

Path Planning with Phased Array SLAM and Voronoi Tessellation

Jonathan Tse, Eric VanWyk, and James Whong
Olin College

Abstract—Autonomous vehicles must often navigate environments that are at least partially unknown. They are faced with the tasks of creating a coordinate system to localize themselves on, identify the positions of obstacles, and chart safe paths through the environment. This process is known as Simultaneous Localization and Mapping, or SLAM. SLAM has traditionally been executed using measurements of distance to features in the environment. We propose an angle based methodology using a single phased array antenna and DSP, aimed to reduce this requirement to a single path for each data type. Additionally, our method makes use of rudimentary echo-location to discover reflective obstacles. Finally, our method uses Voronoi Tessellation for path planning.

I. INTRODUCTION

The problem of localization and mapping in unknown environments is becoming increasingly important in the field of autonomous vehicles. As technology in the field develops, new sensors are integrated to form a more accurate picture of the world around the vehicle. Rather than integrating yet another high end sensor, this paper puts forth a simple sensing methodology that requires very low-tech sensor technology. In fact, these sensors may already exist on some system deployments as part of their communications package, and may be retrofitted with a software update.

We propose a variant of beacon based localization. In this setup, a beacon is a deployable stationary marker fitted with an electronics package. These beacons may be deployed by the vehicle or through some other mechanism. However, the beacons must remain stationary after deployment. Once deployed, a beacon begins emitting a modulated signal on a previously agreed frequency. This signal may be either sonic or electromagnetic in nature, and may be used to encode information. In our system, the vehicle may not send any information to a deployed beacon.

As the vehicle moves, it takes snapshots of its position relative to the deployed beacons in order to create a sense of “absolute” position. As it does so, information from sensors are laid upon an internally generated map. Our method combines these steps and presents a method by which to simultaneously discover information about the location of a beacon and the location of any obstacles in the environment.

Our system can be broken down into three main steps. The first step is to find the angle or bearing to any beacons or obstacles within range at any given time. The second step is to integrate this data with information collected from previous positions to

create a full map of the environment. The third step is to use this map to navigate the environment while avoiding obstacles.

II. FINDING BEARING

Finding the bearing to the beacon can be done in several ways. The most obvious is using a rotating antenna, much like a radar dish. However, this method is only capable of examining a single section of its sweep at any given time, and the sweep speed is limited by the physical rotation speed. Because of this limitation, a phased antenna array is a more viable solution.

A. Phased Antenna Arrays

A phased antenna array is a collection of n physically distinct omnidirectional antennae that work together to create a single virtual antenna that has properties that would be otherwise unobtainable. Traditionally, if the array is being used to transmit a signal, software would determine the virtual antenna’s output based on the sum of the signals sent from each physical antenna. Since the antennae are spatially disparate, there will always be a slight phase shift between the signals received from each physical antenna at some location distant from the array. By using software to adjust the phase of the output signal of each physical antenna, the combined output signal can be made to focus on a signal bearing. The basic mechanism behind this is the selective use of constructive and destructive interference. By changing the signal phase of each antenna in the right way, the output signals from all the antennae can be made to sum constructively in only one direction and destructively in all the rest. Effectively this forms a gain pattern, where the signal has been attenuated due to destructive interference in all but a narrow cone. The result of changing this gain pattern is a virtual directional antenna that can be “swept” without requiring any physical motion.

We may apply this principle in reverse using the same hardware, but with different software. Given a signal received on a phased antenna array, we can find the bearing to its source. If we know the time of flight from each antenna to the beacon, this becomes quite trivial. Take any two physical antennae and form a triangle with the antennae forming two vertices and the source, or beacon, forming a third. We know the distance between antennae and can find the distances to the beacon by combining the time of flight information with knowledge of the velocity of the signal in the transmission medium. Since each of the lengths of this

triangle are known, simple application of the law of sines will tell us all angles involved.

Unfortunately, knowledge of the absolute time of flight requires some more complex methods:

- 1) Send-Receive Interaction - The phased array antenna radiates energy to “ask” for a signal, and starts a timer. The beacon receives the signal from the phased array, and responds. By measuring the time this interaction takes, we can extract the time of flight by subtracting the beacon’s latency and processing time and dividing by a factor of two.
- 2) Multiple Signals - If the beacon simultaneously sends different signals that have different, known propagation velocities, measuring the difference in time between receipt of the signals will give the time of flight. For example, the beacon simultaneously emits a radio wave and an ultrasonic pulse. This method is essentially the same as the method of estimating the distance to a storm by timing the interval between the lightning and the thunder.
- 3) Strict Timer Synchronization. - The beacon and receiving phased array have synchronized timers, and the beacon agrees to send a signal at pre-described intervals. This gives an exact measure of time of flight, but often requires sophisticated timers, such as the atomic clocks found on GPS satellites.

However, we are restricting ourselves to only being able to receive signals, and we have no way of guaranteeing timer synchronization, so we are limited to knowledge of the relative, not absolute, time of flight information. This means that we only know that antenna A_i is Δd meters further from the beacon than antenna A_j . This gives our previously fully constrained triangle a single degree of freedom that we can not resolve.

