Time Difference of Arrival (TDOA) Algorithms

UBC Subbots

July 19, 2020

Revision 1.0

Contents

1 Introduction

Assuming a constant speed of sound in water, the 3D coordinates of the pinger can be found using the following equations:

$$\left|\vec{r} - \vec{R}_i\right| = ct_i \tag{1}$$

Where $\vec{r}$ is the position of the pinger, $\vec{R}_i$ is the position of the ith hydrophone, c is the wave speed, and t_i is the time of arrival (TOA) of the signal at the ith hydrophone.

Since the pressure sensor can be used to give an accurate reading of the height of the hydrophones relative to the pinger (assuming the height of the pinger is known), solving for the position of the pinger in 2D is sufficient (with the added known z term). Thus, equation 1 can be reduced and simplified using polar coordinates r and ϕ to be:

$$r^2 + r_i^2 - 2rr_i\cos(\phi - \phi_i) + (z - z_i)^2 = c^2t_i^2 \tag{2}$$

Since the hydrophones are not synchronized with the pinger, it is impossible to get an absolute measurement of the time of arrival of the signal. However, through various techniques, the time difference of arrival (TDOA) between any pair of pingers can be determined. Thus, equation 2 can be modified to account for the TDOA as follows:

$$\sqrt{r^2 + r_i^2 - 2rr_i\cos(\phi - \phi_i) + (z - z_i)^2} - \sqrt{r^2 + r_j^2 - 2rr_j\cos(\phi - \phi_j) + (z - z_j)^2} = c(t_i - t_j) = \delta_{ij} \tag{3}$$

This produces a set of n-1 independent, non-linear equations for a set of n hydrophones. This document sets to explore and summarize different techniques and algorithms to solve the TDOA system of equations.

This set of equations can be simplified further by placing the origin of the coordinate system at hydrophone 1, and forcing all equations to be in comparison to it. Thus, we get a final form for the 2D TDOA equations:

$$\sqrt{r^2 + z^2} - \sqrt{r^2 + r_j^2 - 2rr_j\cos(\phi - \phi_j) + (z - z_i)^2} = c(t_1 - t_j) = \delta_j, \quad j\epsilon[2, n] \tag{4}$$

where $\delta_j = \delta_{1j} = c(t_1 - t_j)$.

2 2D TDOA Analytical Solution

The TDOA equations in a 2D space can in fact be solved analytically using algebraic manipulations.

To begin, we will assume a minimum of 3 hydrophones are needed to produce a unique solution (this is due to the fact that there are two unknown variables, r and ϕ, and a set of 3 hydrophones will reduce equation 4 to a set of 2 equations. However, due to the non-linearity of the equations this may end up

being false). Thus, equation 4 a set of two equations relating the angle of the pinger's position and it's distance relative to hydrophone 1:

EQUATIONS NEED UPDATING SINCE THEY NEGLECT THE Z TERM

$$\cos(\phi - \phi_2) = \frac{2r\delta_2 + r_2^2 - \delta_2^2}{2rr_2} \tag{5}$$

$$\cos(\phi - \phi_3) = \frac{2r\delta_3 + r_3^2 - \delta_3^2}{2rr_3} \tag{6}$$

Furthermore, the following trigonometric identity holds true for any value of ϕ, ϕ_2, and ϕ_3:

$$[\sin\phi_3 \cos(\phi - \phi_2) - \sin\phi_2 \cos(\phi - \phi_3)]^2 + [\cos\phi_3 \cos(\phi - \phi_2) - \cos\phi_2 \cos(\phi - \phi_3)]^2 = \sin^2(\phi_3 - \phi_2) \tag{7}$$

