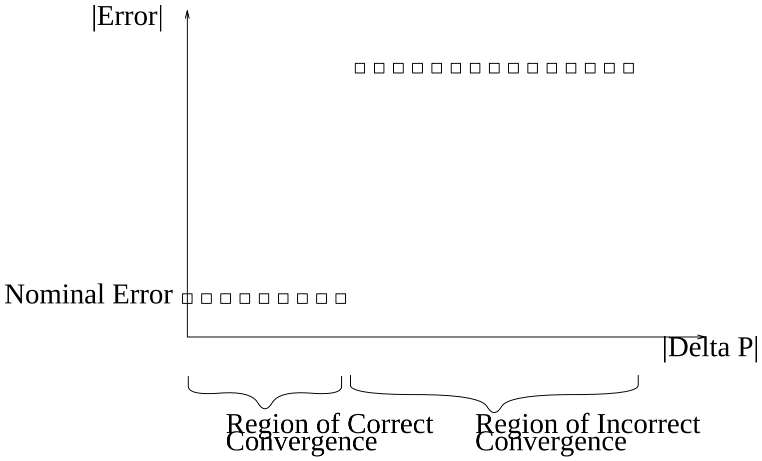


FIGURE 24. Error Versus Delta P

90°.

Large rectangular rooms present the additional problem of range. If walls are too far to be reached by the sonar, somewhat shorter distances are recorded which must be removed as noise. Also, if only one wall is found with the sonar, the distance along the wall remains unknown until a non-parallel wall is reached. This is the problem of parallel (single) target lines as seen in the next room type. Another more subtle characteristic of scans taken in larger rooms is that a small error in heading moves distant readings proportionally far from their true location. The result is lower rotational noise immunity.

Parallel walls present a situation where the position along the hall is non-deterministic and is assumed to be the position given by the odometry. However, $\Delta\theta$ may be as large as 90° without an incorrect convergence.

Intersecting corridors and other complex rooms are perhaps the most difficult to match due to the fact that a small ΔP can place points close to the wrong side of either or both hallways leading to an incorrect convergence. In other words, there are many false local minimums near the correct one when many mutually close parallel lines make up the local line segment model.

One coule certainly produce other room types from which additional conclusions could be drawn, but we chose to limit our observations to the simple-yet-complete types shown for simplicity. We also believe that these rooms are characteristic of those most often occuring in ‘real’ buildings. This is especially true of corridors through which the robot would spend much of its time navigating.

Analysis of Matching Errors

We begin our presentation of the results of the simulations by using several concrete examples from which we can begin to evaluate the matching algorithm in practice. All four types of rooms are represented with various Δ’s used.

10. Room 0

10.1. Correct Convergence

We begin with a small rectangular room. This is the basement lab of Dr. Perkowski's in which much of the PSUBOT was developed. The room has shelves covering nearly 3 of the walls and other assorted clutter which was not entered into the model of the room. The wall on the left is smooth, and in the scan shown, I am standing near the doorway which is near the right side of the bottom wall. First we show the results of the matcher with $\Delta = (-5,-5,0.35)$; i.e. the estimated position is wrong by 5 feet in both x and y directions, and the believed heading is wrong by 20°. See following three figures.

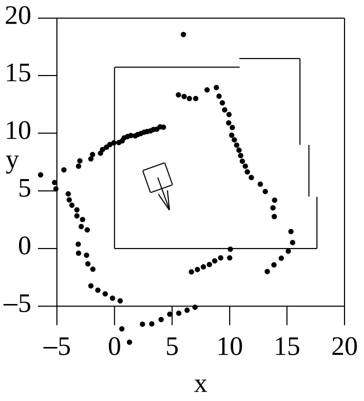


FIGURE 25. Shifted and Rotated Image

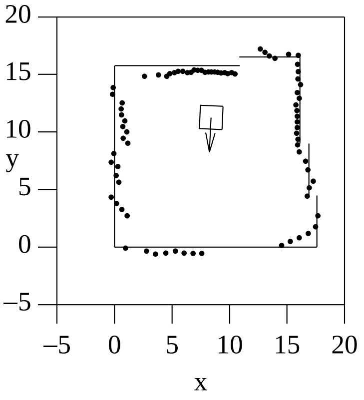


FIGURE 26. Matched Image Without Noise

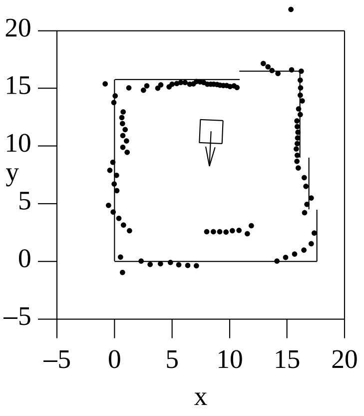


FIGURE 27. Final Matched Image

The second image shows the data considered just before the last correction. From it we can see that the noise removal algorithm was successful in eliminating those points representing the person standing near the wall. Table XXX tabulates the statistics of the matching session shown.