B. Finding ϕ from Δd

Assuming we have two antennas, as shown in Figure 1, we can determine the bearing to the beacon. Note that the distance to the beacon is d and the distance between the antennae is a . From the figure, $\Delta d < a$, because a is the hypotenuse of a right triangle. Furthermore, we may assume that $a \ll d$ due to the geometries involved: a is constrained by the size of the vehicle, and d is constrained by the size of the environment within which the vehicle exists. Thus we can make the assumption that the lines to the beacon are essentially parallel, and this allows us to create the right triangle as shown in Figure 1. Since we know a and Δd , we may find ϕ using trigonometry.

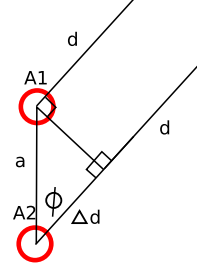


Fig. 1. Right triangle representing the signals received by the pair of antennae, A_1 and A_2 . a is the known distance between A_1 and A_2 , d is the distance to the beacon, and ϕ is the bearing to the beacon.

Note that this creates two possible solutions: ϕ and $-\phi$. With a pair of antennae it is impossible to determine which is the correct bearing. However, adding a third antenna creates two new antenna pairs, each of which will create two possible solutions. Of the 6 total solutions, it becomes a simple matter of determining which vectors are parallel to select the correct bearing ϕ .

From here, we may find the bearing ϕ to any radiation source given that we can discover the Δds involved. The algorithm is essentially reduced to finding these Δds reliably despite interference. Figure 2 shows an example solution set and the correct result.

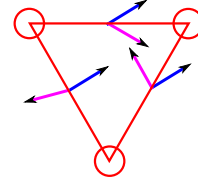


Fig. 2. Phased array of three antennae showing the six possible solutions for ϕ . The parallel—correct—solutions are shown in blue.

C. Finding Δd without Interference

For clarity, assume an infinite expanse with no reflective surfaces and no sources of radiation other than a single beacon. This beacon repeatedly radiates a sinusoid of known frequency for a given amount of time and then is silent.

Ideally, Δd would be discovered measuring the time difference in the start of the received signals at each antenna. Unfortunately, this method demands very good measurements of the received signal. The accuracy of Δd is based on to the velocity of the medium and the sampling frequency of the D/A converters. This quickly becomes infeasible using available components. The information available in that single moment of time is too hard to retrieve.

To mitigate this error, we may extract information from the signal over the entire course of the pulse by using phase information. Given two phasors Ψ_i and Ψ_j we may find $\Delta d_{i,j}$ as follows:

$$\Delta d_{i,j} = (\angle \Psi_i - \angle \Psi_j + 2\pi n) * v$$

If the distance between A_i and A_j is less than one wavelength, we can assume that $n = 0$. We now place this constraint on the system designer, and assume that $n = 0$ is valid henceforth.

The accuracy of this method is dependent on the number of samples collected, and therefore on the product of both the sampling frequency and the amount of time spent sampling. Given an infinite sampling time, Ψ is known with nearly infinite precision.¹ However, it is possible to estimate Ψ in a finite time to a reasonable precision beyond which continued sampling is unproductive. This time interval is highly dependent on the exact nature of the specific implementation, and we will therefore encapsulate the value in an arbitrary variable s .

We may now find Δd , and therefore ϕ , assuming that we can receive the beacon's signal for s units of time.

D. Echoes

Assume now that a single reflective surface is added to the aforementioned infinite planar expanse. This infinitely long wall divides the expanse into two regions, and we assume that the receiver and the beacon are located on the same side of it. This reflector now creates a second signal that is a time delayed and attenuated version of the original signal. Let us label this delay as Δt . Recall that the method outlined in section II-C is amplitude invariant, and therefore we may disregard the attenuation due to the reflection.

Since an echo is necessarily of the same frequency as the original signal, the echo combines with the original signal by superposition to form a "ghost" signal that appears to originate from an entirely different direction, as shown in Figure 3.

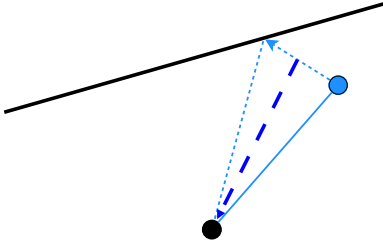


Fig. 3. One beacon (blue), one reflective surface, and a receiver (black). Note the dotted lines representing the multipath signals resultant from the echo from the wall. The bold dotted signal vector represents the "ghost" signal from the superposition of the two received signals.

Given our current setup, this effect is unavoidable, and we must therefore concentrate on the times when either the echo signal or the original signal have are "at" the receiver, but not both. If $\Delta t > s$, this is quite trivial. The beacon transmits a signal for some time $t_B | \Delta t > t_B > s$. The receiver processes the original signal, waits for $\Delta t - t_B$ and then processes the echo. This would give us bearing to the beacon and to the point of reflection. As shown in Figure 4, the range for which this method works is modeled as an ellipse with foci at the beacon and receiver with a minor axis length of $s * v$. If a reflective surface is contained within the boundary ellipse, Δt will be smaller than s and will therefore create irresolvable interference. Conversely, if there are

¹If the sampling frequency is an exact multiple of the carrier frequency, it is impossible to fully resolve Ψ

no reflective surfaces within the ellipse, Δt is guaranteed to be larger than s , which means that there is sufficient time to gather the requisite number of samples for finding bearing.

However, this requires the beacon have knowledge of Δt , which changes depending on the location of the beacon, receiver, and reflective surface.

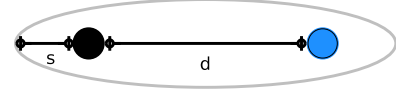


Fig. 4. One beacon (blue) and a receiver (black).

