rti_stub_new.py and rti_stub.py Documentation

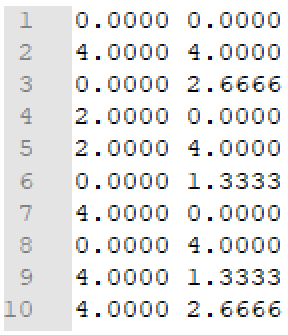
Joel Hahn, Sharath Sundaram, Nathan Mester

Program Function:

rti_stub_new.py is the primary python file for the Radio Tomographic Imaging (RTI) algorithm. The program works in conjunction with listen_new.py and rti_new.py. rti_new.py contains methods used in rti_stub_new.py while listen_new.py handles data structure conversion. See separate documentation at https://sites.wustl.edu/ese498devicefreelocalization/documentation/ for additional information on listen_new.py. The three python programs are used to record Radio Signal Strength (RSS) values for a group of radio transmitters/receivers referred to as sensors/nodes and display a video and coordinate estimate of a person standing somewhere within the group of sensors. For a space of four-by-four meters and ten properly set up sensors, the algorithm can estimate a person's coordinates to within one meter of their real location.

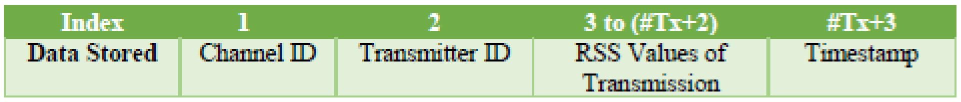
Input:

rti_stub_new.py requires rti_new.py to be present in the same folder and the numpy, matplotlib, pandas, and scipy packages to be imported to function. It requires a txt to be given under `coordFileName` that lists the x and y coordinates of all the sensors in the following format:



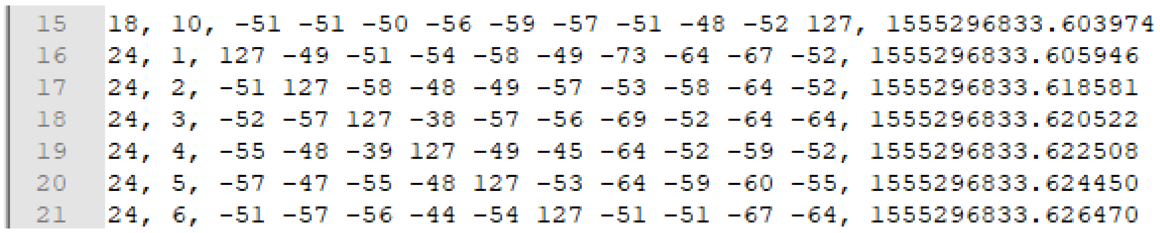
Node ID	X-Coord	Y-Coord
1	0.0000	0.0000
2	4.0000	4.0000
3	0.0000	2.6666
4	2.0000	0.0000
5	2.0000	4.0000
6	0.0000	1.3333
7	4.0000	0.0000
8	0.0000	4.0000
9	4.0000	1.3333
10	4.0000	2.6666

rti_stub_new.py requires an input file referred to as `infile` to be given through stdin if running the program on live data, or as an argument argv[1] if using existing data. The input file needs to have the format of the output file of listen_new.py, where each line contains data regarding the transmitting RSS values of one node to all other nodes, shown below:



Index	1	2	3 to (#Tx+2)	#Tx+3
Data Stored	Channel ID	Transmitter ID	RSS Values of Transmission	Timestamp

Example input or a ten-node system:



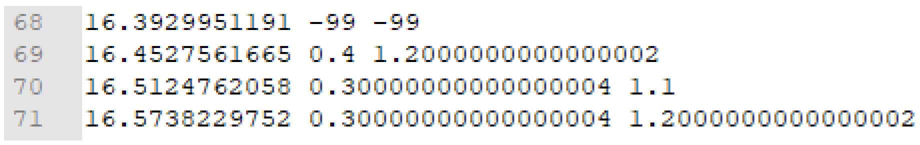
```
18, 10, -51 -51 -50 -56 -59 -57 -51 -48 -52 127, 1555296833.603974
24, 1, 127 -49 -51 -54 -58 -49 -73 -64 -67 -52, 1555296833.605946
24, 2, -51 127 -58 -48 -49 -57 -53 -58 -64 -52, 1555296833.618581
24, 3, -52 -57 127 -38 -57 -56 -69 -52 -64 -64, 1555296833.620522
24, 4, -55 -48 -39 127 -49 -45 -64 -52 -59 -52, 1555296833.622508
24, 5, -57 -47 -55 -48 127 -53 -64 -59 -60 -55, 1555296833.624450
24, 6, -51 -57 -56 -44 -54 127 -51 -51 -67 -64, 1555296833.626470
```

rti_stub_new.py requires a list of all channel ID to be given in order under `channel_list` to correctly parse `infile`.