Plugging in equations 5 and 6 into the identity above and rearranging for r and ϕ gives the following results:

$$A_{123}r^2 + B_{123}r + C_{123} = 0 \tag{8}$$

$$D_{123}\tan^2\phi + E_{123}\tan\phi + F_{123} = 0 \tag{9}$$

where

$$A_{1ij} = 4[\delta_i r_j^2 + \delta_j r_i^2 - 2r_i r_j \delta_i \delta_j \cos(\phi_i - \phi_j) - r_i^2 r_j^2 \sin^2(\phi_i - \phi_j)] \tag{10}$$

$$B_{1ij} = 4[r_j^2\delta_i(r_i^2 - \delta_i^2) + r_i^2\delta_j(r_j^2 - \delta_j^2) - r_i r_j \cos(\phi_i - \phi_j)(\delta_j(r_i^2 - \delta_i^2) + \delta_i(r_j^2 - \delta_j^2))] \tag{11}$$

$$C_{1ij} = r_j^2(r_i^2 - \delta_i^2)^2 + r_i^2(r_j^2 - \delta_j^2)^2 - 2r_i r_j(r_i^2 - \delta_i^2)(r_j^2 - \delta_j^2)\cos(\phi_i - \phi_j) \tag{12}$$

$$D_{1ij} = [r_i \sin\phi_i(\delta_j^2 - r_j^2) - r_j \sin\phi_j(\delta_i^2 - r_i^2)]^2 - [\delta i(\delta j^2 - r_j^2) - \delta_j(\delta i^2 - r_i^2)]^2 \tag{13}$$

$$E_{1ij} = 2[r_i \sin\phi_i(\delta_j^2 - r_j^2) - r_j \sin\phi_j(\delta_i^2 - r_i^2)][r_i \cos\phi_i(\delta_j^2 - r_j^2) r_j \cos\phi_j(\delta_i^2 - r_i^2)] \tag{14}$$

$$F_{1ij} = [r_i \cos\phi_i(\delta_j^2 - r_j^2) r_j \cos\phi_j(\delta_i^2 - r_i^2)]^2 - [\delta_i(\delta_j^2 - r_j^2) - \delta_j(\delta_i^2 - r_i^2)]^2 \tag{15}$$

Note that this method leads to a quadratic equation for both r and ϕ, meaning that 3 hydrophones are not sufficient to produce a unique solution to the 2D TDOA equations.

Let Q_{1ij} to be the pair of quadratic equations for r and ϕ resulting from the 2D TDOA equations of hydrophones 1, i, and j. Then a fourth hydrophone would allow to obtain both Q_{123} and Q_{124}. This would provide a system of quadratic equations for r and ϕ that would allow to solve for a unique solution.

This leads to the following solution for r and ϕ:

$$r = \frac{A_{123}C_{134} - A_{134}C_{123}}{A_{134}B_{123} - A_{123}B_{134}} \tag{16}$$

$$\tan\phi = \frac{D_{123}F_{134} - D_{134}F_{123}}{D_{134}E_{123} - D_{123}E_{134}} \tag{17}$$

Note that this assumes an error free environment and perfect compliance with the model.

3 Iterative Numerical Solutions

While a unique analytical solution to the 2D TDOA problem can be found, multiple sources report that it is quite sensitive to error, and may be less accurate in a real system than numerical methods. It is also important to explore what a solution might look like if the system became over constrained with additional hydrophones to reduce error. For that reason, various numerical methods to solving the 2D TDOA problem should be explored in detail.

3.1 Method of Nonlinear Least Squares

With iterative numerical solutions, one of the main goals of a given method is to minimize (or maximize) some quantity that gives a measure of the quality of the result. One such optimization is the minimization of the sum of square residuals of the outputs from the given model equations (aka least squares). To provide a better idea of the quantity we are trying to minimize, the 2D TDOA equations are written in the following form:

$$t_{1j} = f(r_j, \phi_j, z_j, z, r, \phi) = f_j(r, \phi) \quad j\epsilon[2, n] \tag{18}$$

For a given value of r and ϕ, r^* and ϕ^*, we can find the value of the sum of the difference of squares between the predicted time difference and the measured time difference:

$$s = \sum_{j=2}^{n} e_j^2 = \sum_{j=2}^{n} [t_{1j} - f_j(r^*, \phi^*)]^2 = \tag{19}$$

where e_j is the error for the 2D TDOA equation of hydrophones 1 and j. The method of least squares uses the value of s as a measure of the quality of the solution and seeks to minimize it.