E. Short Echo Paths

In order to be able to function within the ellipse of interference shown in Figure 4, we must further modulate the signal. This amounts to transmitting for some amount of time less than Δt , waiting for the echo to clear, and transmitting again. This works if we may assume that the beacon's oscillator continues to run and keep time even if the antenna are not transmitting. The receiver's software can then separate the received signal portions into the appropriate "bins", combine appropriate packets of this data and estimate ϕ using the combined data. In this manner, it is possible to achieve arbitrary resolution given a sufficient number of uninterrupted signal portions. For the purposes of this paper we will ignore the costs of aligning the start and end of each portion, as they may be accounted for by taking more packets.

The original concept of s now becomes a measure of how many portions or packets of signal are required to establish ϕ to a given accuracy. Also, we have created the concept of a modulation code that is used to time the presentation of these packets. Let us denote $M(s, q)$ to be a set of modulation codes that:

- 1) Are composed of a binary string that directly corresponds to a binary amplitude shift keying (ASK) mechanism. Without loss of generality, assume that a 1-bit indicates transmission of the sinusoid, whereas a 0-bit indicates silence.
- 2) Have bit duration T_b . In other words, ASK is toggling the sinusoid on or off at a frequency of $f = \frac{1}{T_b}$. Note that T_b must be less than half the period of the carrier wave in order to have enough resolution to be effective.
- 3) Require at least the first q bits to be free of echoes. In other words, this code requires that the vehicle never be within $T_b q v$ of a reflector.
- 4) Guarantee that at least s bits will be received clear so they may be used in the previously outlined algorithms.
- 5) Reduces the boundary ellipse's major axis to $q f v$ from $s v$. This implies that codes are only "useful" in reducing boundary ellipse size if $q < s$. However, codes with $q \geq s$ can be useful if they allow the receiver to distinguish between sources.

Assuming that we can find a set $M(s, q)$, we can deploy $B = |M(s, q)|$ beacons, each transmitting one of the codes within M as their identification marker.

F. Using Modulation Codes

Assume for a moment that we have found a set of codes that satisfies $M(s, q)$, and let each code within the set be represented as M_1, M_2, \dots, M_n . Place n beacons in some random positions in a planar expanse containing one reflective wall and one receiving node. Each beacon is assigned a unique code from M by whatever mechanism placed them. We may assume that some mechanism further ensures that only one beacon is transmitting at any given time. Details of this time slotting mechanism are not within the scope of this paper, but it could be as simple as a carrier sense algorithm running on each beacon.

Depending on the geometry of the situation, beacon position may be categorized into one of three types of locations. Refer to Figure 5 for a representative sample positioning of these beacons.

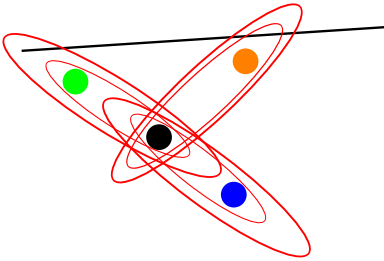


Fig. 5. Three beacons (blue, orange, and green) and a receiver (black). Outer bounding ellipses indicate the minimum spacing to obstacles without using modulation. Inner ellipses indicate minimum spacing to obstacles using modulation.

The blue beacon is in optimal position. As shown by the boundary ellipses, there are no reflective surfaces close enough to create an interfering echo pattern. Therefore, the receiver sees a clean copy of M_1 , a period of dead air, and then another clean copy of M_1 . The bearings of the beacon-generated signal and echo-generated signal can be easily found by using the algorithm described in II-C.

The orange beacon is in a location that makes its data irretrievable. The reflective surface is within its inner boundary ellipse. This implies that the echo will have a Δt less than qf , which violates a previously established requirement.

The green beacon is in a position that optimally shows the benefit of the modulation codes. The wall intersects its outer boundary ellipse, which implies that a naive non-modulated method would have insufficient time to establish bearing. However, the wall does not intersect the inner boundary, which implies that the modulation allows us to pick out sufficient bits to reconstruct bearing.

Let us examine this process in depth now. To do so, we will use the trivial set $M(2, 2) = \{ '1011', '1101' \}$. See Table I for a listing of how these codes are received when there is a given amount of delay in the echo. This table was found by combining the signal with an appropriately shifted version of itself. If both the original and the echo were silent, a 0 was recorded. If either were impinging, a 1 was recorded. If both were impinging, a 2 was

recorded. Note that the 0 delay column is physically impossible, and merely is there as a reminder of the codes.

| Echo Delay | M_1 Recieved | M_2 Recieved |
|------------|----------------|----------------|
| 0 | 1011 | 1101 |
| 1 | 11121 | 12111 |
| 2 | 102111 | 111201 |
| 3 | 1012011 | 1102101 |
| 4 | 10111011 | 11011101 |
| 5 | 101101011 | 110101101 |

TABLE I
 $M(2, 2)$

Recall that at this stage we are unable to decipher bearing, and must identify these codes using only the presence or absence of a sinusoid. As such, all 1's and 2's appear identically. Given this constraint, we see that every element in the table is uniquely identifiable for Delay > 1 . This establishes that $q > 1$. Now that we know which message was intended to be sent, it is a simple matter to split the packets into their appropriate bins for phase finding. To verify s , we must find the minimum number of usable bits. Each usable bit is denoted by a 1, and with a simple scan of the table we see that all usable codes contain at least 4 1s. Since these usable bits are necessarily split between the original and the echo, we know that we are guaranteed that each bearing calculation has at least 2 usable bits. Therefore, $s = 2$. This completes the verification that $\{ '1101', '1011' \}$ satisfies $M(2, 2)$.