We can see from the table that initially there were many points dropped, as there were many points far from their targets. As they came closer to their target lines the number was reduced to those which were truely noisy or simply wrong (for a distinction see Chapter IV/Removal Of Outliers). Notice also, that the process terminated when the maximum change in *x* and *y* was less than 0.2 feet and the maximum

$\Delta=(-5,-5,0.35)$, $F=1$

				Correction		
Iteration	Average Dist	Std Dev of Dist	Pts Dropped	x	y	θ (rad)
1	-0.628255	5.74415	36	2.72101	2.13262	0.01138
2	0.894017	4.52347	31	1.12595	1.20243	-0.150398
3	1.25128	4.64336	29	0.57043	1.05755	-0.115745
4	1.27601	4.20551	19	0.206124	0.571806	-0.101845
5	1.31315	3.90068	15	0.026033	0.133716	-0.041724
Total				4.64955	5.09812	-0.398332
Error				-0.350452	0.098124	-0.048332

change in θ was less than 0.1. Faster matching sessions may be achieved by changing the termination condition at an expense of accuracy.

As can be seen both pictorially and numerically the matching algorithm found the correct orientation of the robot dispite the initial error. shown.

10.2. Incorrect Convergence

We now demonstrate a situation where the matcher converges on the wrong local minimum. If $\Delta=(5,-5,0.35)$ the matcher continues to rotate the image further rather than rotate it back. The result is a relatively good (yet incorrect) match for this rather square-like room as it is similar after rotating 90°. Also notice that it takes many more iterations for the algorithm to settle. This is because of the psudo-rotation used in the derivation of the sum of least squares equations. It is found that large rotations require many iterations.

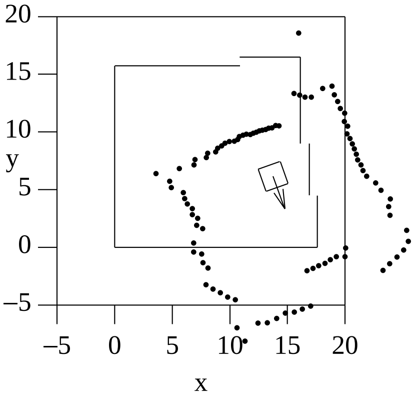


FIGURE 28. Shifted and Rotated Image

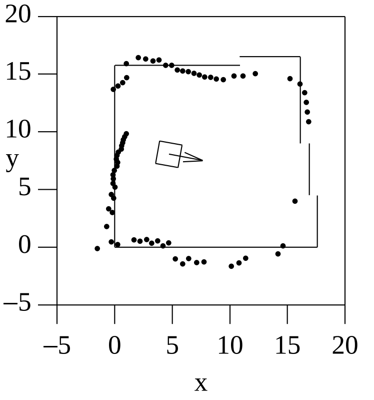


FIGURE 29. Matched Image Without Noise

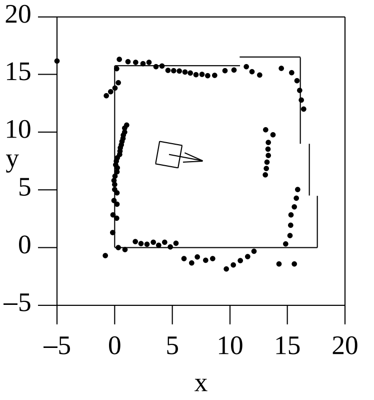


FIGURE 30. Final Matched Image

Looking at the two matching sessions we can see that for this image the matcher worked for $\Delta=(-5,-5,0.35)$ but did not work for $\Delta=(5,-5,-.35)$. From this we see that convergence depends somewhat on the direction of displacement and not on its magnitude only (the magnitudes were exactly the same).