As with every optimization problem, the solution involves the gradient of the function (since the problem is optimized when the gradient is 0). We can then find the components of the gradient as:

$$\frac{\partial s}{\partial r} = \frac{\partial}{\partial r} \sum_{j=2}^{n} [t_{1j} - f_j(r, \phi)]^2 = -2 \sum_{j=2}^{n} [t_{1j} - f_j(r, \phi)] \frac{\partial f_j}{\partial r} \tag{20}$$

$$\frac{\partial s}{\partial \phi} = \frac{\partial}{\partial \phi} \sum_{j=2}^{n} [t_{1j} - f_j(r, \phi)]^2 = -2 \sum_{j=2}^{n} [t_{1j} - f_j(r, \phi)] \frac{\partial f_j}{\partial \phi} \tag{21}$$

By defining the error vector $\vec{e}$ and Jacobian Matrix $\mathbf{J}$:

$$\vec{e} = [e_2, \ldots, e_n]^T = [t_{12}, \ldots, t_{1n}]^T - [f_2(r^*, \phi^*), \ldots, f_n(r^*, \phi^*)]^T \tag{22}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_2}{\partial r} & \frac{\partial f_2}{\partial \phi} \\ \vdots & \vdots \\ \frac{\partial f_n}{\partial r} & \frac{\partial f_n}{\partial \phi} \end{bmatrix} \tag{23}$$

We get a closed form equation for the gradient of the sum of squares of the 2D TDOA problem:

$$\vec{\nabla} s = -2 \mathbf{J}^T \vec{e} \tag{24}$$

For a linear least squares problem, setting the gradient 0 and finding the minimizing solution can be done quite easily analytically. However, since these equations are non-linear, we must find an alternative way to optimize the problem.

In essence, the optimizing solution must be found iteratively, by defining a starting point and stepping in a determined direction until satisfying a minimization condition. The starting point can simply be the analytic solution to the TDOA equations. However, there are multiple ways to determine the step size and direction.

NOTE: This same analysis can be done in terms of x,y coordinates, and might actually lead to better results. Worth exploring.

3.1.1 Gradient Descent Optimization

The main principle of gradient descent is to utilize the fact that the gradient of a function points in the direction of maximum rate of change (and conversely, in the opposite direction of maximum decrease). Thus by stepping in the direction of the negative gradient, we can approach the minimum of the function.

The method has the following limitations:

1. The method could be quite sensitive to the starting point of the function since it follows its gradient. This can be seen quite clearly in Figure 1, where a slight shift in the starting point could cause it to fall down in different directions.
2. The method requires some careful consideration and experimentation with the step size. Too large of a step size, and the method will become unstable (always overshoot in its stepping direction, zigzag in its path like in Figure 1, etc.). Too small of a step, and the algorithm will take too long to converge. Thus, a parameter between 0 and 1 defining the step size (proportion of the magnitude of the gradient) should be tuned.

3. The termination condition also plays a crucial role similar to that of the step size. Since a value of exactly 0 cannot be reached numerically, we must define what might be considered "good enough". The termination condition also presents a trade off between convergence time and accuracy.