G. Finding Modulation Codes

Although verification of a given code set is computationally easy, it is rather difficult to elegantly create these codes in non-trivial cases. Brute force methods met a moderate deal of success, but they offer no convincing proof of validity. Generally the method is to take spread-spectrum codes and check them to ensure robustness.

We have begun looking into using the spanning fields of prime fields. For example, $\{0, 1, 2, 5\}$ spans $\mathbb{Z}_{\text{mod}7}$. This can create two forms of code: dense and sparse.

1) *Dense Prime Field Codes*: To create a dense code from a given spanning field, interpret each value in the field as the location of a '1' in a bit string. All other values are assumed to be '0'. Our previous example of $\{0, 1, 2, 5\}$ translates to '111001' in this way. Since we are guaranteed that this will never produce a symmetric code, we may also take its reverse '100111' as a code. This code happens to satisfy $M(2, 1)$. We will soon find that these spanning fields necessarily generate codes of $M(s \geq 2, 1)$.

Given a spanning field f each pair of elements within f are situated a unique distance from each other with the noted exception of the 0 value. Let us ignore the 0 position for the moment. Given an echo, we only generate a collision if the distance between a pair of "1"s is equal to the echo displacement. Since these distances are guaranteed to be unique, we generate only one collision. By reinserting the 0 position, we create one additional possible collision. Therefore, in a dense code we have

at most 2 collisions for any given echo displacement time. This implies that a given spanning field f of length l creates a pair of valid codes that satisfies $M(|f| - 2, 1)$

2) *Sparse Prime Field Codes*: In the case where we have more than 2 beacons, a sparse code may be used. In this case, we take a given field and find all of its permutations. Take this reordering to be a run-length coded version of the 0s in the code. For example, the permutation $\{2, 5, 1, 0\}$ becomes '100100001011'. Again, this satisfies $M(|f| - 2, 1)$ However, the set of codes generated now has a magnitude of $(|f|)!$. This allows the deployment of an unbounded number of beacons assuming that we can find spanning fields of sufficient size.

3) *More Codes*: This generation of codes is still in need of much development. The two methods shown have only been shown to provide codes, but are in no way guaranteed to produce optimal codes. An optimal code is one that requires the least amount of transmission time while satisfying its requirements for q and s . It is our belief that relaxing q will allow for shorter codes in general.

H. Results

By combining the ϕ finding algorithm with the echo de-tangling algorithm, we know the bearing to the source, the bearing to the echo location, and the difference in flight time of the two paths. This information alone is not sufficient for localization. The next phase of processing will take these three pieces of information to reconstruct range information.

Note that this assumes that Δt is a multiple of the bit length. This is an unrealistic assumption, as all objects are in general position. Creating codes that can handle bearing calculations with non-discrete echo times is easy - each of the codes shown support this already. However, creating codes that can be recognized individually with non-discrete echo times is a bit harder, and is left as an exercise for the reader.

III. OBJECT LOCATION AND BEARING

We start with a simple case, shown in Figure 6, of a few beacons, a reflective surface, and one receiver at time $t = 0$.

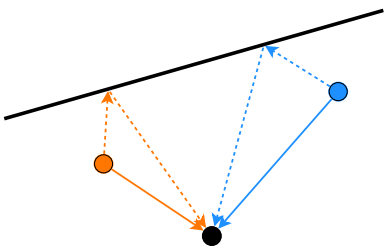


Fig. 6. At time $t = 0$, two beacons (orange and blue), one reflective surface, and a receiver (black). Note the dotted lines representing the multi path signals resultant from the echo from the wall.

The paths shown are the actual paths taken by the signals from the beacon. However, the only information we actually have is the bearing of each of the signal vectors we receive at $t = 0$,

giving us a situation as described by Figure 7. The signal vectors generated from the bearing readings have indefinite length. Other than the difference in time of flight, which may or may not be reliable, we cannot determine any other information from the bearing readings. In order to get more information about where the beacons are, we must employ some more sophisticated methods, such as spatial disparity over several timesteps.

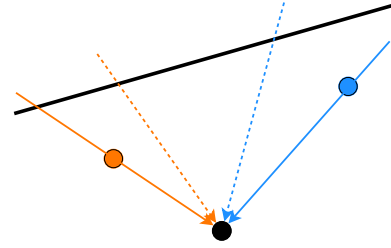


Fig. 7. At time $t = 0$, two beacons (orange and blue), one reflective surface, and a receiver (black). Note the dotted lines representing the multipath signals resultant from the echo from the wall. Also note that the signal vectors extend indefinitely, as we only have bearing information.

A. Locomotive Spatial Disparity

We can make use of spatial disparity by moving the receiver. So at time $t = 1$, we move the receiver a short distance. Because the beacon and reflective surface remain immobile in world space, the bearings of the signal vectors must change as the receiver changes position, as shown in Figure 8. Also from the figure, it is readily apparent that the signal vectors representing the direct beacon-to-receiver path always intersect at the beacon. At this point, we still do not have enough information to judge distances to any object, because by extending the signal vectors from the echoes indefinitely, we get another intersection that has an equal probability of being the beacon location, behind the wall. Also, depending on the location of the beacon, the wall, and the receiver, we can get up to two more signal vector intersections per beacon that all are equally likely to be the beacon at time $t = 1$.