We forgo showing incorrect matchings for the remainder of the rooms. Our purpose in this section is to give the reader a feel for what types of images can be expected from various room, not to show every possible matching session.

$\Delta = (5, -5, 0.35)$, $F = 1$

					Correction	
Iteration	Average Dist	Std Dev of Dist	Pts Dropped	x	y	θ (rad)
1	2.92486	6.31638	40	0.127534	0.802872	0.110878
2	3.94704	6.08292	37	-1.22332	-0.041724	0.023026
3	3.71266	6.52362	40	-1.11634	0.635029	0.054278
4	3.33529	6.69707	42	-1.11096	0.978665	0.103298
5	3.35244	6.13868	42	-0.624043	0.500646	0.042937
6	3.62995	5.56763	38	-0.591566	0.168663	0.058923
7	3.48851	5.82722	39	-0.395784	-0.001905	0.029393
8	3.18915	4.99423	34	-0.681239	-0.062811	0.051554
9	3.12735	5.51086	37	-0.65641	-0.123027	0.091193
10	2.44497	6.02809	37	-0.755652	-0.343665	0.090703
11	1.39015	6.05886	36	-0.665905	-0.429504	0.079256
12	0.962728	5.48436	33	-0.46129	-0.093625	0.05585
13	0.926138	5.21210	35	-0.299728	0.100147	0.047837
14	0.954267	5.13532	34	-0.500692	-0.179407	0.103302
15	0.023358	3.74207	25	-0.092678	-0.020519	0.094662
Total				-9.04808	1.88983	1.03709
Error				-4.04808	-3.11017	1.38709

11. Room 1

The second room demonstrated is a large rectangular room. It is the main lobby of the PCAT building. It is uncluttered except for some chairs which are in a line about 4 feet the walls on the right and left. There are also some short tables which do not enter into the image because of their height. The walls behind the chairs are nearly ideal for sonar. They have vertical slats spaced about one inch apart which allows the ultrasonic pulse to bounce back to the transmitter. As an added bonus corkboard lines the back side of the slats for a reduction in echo.

Figure 31 shows the robot near the double doors. Several people were watching as the scan was made, as can be seen from the apparent wall above and to the left of the robot. Notice that most of this noise was removed in the second image.

12. Room 2 Next we demonstrate the matcher in a corridor. To the matcher, there are only two parallel target lines, so it uses its parallel (two dimensional) solution. This means that the matcher will correct the position of the robot only along the corridor; in the y direction.

As this corridor is made up of glass on both sides, it proves to yield noisy scans. Since it is long and narrow and due to the spreading effect of the sonar beam, long distances along the hallway are nearly always shown as much shorter than they really are. The result is what looks like a very narrow rectangular room.

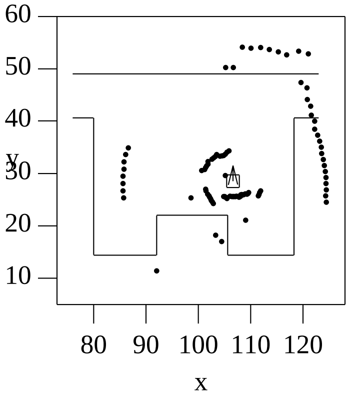


FIGURE 31. Large Rectangular Room

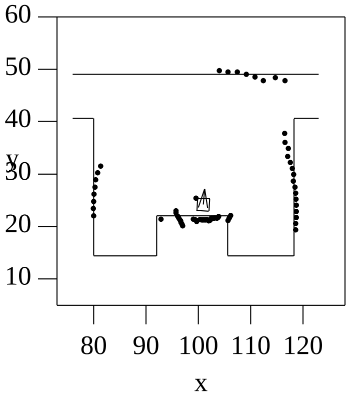


FIGURE 32. Matched Image Without Noise

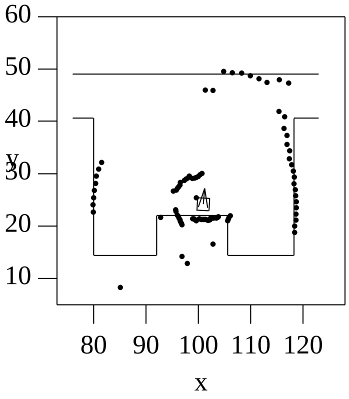


FIGURE 33. Final Matched Image

Also note that the algorithm is more tolerant of noise in this setting than in the previous ones. This is because the standard deviation of the distance from each point to its target line is much larger. However, that noise which remains after filtering tends to be cancelled out by a point from the opposite wall.