4. If the function is highly unstable and changes in multiple directions rapidly, this method performs quite poorly.

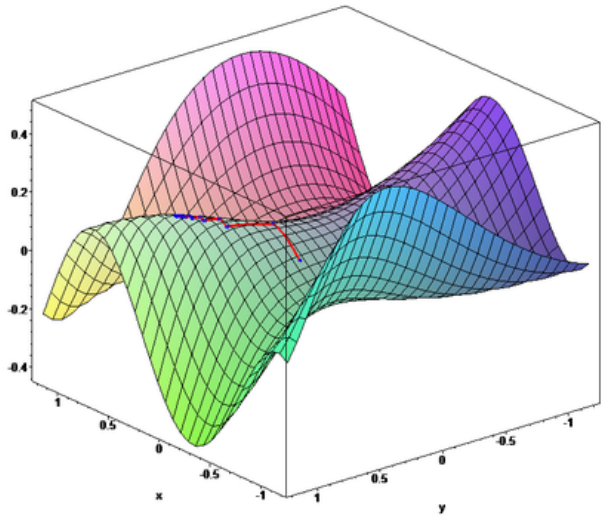

Figure 1: A visualization of gradient descent in a 2D function. The algorithm can be seen to always take paths "downhill".

This leads to the following step formula:

$$\Delta\vec{r} = -\gamma\vec{\nabla}s = \gamma(2\mathbf{J}^T\vec{e}) \tag{25}$$

Where γ is the adjustable step size parameter.

3.1.2 Nelder Mead Optimization

The Nelder Mead optimizer works by drawing a simplex (n+1 vertex shape in n dimensions resembling a triangle: triangle in 2D, tetrahedron in 3D). Since we work in a 2D space, this section will focus on the 2D, triangular Nelder Mead implementation. The algorithm works as follows:

1. Starting at a center point, draw a triangle in the parameter space by choosing 3 test points

2. Find the value of the least squares function, s, at each test point

3. Rank each test point by the quality of s. Label the test point that minimizes s (best result) as $\vec{r_1}$ and the point that maximizes s (worst result) as $\vec{r_3}$

4. The next triangle can be constructed by replacing $\vec{r_3}$ with a new test point, keeping $\vec{r_1}$ and $\vec{r_2}$ as is. The new replacement test point can be found as follows:

   (a) Find the centroid of $\vec{r_1}$ and $\vec{r_2}$ (the two unchanging test points), $\vec{r_0}$

   (b) We compute a reflected point, $\vec{r_r}$, by reflecting the discarded point $\vec{r_3}$ across the centroid r_0, scaled by a parameter $\alpha > 0$:

$$\vec{r_r} = \vec{r_0} + \alpha(\vec{r_0} - \vec{r_3}) \tag{26}$$

(c) If $\vec{r_r}$ produces a smaller value of s than $\vec{r_2}$ but a larger value than $\vec{r_1}$ (i.e. it is right in the middle in terms of quality), then $\vec{r_r}$ becomes the last test point and the next triangle is complete (at this point break from these sub-bullets since an answer is found)

(d) If $\vec{r_r}$ is the best point, we would like to move further in that direction to speed up termination. In that case, we re-compute the reflected point but replace α with another parameter $\gamma > \alpha$. We name this expanded point $\vec{r_e}$:

$$\vec{r_e} = \vec{r_0} + \gamma(\vec{r_0} - \vec{r_3}) \tag{27}$$

(e) if the expanded point $\vec{r_e}$ performs better than the reflected point $\vec{r_r}$, then $\vec{r_e}$ becomes the new test point. Otherwise ($\vec{r_r}$ still performs better than $\vec{r_e}$), then we take $\vec{r_r}$ as the new test point. In both cases, the next triangle is complete (at this point break from these sub-bullets since an answer is found)

(f) Lastly, we deal with the case that $\vec{r_r}$ is still the worst performing test point. In which case, we pull the reflected point closer to the center. We recompute the reflected point as the contracted point, $\vec{r_c}$, by replacing α with $\rho\epsilon(0, \alpha)$:

$$\vec{r_c} = \vec{r_0} + \rho(\vec{r_0} - \vec{r_3}) \tag{28}$$

(g) If the contracted point, $\vec{r_c}$ performs better than the initial worst point $\vec{r_3}$, then the last test point becomes r_c and the next triangle is complete (at this point break from these sub-bullets since an answer is found)

(h) In the very rare and RIP case that the contracted point performs even worse than the originally worst performing point $\vec{r_3}$, the space must be quite complex. In this case, we shrink all points towards the best performing point $\vec{r_1}$ to form the next triangle in the hope that the new region's space will be simpler. Here, σ is the shrink constant:

$$\vec{r_i} = \vec{r_1} + \sigma(\vec{r_1} - \vec{r_i}), \quad i = 1, 2 \tag{29}$$

5. continue iterating through triangles until the standard deviation in the value of s across different test points it sufficiently small. At that point, take the best performing test point.