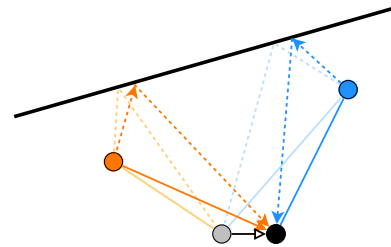


Fig. 8. At time $t = 1$, two beacons (orange and blue), one reflective surface, and a receiver (black or grey, where grey is the old position of the receiver). Note the dotted lines representing the multipath signals resultant from the echo from the wall. Rather than vectors of indefinite length, the signal vectors representing the actual path of the signals have been drawn for clarity.

Moving the receiver again, at time $t = 2$, allows us to begin to differentiate the location of the beacon from the false positions suggested by the echoes. As shown in Figure 9, all three direct

beacon-to-receiver signal vectors intersect at the beacon, whereas the echo-generated signal vectors from times $t = 0$ and $t = 1$ intersect at a different point than the echo-generated signal vectors from $t = 1$ and $t = 2$. However, there still is a chance that the beacon lies behind the wall, especially in the cases where the geometry causes the intersection of the echo-generated signal vectors to all intersect at points that are very close to one another.

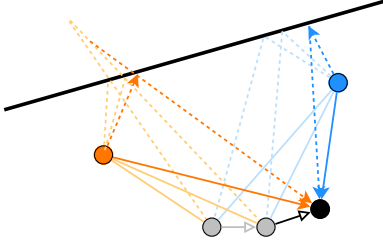


Fig. 9. At time $t = 2$, two beacons (orange and blue), one reflective surface, and a receiver (black or grey, where grey is the old position of the receiver). Note the dotted lines representing the multipath signals resultant from the echo from the wall. The orange signal vectors generated by the echoes from the wall have been extended until they intersect with one another. Note that they intersect at different points.

As the receiver continues to move around in worldspace, we continue to generate more signal vectors, as shown in Figure 10. Again, note that the vectors from the direct beacon-to-receiver path all intersect at the beacon, whereas the echo-generated vectors all intersect at varying points behind the wall. The easiest algorithm to implement now is to find the point at which the most vectors intersect, because that point has the highest probability of being the beacon. One way of implementing this algorithm is described in Section III-B.

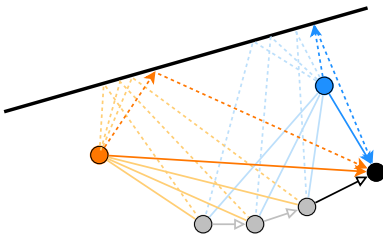


Fig. 10. At time $t = 3$, two beacons (orange and blue), one reflective surface, and a receiver (black or grey, where grey is the old position of the receiver). Note the dotted lines representing the multipath signals resultant from the echo from the wall. Rather than vectors of indefinite length, the signal vectors representing the actual path of the signals have been drawn for clarity.

B. Multipath Signal Rejection

The algorithm described here has been adapted from the one described in [1]. The original algorithm as described makes use of the range to beacons, instead of bearing information. As such, we have adapted the algorithm to use the range to an intersection of two signal vectors. For simplicity, we've chosen a 1-beacon example. This algorithm is then repeated for the remaining $n - 1$ beacons.

1) *Circle Construction:* The algorithm is based on the idea that the point at which the most vectors intersect is the beacon. One way to find this is to draw circles centered at the current location of the receiver with radii equal to the distance to each of the signal vector intersections, as shown in Figure 11. Since every circle represents the intersection of two signal vectors, the intersection of two circles suggests the possible intersection of some of the vectors used to generate the two circles. Recalling that the most heavily intersected vectors are the ones that go through the beacon, it follows that the most heavily intersected circles also are likely to go through the beacon.

Note that we must start at $t = 1$ because that is the first time t that allows for signal vector intersections. Also note that we are choosing circles centered at the position of the receiver at time $t = 1$. Since all the circles generated at any time t are concentric, we are guaranteed that they will not intersect with one another.

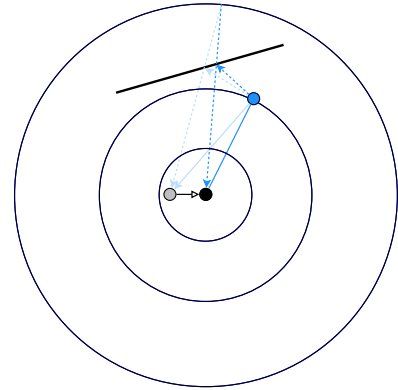


Fig. 11. Time $t = 1$. Circles have been drawn centered around the receiver position at time $t = 1$, with radii equal to the distances to each of the signal vector intersections. Note that one circle passes through the beacon.

As we advance the algorithm in time to time $t = 2$, we can see a similar set of circles centered at the position of the receiver at time $t = 2$, as seen in Figure 12. The largest circle in this particular graphic is a result of the two echo-generated signal vectors. In this particular geometry, the largest circle does not intersect with any other circles, as seen in Figure 14, again reinforcing the idea that the most heavily intersected circles are more likely to go through the beacon.