rti_stub_new.py requires an output file name to be given under `outputFileName`. This file should be empty, as estimated coordinates of the person will be written to it with the following format:

Index	**1**	**2**	**3**
Data Stored	Timestamp	X-Coordinate	Y-Coordinate

An X and Y coordinate of -99, -99 means that no person in the area at that timestamp. An example output is shown below:



```
16.3929951191 -99 -99
16.4527561665 0.4 1.2000000000000002
16.5124762058 0.30000000000000004 1.1
16.5738229752 0.30000000000000004 1.2000000000000002
```

If set of actual coordinates with matching timestamps is available, and the Boolean `actualKnown` is set to true, rti_stub_new.py requires an .xlsx file under `excelFile` timestamps in one column, and x and y coordinates in two other columns. Set the desired columns with `usecols`.

Parameters

Rti_stub_new.py has many parameters that can be changed to affect the algorithm output. These parameters start at line 88:

- `startSkip` is the number of lines that should be skipped at the start of the file. This can be used to remove erroneous data that can often be found at the start of infile. *Default*: 80
- `plotSkip` is the number of lines that should be received between plot updates. This parameter is also used for error calculation and estimated coordinate output. This parameter effectively allows skipping video frames, so the higher the value of this parameter, the faster the plot will update, which can be useful for live data. Additionally, testing has shown that estimation error is reduced by approximately 20% if `plotSkip` is a multiple of the number of sensors, so that it is updated once per channel. *Default*: 20
- `calSets` is number of full sets of all channel's RSS data that should be used for calibration. During this time, the area that the sensors surround should be empty to obtain

background RSS values. *Default*: 50 sets, but lower values still result in similar calibration data.

- `delta_p` is the distance between pixels for display and estimation. Lower `delta_p` values will increase resolution but increase computation time and slow down plot speed. *Default*: 0.1
- `sigma_x2`, or σ_x^2 is the weight for the entire pixel covariance matrix. Higher values will result in a smoother image. *Default*: 0.5
- `delta` is the pixel covariance falloff with the exponential model. Higher values will result in greater pixel covariance, which will result in a smoother image. *Default*: 1.0
- `falloff` is the pixel weight falloff for individual links with an exponential model. A smaller value means pixels closer to the line that directly connects two nodes will be weighted more highly. *Default*: 0.01
- `excessPathLen` determines the size of the ellipse around a link between two nodes. Only pixels within this ellipse will be included in the pixel weight matrix. *Default*: 0.05
- `chLinkValue` is the weight that each included channel-link pair should be weighted by. Can be used a constant to increase the overall pixel values in the image. *Default*: 4.0
- `topChs` is the number of channels that should be included for each link. Testing has shown that keeping all channel-link pairs for calculation decreases error, but significant interference on some channels could outweigh the error reduction.
  Default: All Channels/8
- `alpha` is the value that all the previous combined images should be weighted by when adding in a new image. Higher alpha values will result in a less noisy image, but the image will have greater position estimation delay, as new data will take longer to become prevalent. The value should be between 0 and 1. *Default*: 0.985
- `personInAreaThreshold` is the pixel value at which the algorithm decides to plot and output a position estimate. This value should be low enough that an estimate is always given when someone is in the area but should be significantly above background noise. *Default*: 1.0
- `actualKnown` is a Boolean that decides whether the actual coordinates taken from an excel document should be used for error calculation.
- `plottingEnabled` is a Boolean that decides whether a plot of the images should be shown. The position estimation runs significantly faster if plotting is disabled.

Initialization

Once all required input, outputs, and parameters are given or specified, the algorithm can begin. If `actualKnown` is true and error calculation will be required, the algorithm first parses all of the data from the excel file and uses scipy's `interpolate.interp1d()`, the timestamps and x-y coordinates to create `fx`, and `fy`, two interpolated functions that give the actual coordinates of the person at any specified time.

The algorithm next loads in the sensor coordinates, the output file, the number of sensors, and the number of channels from the input files and specifications.

The algorithm then uses the rti_new.py method initRTI to create the pixel grid and pseudoinverse matrix, inverse, between the pixel values and all links, using the formula $\Pi = (\mathbf{W}^T\mathbf{W} + \mathbf{C}_x^{-1}\sigma_N^2)^{-1}\mathbf{W}^T$, where W is the link-pixel weight matrix and Cx is the pixel covariance matrix. Additional information on this pseudoinverse matrix creation can be found in the Final Report at https://sites.wustl.edu/ese498devicefreelocalization/

A single line of infile is then read to obtain the number of nodes and number of links. The number of links is numNodes*(NumNodes-1), and the links are organized into TX-RX pairs, with the RX ID incrementing first, so that the entire ordered set of links will be:

1->2,1->3,…,1->10,2->1,…,10->8,10->9

With this information, an array prevRSS can then be created, which has size numLinks by numChannels, and contains the previous value of each channel-link pair as it is received. The total number of calibration lines, calLines=calSets*numNodes*numChannels can also be created along with an incrementor counterRTI to keep track of calibration lines. Many other arrays, constants, and Booleans are created at the end of initialization for later use.