Note that part of the challenge with this method in addition to the algorithm parameters is the choice of the starting test points and the termination condition. Too small of a triangle can lead to too local of a search, while too large of a triangle can lead to the algorithm taking a long time to terminate by covering too many changes in the solution space. The termination condition decision is similar to gradient descent. Note that for relatively "flat" functions, the resulting values can be highly sensitive to the termination condition. An additional condition for the area of the triangle can be added to determine if termination happened in a satisfactory fashion.

3.1.3 Gauss-Newton Algorithm

Equation 24 can be re-written as the following set of equations:

$$\frac{\partial s}{\partial a_j} = -2\sum_{i=2}^{n} e_i J_{ij} = 0, \quad j = 1, 2 \tag{30}$$

where a_j is the jth component of the predicted position vector, $\vec{r}$. This could be either [x,y] or [r,ϕ].

The minimization of least squares requires the gradient to be 0, and thus, we wish to solve when equation 30. However, this cannot be done analytically, so we define a step size in the value of each a_j:

$$\vec{r_{k+1}} = \vec{r_k} + \Delta\vec{r} \tag{31}$$

and thus can write the finite difference formula for the model function:

$$\vec{f}(\vec{r_{k+1}}) \approx \vec{f}(\vec{r_k}) + \mathbf{J}\Delta\vec{r}, \quad \vec{f} = [f_2, f_3, \ldots, f_n] \tag{32}$$

This could be replaced into equation 30 to find the step size required for the next iteration:

$$\frac{\partial s}{\partial a_j} = -2\sum_{i=2}^{n}(t_{1i} - f_i(\vec{r_{k+1}}))J_{ij} \approx -2\sum_{i=2}^{n} J_{ij}\left(t_{1i} - f_i(\vec{r_k}) - \sum_{m=1}^{2}\Delta a_m J_{im})\right) = 0, \quad j = 1, 2 \tag{33}$$

This can give us an equation to solve for the step size in terms of the current value of $\vec{r}$ (the factor of (-2) was discarded since the expression is equated with 0):

$$\sum_{i=2}^{n} e_i(\vec{r_k})J_{ij} - \sum_{i=2}^{n} J_{ij}\sum_{m=1}^{2}\Delta a_m J_{im} = 0, \quad j = 1, 2 \iff \left(\mathbf{J}^T\mathbf{J}\right)\Delta\vec{r} = \mathbf{J}^T\vec{e}(\vec{r_k}) \tag{34}$$

Finally, we can find a closed form expression for the desired step:

$$\Delta\vec{r} = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\vec{e}(\vec{r_k}) \tag{35}$$

Note that $(\mathbf{J}^T\mathbf{J})$ is always invertible as long as $\mathbf{J}$ has linearly independent columns. This is because $(\mathbf{J}^T\mathbf{J})$ and $\mathbf{J}$ have the same nullspace by the following proof:

$$\mathbf{J}^T\mathbf{J}\vec{x} = 0 \rightarrow \vec{x}^T\mathbf{J}^T\mathbf{J}\vec{x} = 0 \rightarrow (\mathbf{J}\vec{x})^T(\mathbf{J}\vec{x}) = 0 \rightarrow ||\mathbf{J}\vec{x}||^2 = 0 \rightarrow \mathbf{J}\vec{x} = 0 \tag{36}$$

To aid with convergance, a parameter $\gamma\epsilon(0, 1)$ is added to the equation. Its role is similar to the step size parameters in the previous algorithms:

$$\Delta\vec{r} = \gamma(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\vec{e}(\vec{r_k}) \tag{37}$$