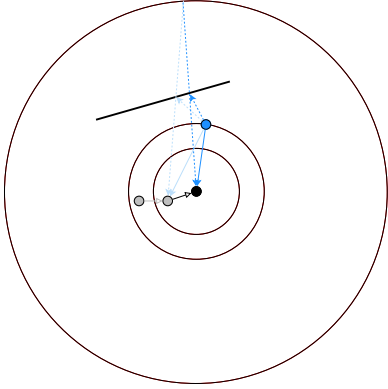


Fig. 12. Time $t = 2$. Circles have been drawn centered around the receiver position at time $t = 2$, with radii equal to the distances to each of the signal vector intersections. Note that one circle passes through the beacon.

For completeness, we show the $t = 3$ case in Figure 13. At time $t = 3$, there is one additional circle because the geometry of the beacon, receiver, and reflective surface generates one additional signal vector intersection. We will see that the algorithm is robust enough to handle this case.

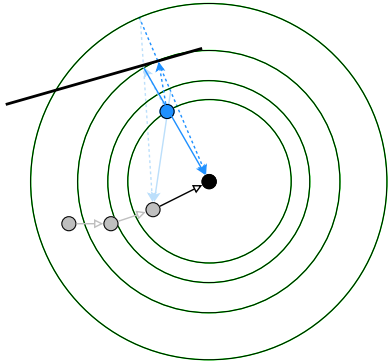


Fig. 13. Time $t = 2$. Circles have been drawn centered around the receiver position at time $t = 2$, with radii equal to the distances to each of the signal vector intersections. Note that one circle passes through the beacon.

2) *Circle Superposition and Intersection:* Having generated the circles, we can now superimpose them on one another, as seen in Figure 14. The next step of the algorithm requires us to find all the intersections between circles.

We must be careful to not make this step the most computationally difficult of the algorithm. There are many different algorithms that solve this problem, such as the algorithms described in [2], which make use of trigonometry and vector products to solve for the actual point of intersection of the circles.

Fortunately, we only need to determine whether or not the intersect at all, so we can assume a simple algorithm that makes use of the Cartesian coordinate system. If we have a circle centered at (x_1, y_1) with radius r_1 and another circle centered at (x_2, y_2) with radius r_2 , we only need to evaluate the following expression:

$$r_1 + r_2 \geq \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (1)$$

If Equation 1 is true, then we know the circles intersect because the sum of their radii is greater than the distance that the centers of the circles are apart.

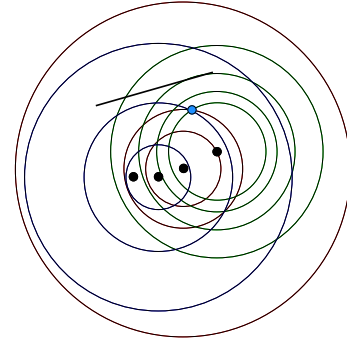


Fig. 14. Superimposed circles from $t = 1, 2, 3$. Note the heavy intersections between circles of different colors. Also note that all similarly colored circles are concentric and therefore do not intersect.

For clarity, we have highlighted in Figure 15 all the circles that go through the beacon in purple. Since there are three purple circles, each one of those circles is guaranteed to intersect with at least two other circles—the other two purple circles. Other circles, such as the largest circle in Figure 12, have no such guarantee, in fact, the circle from Figure 12 does not intersect with any circles at all.

Note that in the actual implementation of the algorithm we do not have the luxury of being able to select the circles that go through the beacon, as that would be assuming that you know what you're trying to find.

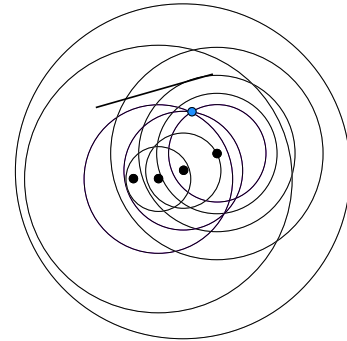


Fig. 15. Superimposed circles from $t = 1, 2, 3$. Note that the circles highlighted in purple all go through the beacon.

3) *Graph Construction and Filtering:* From Figure 14 and liberal application of Equation 1, we can construct a graph that will help us solve this problem. If we treat each circle in Figure 14 as a vertex in the graph, we can show each circle-circle intersection with an edge connecting the corresponding vertices.

Recall that at each timestep, all the circles generated by this algorithm are concentric, and thus do not intersect. Thus, we can draw the graph as a n -partite graph, where n is the number of timesteps we are examining. Since we are looking at times $t =$

1, 2, 3, we have a 3-partite graph, as shown in Figure 16. We can now make use of our knowledge that the most heavily connected circles are most likely the circles through the beacon.

Now we choose the vertex with greatest degree from each partite. Each one of these vertices represents the circle with the greatest number of intersections from that particular timestep, and thus the circle that goes through the beacon. However, in the case of the third timestep, represented as the green partite in Figure 16, there are multiple vertices with degree 4. In this case, the decision would be made based on which of those vertices is connected to the most chosen vertices in the other partites. In other words, which circle intersects with the most other circles that we think go through the beacon. Unfortunately, in this case, we cannot make a decision for the green partite, so we discard the data. As we said earlier, the algorithm is robust enough to discard faulty data. Thus, we know the medium sized circles from $t = 1$ and $t = 2$ go through the beacon.