Calibration

During the calibration period, lines are read in one at a time and parsed for their timestamp, channel ID, TX ID, and RSS data. Any RSS data with a value greater than -10 dB is erroneous and is replaced with its corresponding channel-link pair RSS value. These RSS values are then recorded in prevRSS. Note that since prevRSS is initialized to an array of zeros, if a valid RSS value for that channel-link pair has not yet been received, the replaced RSS value might still be zero. To begin calibration, the program waits for the line corresponding to TX ID 1 and CH ID 1 to be received. Waiting for the first channel to be received is purely for synchronization with the old algorithm and is not strictly necessary.

Once the desired first line has been received, the program reads through its RSS values, and after replacing erroneous values with their prevRSS counterparts, adds them to a sumRSS array and increments the value corresponding location in countCalLines. If the RSS value is still above -10 dB, then no valid value has yet been received and then RSS value is skipped over. Both sumRSS and countCalLines are the same size as prevRSS and contain space for each channel-link pair. This effectively keeps track of the number and sum of all channel-link pair RSS values received. For each line received, counterRTI is incremented and this is repeated until counterRTI equals calLines.

At the end of the calibration period, a meanRSS array is created by elementwise dividing sumRSS by countCalLines. If at any index, countCalLines is less than one, the program exits, as calibration time was not long enough to obtain a valid RSS value for each link-channel pair. This meanRSS array can then be used to create a channel weight matrix, chLinkWeight, which chooses topChs number of channels that contain the highest RSS values for each link and weighs them by `chLinkValue`. The current implementation includes all channels however, so `chLinkWeight` acts only as a difference multiplier.

Estimation and Display

Once `meanRSS` array is created, which contains the average background RSS value for each channel-link pair, the algorithm can use the remaining incoming RSS data to estimate and display a person's location. The first step is to calculate `diffVector`, which is the absolute value of the difference between the RSS values just obtained, and the corresponding slice of `meanRSS` that contains the average values of those channel-link pairs. `diffVector` is then multiply by the corresponding slice of `chLinkWeight`, which currently just multiplies this difference by `chLinkValue`. This weighted difference vector is then multiplied by the corresponding slice pseudoinverse matrix, inverse, to obtain a 1D array, called `subImage`, of all the pixel values that were affected by the difference in last obtained RSS values. This image is not useful for estimation by itself, but it is multiplied by `(1-alpha)` and added to every previous image multiplied by `alpha`. The formula for this is:

$$\hat{x}[n] = \alpha\hat{x}[n-1] + (1-\alpha)\Pi y[n]$$

Where x-hat[n] is the new `totalImage`, x-hat[n-1] is the previous `totalImage`, and PI*y[n] is the new `subImage`. This formula creates a weighted average of all sub-images, where newer sub-images are weighted by a greater factor than older sub-images. This formula has the benefit of being able to update with only partial sets of data, as pseudoinverse's linear relationship between the links and the pixels means that multiplying and adding the pixel values separately obtains the same result as multiplying entire channel sets. Additionally, this method considers that an entire channel set of RSS data is not recorded simultaneously, but is instead a TX node at a time, by weighing down earlier RSS data within the same channel. Finally, this method creates smoother image changes than working with entire sets, since RSS data is never thrown out of the total image but is instead exponentially decayed away.

Once the `totalImage` for this line has been processed, if the current `counterRTI` value is a multiple of `plotSkip`, the maximum pixel value of `totalImage` is found with another rti_new method, `imageMaxCoord` and the image is plotted with `rti_new.plotImage`. The coordinate of the maximum pixel value is plotted along with the image if the value of `the` maximum pixel value is above the `personInAreaThreshold` and that coordinate is written to `outputFileName` along with the current timestamp.

Additionally, if a set of actual coordinates is also given, that coordinate is also plotted. Since the actual coordinate's timestamps are relative to the recording start time and are not absolute time, they are automatically time-synchronized by setting the start of the actual coordinates to when a person is first estimated to appear. Note that there could be additional display latency that is removed by this time-synchronization that would be present in live test. For error calculation, the number of times a person is estimated, `personInAreaCounter`, and distance between the person's actual location and their estimated location, `RTI_err` is recorded.

At the end of the runtime, the estimated total error is calculated by taking the RMSE (root mean square error) of all the individual error distances. A score value is then calculated with the formula: $\frac{personInAreaCounter*plotSkip}{RMSE*100}$. `personInAreaCounter` is multiplied by `plotSkip` to adjust for the fact the coordinates are only estimated at multiples of `plotSkip`, but a lower `plotSkip` value is not inherently better. This score will then try to maximize the number of frames with an estimated coordinate while minimizing the RMSE. Note that this score should not be relied upon for large parameter changes without observing the actual plot, as the algorithm could begin to estimate a person's location when no one is in the space if `personInAreaCounter` is brought too high.