Documentation

# Introduction

Localization algorithms are used to locate the source of a signal, based on the characteristics of the signal when it is received at different locations. The process involves transmitting a signal from a device an unknown location (called the “target” node) and recording signal characteristics at receiver nodes at known locations, also called “anchor” nodes. Some applications include locating the center of the rotor core in hearts with Atrial Fibrillation (AFib), which allows for treatment, and also locating the source of a forest fire. Several localization techniques exist, such as Time of Arrival (TOA), Time Difference of Arrival (TDOA), Modified Time Difference of Arrival (mTDOA), and Received Signal Strength (RSS). It has been shown that mTDOA is particularly useful in the case of AFib modeled by the FitzHugh-Nagumo (FN) model since it gives the signal propagation speed as an output rather than an input. mTDOA

This project aims to improve this algorithm by considering the entire received signal rather than the initial time of reception and applying synchronization-based techniques to estimate time delays. In synchronization, the influence delay between two signals can be deduced using the mutual information (MI) between time-shifted versions of those signals. When the MI is at a maximum, it is likely that the particular time delay that yields the maximum MI is the correct estimate of the delay between the transmitted and received signal. Outputs from the FN model at the locations of the anchor nodes can be put under this MI Time Shift process to learn the time difference between two anchor nodes, which can be fed to the standard mTDOA algorithm.

[Add output/results–we don’t have output/results yet]

# The Theory

There are two major kinds of signal localization, localization involving a single transmitted “message” or signal, and that involving a continuous signal received over time. Both use many “anchor nodes”, which receive the signal and one or more target nodes which transmit the signal.

## Single message

With the single message system, it is easy to determine the time delay of the received signal between two anchor nodes; the received time at one node can be subtracted from the received time at another node. These time delays, denoted t­jk for the time difference between the jth and kth nodes, can be fed into matrices which can produce an estimate for the location of the target node. In each localization method, there are three matrices/vectors, coming together into the equation, **Hx**=b. **x** contains the location the target node and sometimes other helpful information, such as the time of transmission or the speed of propagation. **H** and *b* contain the gathered information from the time delays. **r**j is the position of the jth node (with 0 being target), tj is the absolute time of arrival of the message at the jth node, and c is the propagation speed. These are the matrices for mTDOA.

Diagram

Description automatically generated

The code supports TOA, TDOA, mTDOA, and cTDOA.

If the system includes multiple target nodes, an iterative approach may be needed to localize all of them, even transmitting from the same target node twice.

## Continuous signal

In many cases, the received signal is not one event but a continuous, time-varying signal. The time delay between two signals can be determined using the Mutual Information (MI) Time Shift method mentioned in the introduction. For each time shift, the MI between the two signals is calculated, and the time shift that yields the highest MI is said to be the time delay between the two received signals. Once the time delays between the received signals at each node and the received signal at node 1 (the “base” anchor) are calculated, the time delays can be fed into whichever algorithm is needed. Typically, mTDOA or cTDOA will be used for this because they do not require the propagation speed.

The code supports a few signal types including Gaussian Auto-Regressive (“ar”) and FitzHugh-Nagumo (“fn”).

# The Code

## Initializing a WSN

The localization algorithms are implemented in the WSN class (WSN.py). To run the algorithms, a WSN can be created and worked with directly, or an App (App.py) can be created which creates a WSN for itself. Assuming, a WSN is being used directly, it accepts the parameters: N, the number of nodes (anchors + targets); and size, the width of the location space. The location space is always square, [0, size)2. 100 is typically used for size.

## Creating the nodes

Then, the nodes must be created by calling either reset\_nodes or reset\_clusters. reset\_nodes randomizes N node positions and stores them in self.nodes. If cTDOA is being used, reset\_clusters should be called so that the nodes are initialized in clusters, and num\_clusters and cluster\_size should either be set by passing key word arguments to WSN’s initializer or setting them before calling reset\_clusters. reset\_clusters chooses num\_clusters random positions around the location space and then places cluster\_size nodes around each cluster. It also resets self.N to num\_clusters \* cluster\_size.

Alternatively, the nodes can be set manually be assigning the WSN’s variable, nodes with a numpy array of shape (N, 2).

## Selecting anchor nodes

After the nodes are created, the anchor nodes should be selected by passing a set of indices to reset\_anchors. One can select the first n nodes to be anchor nodes by calling reset\_anchors(set(range(n))). If cTDOA is being used, then all nodes should be anchor nodes. (The estimated position of the target node should be the center of the rotor core in the FN model.)

## Estimating target position

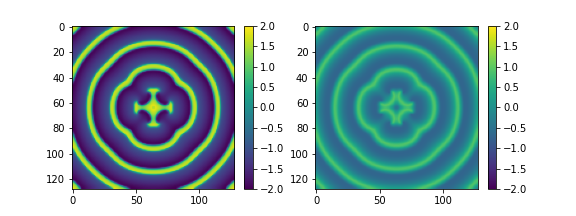
Finally, the localization can be done using localize for single-message localization or localize\_continuous for continuous-signal localization.

localize receives an argument, method, a string describing which method to use (“TOA”, “TDOA”, etc.). First, it generates the time delays using transmit, which returns a list of tuples containing the positions, received times, and other information from each node and passes those results to get\_H\_and\_b for whichever method is being used, and calculates the estimated position of the target node.

localize\_continuous receives two arguments, method and signal\_type. signal\_type is a string containing which signal should be used in the simulation (“ar” or “fn”). It uses transmit\_continuous, which returns a list of tuples containing the positions and received signals (as a numpy array) at each node. Then it uses get\_time\_delay (from mutual\_information.py) to find the time delay between the signal at each anchor and the base anchor. It reformats the results from localize\_continuous to look like the results from localize and passes them to get\_H\_and\_b like normal.

## The FitzHugh-Nagumo Model

The FitzHugh-Nagumo (FN) model is one type of signal which works well with TDOA. It is described by this system of partial differential equations and can be applied to 0+ dimensions. Here, the 2D version is used. *v* and *w* are the outputs, and *ε*, *Ip*, *D*, *β*, *γ*, *wh*, and *wl* are parameters of the model. *Ip* represents the added stimulus which creates the waves, but it only applies certain Text, letter

Description automatically generatedlocations at certain times. With this version of the model, waves only came when the start time of *I* was greater than 0 (~20 frames).

The solution to the FN model was approximated using py-pde (which uses the Finite Difference Method). The model itself is found in fn.py, and the parameters for generating the default FN solution are in mutual\_information.py under default\_fn\_equ\_params. The parameters for the shown solution are as follows: {"N": 128, "T": 1000, "dt": 0.1, "D": 1.0, "a": 0.5, "b": 0.7, "c": 0.3, "I0": 1.0, "stim": [[[25, 40], [59, 69], [59, 69]]]}. Note that I0 represents the magnitude of of Ip, and stim specifies the location and duration of the stimulus (Frames 25-40, xrange 59-69, yrange 59-69). Some of the variables used in the code go by different names than in the FN equations. a, b, and c in the code correspond to γ, β, and 1/ε, respectively.

The shown solution is the final frame of a solution to the equation with the above parameters with the horizontal axis representing x, the vertical axis representing y, the left graph for v, and the right graph for w. To find the solution with the default parameters, call solve\_default\_fn\_equ from mutual\_information.py and read from default\_fn\_sol which is a numpy array of shape (T, 2, N, N) containing the states of v and w for each of T frames. Be sure to import mutual\_information as mtin and read from mtin.default\_fn\_sol, or else default\_fn\_sol will not get written to by solve\_default\_fn\_equ.