With some modifications, the algorithm can be made robust enough to survive the case that there is an obstruction between the beacon and the receiver for one timestep. In other words, during one timestep there is no direct path from the beacon to the receiver. In this case, the decision scheme of choosing the vertex with highest degree in a partite is not sufficient, as this case allows for selection of an incorrect vertex. To solve this problem, we can use the original algorithm outlined in [1], which finds a subgraph of the graph in Figure 16 that maximizes connectivity. That is, the algorithm selects the most highly intersected circles, which is a more general version of the algorithm we have presented. One caveat is that the original algorithm from [1] requires more data points than ours does.

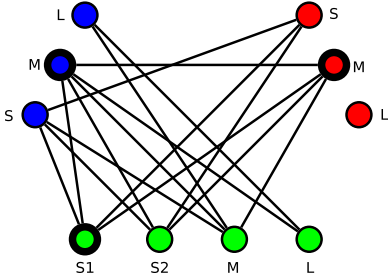


Fig. 16. Graph of circle intersections. Blue vertices are circles from time $t = 1$, red from $t = 2$, and green from $t = 3$. Vertices have been labeled S, M, and L, where S is the smallest circle, M the medium-sized circle, and L the largest circle. Note that there is an S1 and an S2 circle from time $t = 3$. S1 is the smallest circle, and S2 is the second smallest. Vertices representing circles that actually go through the beacon have been shown in bold for clarity.

In summary, from our graph we have determined which circles go through the beacon. In addition, we also know that the vectors used to generate those circles also go through the beacon, giving us Figure 17. As a final result, we have range and bearing to the beacon from the receiver's position.

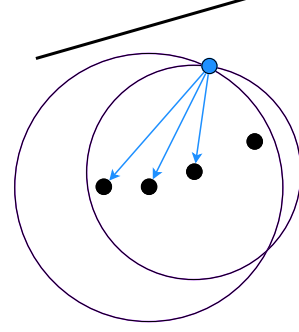


Fig. 17. The two purple circles represent our best guess at the circles that go through the beacon. Also shown are the vectors used to generate those circles.

C. Obstacle Location and Bearing

Now that we have knowledge of the beacon's actual position and bearing, we can determine the location of the reflective wall, albeit with limited resolution. With knowledge of the beacon's bearing and distance, and the bearing to the echo-generated signal, we can use the difference in time of flight between the direct beacon-to-receiver signal and the echo-generated signal to compute the complete geometry shown in Figure 18 using trigonometry. Note that this only gives one point on the reflective surface, not the entire line.

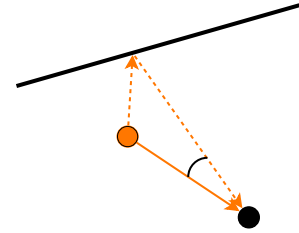


Fig. 18. The beacon (orange) location and bearing are known, as well as the location of the receiver (black). The dotted line represents the echo-generated signal vector, whose bearing is also known.

IV. PATH PLANNING

A. Properties of a Desirable Path

After building a rudimentary map of nearby beacons and reflective obstacles as shown in Figure 19, we want to be able to plan a path between two points. We want this path to have the following properties

- 1) The algorithm to compute this path must be fast enough to run continuously in real time, as our map of the world is constantly being updated.
- 2) The path should not intersect any obstacles or beacons, as this would be destructive to the vehicle.
- 3) The path should be as short as possible so as to minimize the amount of time and energy spent traveling.
- 4) The path should maximize the distance between the vehicle and nearby beacons and reflective obstacles at all times. The closer you are to a reflective obstacle or beacon, the stronger the echoes of signals reflected off that object will

be. By staying as far as possible from all reflective objects, we can mitigate signal interference.

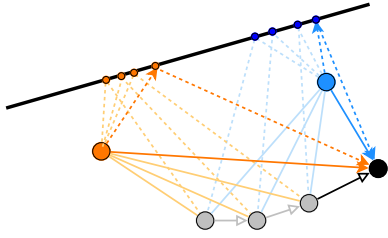


Fig. 19. Computed obstacles shown as small dots with the same color as the associated beacon.

B. Voronoi Tessellation and The Backup Path

The first step of our path finding algorithm makes use of a computational geometry construct called a Voronoi tessellation. A Voronoi tessellation is the division of a 2D space containing n sites into n convex polygons, with each polygon associated with one site. Every point inside of a polygon is guaranteed to be closer to its associated site than any other site in the tessellation, as shown in Figure 21.

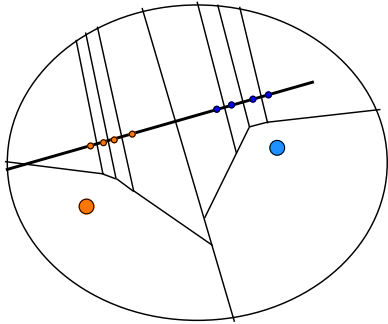


Fig. 20. Example Voronoi tessellation, showing obstacles and beacons only.

A Voronoi tessellation can be constructed in $O(n \log n)$ time using Fortune's Algorithm [3]. We now check to see if our current location and destination point are in the same Voronoi region. If so, moving from one to the other is trivial, as there is at most one object in the path (as there is only one object in each Voronoi region). If not, we now take the edges of the Voronoi tessellation to represent possible paths for our vehicle. These edges satisfy our 4th property for desirable paths in that the edges of the Voronoi tessellation are guaranteed to be a maximal distance from the two nearest sites, where sites in this case are interference-inducing objects.