Here are the three graphs to demonstrate the matcher:

$\Delta=(6,4,0)$, $F=1$

				Correction		
Iteration	Average Dist	Std Dev of Dist	Pts Dropped	x	y	θ (rad)
1	1.34258	12.0981	29	-5.62671	-4.11459	0.053811
2	-2.30444	12.1663	25	-0.007132	-0.195088	-0.101356
3	-1.92595	12.8647	25	-0.028048	-0.009131	-0.031843
Total				-5.66189	-4.31881	-0.079388
Error				0.338109	-0.318807	-0.079388

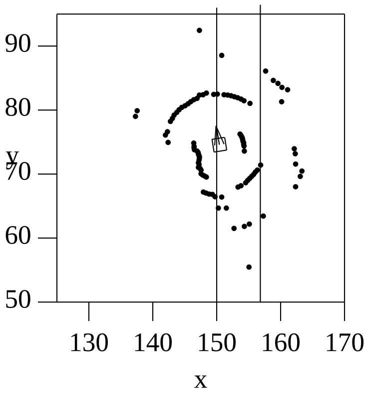


FIGURE 34. Large Rectangular Room

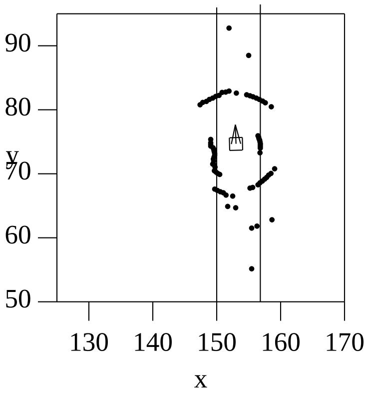


FIGURE 35. Matched Image Without Noise

Regions of Convergence

A large number of pictures and tables as shown above with varying Δ's would not be adequate in our attempt to characterize the matchers performance or to ensure with reasonable doubt that it would correct for reasonable odometry errors. So we chose to graph the region of convergence for various rooms. Such a graph would involve many matching sessions like those shown above, but with dots shown at those locations from which the matcher was able to converge. Relating this to the Δ grid shown previously, we show those points corresponding to the Δ's from which the final error is in near proximity to the error resulting from $\Delta P=(0,0)$. One could visualize it as a slice of the three-dimensional graph of the error versus ΔP.

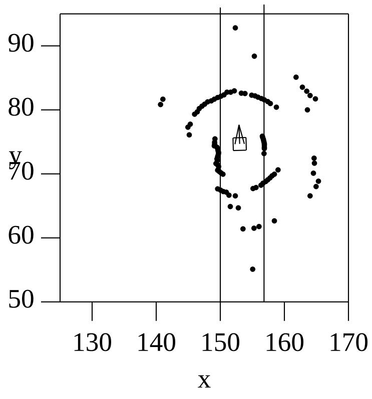


FIGURE 36. Final Matched Image

$\Delta=(-3,0,0.1745)$, $F=1$

Iteration	Average Dist	Std Dev of Dist	Pts Dropped	Correction x	 y	 θ (rad)
1	-2.27493	4.75576	27	2.02419	0.015018	-0.025477
2	-0.592822	3.94933	22	0.321128	0.020192	-0.033475
3	-0.398065	3.82609	21	0.228218	0.034148	-0.054835
4	-0.287945	3.79165	23	0.043171	0.013953	-0.021842
Total				2.61670	0.083311	-0.135628
Error				-0.383295	0.083311	0.038872

Since such a graph could be misleading - the matcher converged incorrectly even for $\Delta P=(0,0,0)$ - we include the error vector and angle with each diagram. The points are superimposed over the line-segment room and the robot's actual location is designated by a box. The actual heading of the robot can be seen by the solid arrow, and $\Delta\theta$ is shown as a dotted line.

References

Kerth88a.

Norman L. Kerth, ‘‘MOOD: A Methodology for Structured Object Oriented Design,’’ *Tutorial presented at OOPSLA 88, San Diego*, 1988.

Cox91a.

Ingemar J. Cox, ‘‘Blanche-- An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle,’’ *IEEE Transactions on Robotics and Automation*, pp. 193-204, April 1991.

Mayi91a.

Dieudonne Mayi, ‘‘Using Computer Vision for the Robot Localization Problem,’’ *Northcon 91*, October 1-3, 1991.