However, since the edges of a Voronoi tessellation extend outwards infinitely, it is possible for our vehicle to select a path extending outwards toward infinity. This is not desirable. To mitigate this, we introduce a backup boundary path that encompasses all known beacons and reflective objects. Where this boundary path intersects the infinite edges of the Voronoi tessellation, we clip the Voronoi edge and insert a vertex at the intersection.

C. Narrow Passage Rejection

If after reducing our possible paths to a set of edges, we need to screen these edges for navigability. Though we are guaranteed that all the edges maximize the distance to the two nearest sites, two sites might be too close together for a path between them to be navigable without hitting one or both of the objects, violating property 2. Since each edge is associated with exactly two sites, determining whether or not an edge is navigable can be done simply by comparing the distance between an edge's two associated sites with a threshold value greater than the width of the vehicle.

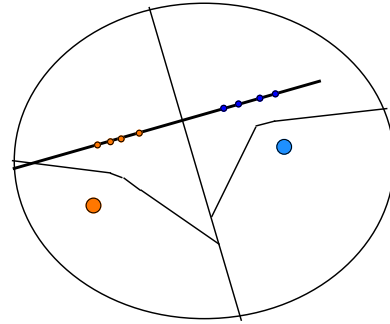


Fig. 21. Voronoi tessellation with narrow paths rejected.

D. Traversing the Tessellation

A Voronoi tessellation can be thought of as a simple weighted planar graph with the corners of the polygons as nodes and the edges of the polygons as edges weighted by length. In the previous step, we removed all unnavigable edges, meaning that every edge in the graph should now satisfy properties 2 and 4. Since every edge in the graph satisfies properties 2 and 4, every path through the graph now satisfies properties 2 and 4. To satisfy property 3, we now simply need to find the shortest path through the graph starting at the point nearest our vehicle and ending at the edges nearest the target, shown as the star in Figure 23. The problem of finding the shortest path through a positively weighted graph is well known and can be solved using Dijkstra's Algorithm in $O(n^2)$ time[4].

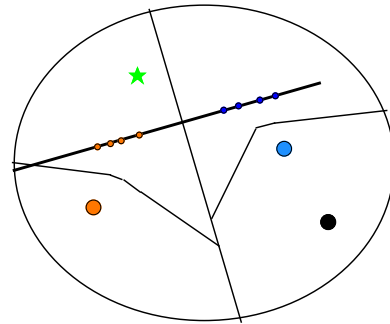


Fig. 22. Voronoi tessellation with narrow paths rejected. The green star is the goal of the vehicle, represented as the receiver (black).

E. Edge Consolidation and Path Smoothing

We now have a path that satisfies all the properties the algorithm set out to satisfy. However, since the path consists of linear sections connected by sharp corners, it is not ideal for an actual vehicle to traverse, as vehicles cannot change velocity instantaneously. As an optional final step, the path can be smoothed out to make it easier for a real vehicle to follow. One way this could be done is by fitting circular arcs with radii greater than the minimum turning radius of the vehicle to all corners in the graph.

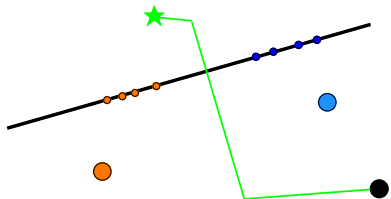


Fig. 23. Example path from vehicle (black) to goal (green star).

Notice that in the provided example, the planned path runs straight through a wall. This is clearly not ideal. When the route was planned, the vehicle knew nothing about the portion of the wall it planned through. However, as the vehicle travels the planned path, it will acquire more and more reflections from the center portion of the wall. As it is constantly rerunning the path planning algorithm, paths running through the center of the wall will be pruned out in the Narrow Passage Rejection phase and the selected path will circumvent the wall.

V. CONCLUSION

In summary, our algorithm starts by using a specialized form of spread spectrum codes in a bearing finding algorithm that can calculate the both the angles to the transmitter and its reflection, and the difference in the time of flight for these two paths. The second step uses spatial disparity over time with the calculated bearings to find the position of the beacons and obstacles. Lastly, a Voronoi tessellation-based path planning algorithm interprets these obstacle coordinates to generate safe paths through the environment.

By combining the three steps of our algorithm, we have established a method for SLAM using only a phased antenna array and a set of active beacons. Although computationally expensive, especially in the bearing finding step, our algorithm puts little strain on the physical layer of the system and requires very little in terms of sensors.

Future work is required in optimizing the modulation codes, as they possibly consume more air time than is necessary. However, they are sufficient to guarantee object detection with an arbitrarily small minimum radius.

VI. ACKNOWLEDGMENTS

We'd like to thank the following professors for their help in research and brainstorming for this paper:

- Sarah Spence Adams
- Raymond Yim
- Brian Bingham
- Gill Pratt

We'd also like to thank Alex Davis and Rob Nix for proofreading this paper.

REFERENCES

- [1] Olson, E.; Leonard, J.; Teller, S., "Robust range-only beacon localization," *Autonomous Underwater Vehicles, 2004 IEEE/OES*, vol., no.pp. 66- 75, 17-18 June 2004
- [2] Middleditch, A. E., Stacey, T. W., and Tor, S. B. 1988. Intersection algorithms for lines and circles. *ACM Trans. Graph.* 8, 1 (Nov. 1988), 25-40.
- [3] Steven Fortune, *A Sweepline Algorithm for Voronoi Diagrams*, *Algorithmica* 2(2):153-174, 1987.
- [4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 24.3: Dijkstra's algorithm, pp.595-601.