Lozan79a.

T. Lozano-Perez and Wesley, M. A., ‘‘An Algorithm for Planning Collision-Free Paths Among

Polyhedral Obstacles,'' *Communications of the ACM, vol. 22, no. 10*, pp. 560-570, October 1979.

Lozan81a.

Lozano-Perez, T., ''Automatic Planning of Manipulator Transfer Movements,'' *IEEE Transactions on Systems, Man and Cybernetics*, vol. 11(10), pp. 681-698, 1981.

Elfes87a.

Alberto Elfes, ''A Sonar-Based Real-World Mapping and Navigation,'' *IEEE Journal of Robotics and Automation*, vol. RA-3, No. 3, pp. 249-265, 1987.

Khati86a.

Khatib, O., ''Real-Time Obstacle Avoidance for Manipulators and Mobile Robots,'' *International Journal of Robotics Research*, pp. 90-98, 1986.

Kodit89a.

D. E. Koditschek, ''Robot Planning and Control via Potential Functions,'' *The Robotics Review 1*, pp. 349-368, 1989.

82a. O'Dúnlaing, C. and Yap, C.K., ''A Retraction Method for Planning the Motion of a Disc,'' *Journal of Algorithms*, vol. 6, pp. 104-111, 1982.

Latom91a.

Jean-Claude Latombe, in *Robot Motion Planning*, Kluwer Academic Publishers, 1991.

Matar90a.

Maja J. Mataric, ''A Distributed Model for Mobile Robot Environment Learning and Navigation,'' MIT Technical Report #1228, May 1990.

Brooka.

Anita Flynn & Rodney Brooks, *Battling Reality*, MIT AI Memo 1148.

Flynn89a.

Anita M. Flynn, Rodney A. Brooks, William M. Wells III, and David S. Barrett, ''SQUIRT: The prototypical Mobile Robot for Autonomous Graduate Students,'' MIT AI Memo# 1120, July 1989.

Crowl89a.

Crowley, J.L., ''World Modeling and Position Estimation for a Mobile Robot Using Ultrasonic Ranging,'' *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, pp. 674-680, 1989.

Lumel87a.

Vladimair J. Lumelsky and Alexander A. Stepanov, ''Path-Planning Stratevies for a Point Mobile Automation Moving Amidst Unknown Obstacles of Arbitrary Shape,'' in *Algorithmica*, Springer-Verlag, New York, 1987.

Espin1.a.

Cecilia Espinosa and Marek A. Perkowski, ''Hierarchical Hough Transform for the Vision System of a Wheelchair Robot,'' *Northcon 91*, October 1-3, 1991..

Evere89a.

CDR H. R. Everett, ''Survey of Collision Avoidance and Ranging Sensors for Mobile Robots,'' *Robotics and Autonomous Systems*, vol. 5, pp. 5-54, Elsevier Science Publishers B.V., North-Holland, 1989.

Elfes86a.

Alberto Elfes, ‘‘A Sonar-Based Mapping and Navigation System,’’ *Proc of 1986 IEEE International Conference on Robotics and Automation*, pp. 1151-1156, 1986.

Cox89a.

Ingemar J. Cox, ‘‘Blanche: Position Estimation for an Autonomous Robot Vehicle,’’ *Proceedings of the IEEE/RSJ International Workshop on Robots and Systems (IROS) ’89)*, pp. 432-439, Sept. 4-6, 1989.

Iijim81a.

J. Iijima, S. Yuta, and Y. Kanayama, ‘‘Elementary Functions of A Self-Contained Robot - YAMA-BICO 3.1,’’ *Proceedings of the 11th International Symposium on Industrial Robots*, pp. 211-218, Tokyo, 1981.

Wang88a.

C. Ming Wang, ‘‘Location Estimation and Uncertainty Analysis for Mobile Robots,’’ *International Conference on Robotics and Automation*, pp. 1230-1235, April 24-29, 1988.

Nelso88a.

Winston L. Nelson and Ingemar J. Cox, ‘‘Local path Control for an Autonomous Vehicle,’’ *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, pp. 1504-1510, 1988.

Guttm82a.

Irwin Guttman, S. S. Wilks, and J. Stuart Hunter, *Introductory Engineering Statistics,* Wiley Seriew in Probablity and Mathematical Statistics-Applied, John Wiley & Sons, 1982.