Paper Sensor Network Intruder detection

# Introduction:

For our seminar “wireless sensor networks” that we applied at Network Embeeded Systems at the University Duisburg-Essen we were given the task to detect an intruder within a given area using the RSSI value of distributed sensors.

In this paper we want to describe a way to observe a given space by measuring the connectivity of deployed sensor nodes. We will present a set of functions to convert the given signals into a KNN cluster that can be used to differ between 3 possible states:

The Room is empty, someone is moving through the room or standing still.

For our measurements we used the RSSI values that are generated between each Sensor node (in this paper we will referee to this as link) and distributed to a base station that will then give the information to the PC where it is processed.

There are some difficulties when using the RSSI value of Sensor nodes; the result is often not stable and the behaviour is often hard to determine as the signal strength does not always (but sometimes) match expectations.

For the Data cleansing we will use an algorithm called Alpha-Trimmed Mean Filter that will eliminate some of the noise and will the cluster the values.

We use the K-nearest-neighbour algorithm to cluster the RSSI values of each link based on different training data (empty, walking, standing); This allows us to get a live view of the current state for each link and based on that results in a statement about the whole situation in the room.

# Materials and Methods

In this paper we used 9 Telos-B sensor node running Tiny-OS to gather RSSI values. 8 of the sensors do communicate between each other and distribute there RSSI values to the 9th ; this sensor node does not gather RSSI values but will provide the collected data to the pc.

The Software consists of a few types modules, that can be chained to achieve a certain goal. Each Module accepts a Data packet consisting of a source node, a destination node and link state information like signal strength. Each module will change the Packet or adds more Information to it. For example the ATMF Filter module will cache Packets till its window buffer is filled and will then emit Packets with a filtered Strength value and added information about the data in his window related to that packet like the standard deviation. Each Filter can emit this data to many following modules.

To feed data into the system we need to pre-process the Packets we receive from the Serial Forwarder into our own Data Packet format. This is done in the Data source module. This module also provides the functions to store and replay packets. We can start recording at any time and can mix live Data with data from log files.

Every Filter module can only work on one Link so we have a Splitter module that distributes these packets to all the Filter chains.

This Filter modules can be chained together to get different data, for this Paper we mainly used an ATMF filter (to steady the signal) feeding into a KNN module (to classify data) fedding into our GUI to show an overall system stat .

There is a special group of Filter modules called Tools. These are used for Debugging or Visualization. We can plug a data logger or visualization Module anywhere in the chain to see how the Data packets at these point in the Process look like. Like a GNUPlot interface to Plot a timeline of the Signal strength or a 2D Diagram of the KNN cluster Points.

The whole process contains the following steps:

I. Collection data on the sensor node (Node)

II. Cleaning each link data with Alpha-Trimmed-Mean-Filter (PC)

III. Clustering each link data with KNN (PC)

IV. Combining each link data to one resulting state (PC)

## I.Collection Data

The TelosB firmware simply broadcasts its own ID and the current signal level of all the motes in the System. Every node listens to these broadcasts and updates the Signal strength of the sender for its own next Broadcast. With this approach we can see every possible link in the System by simply recording all the Broadcasts with one Base station.

Integer \_RSSIValues[Amount of Nodes];

…

receivedMessage(message\* m)

{

\_RSSIValues[m->nodeId] = getRSSI(m);

}

…

timeFired()

{

message\* m();

m->setPaketData(\_RSSIValues);

broadcast(m);

}

As a result we will receive an array for each sensor node containing the current RSSI value from that node to each other node at index = otherNodeID. (The RSSI value to oneself is always 0 as it is never set).

## II. Alpha-Trimmed Mean Filter

The received array will be spitted into link information containing 4 values:  
sourceNodeId, targetNodeId, RSSIValue and timestamp. Each link is the given to it’s own Alpha-Trimmed Mean Filter (where sourceNodeId->targetNodeId ⬄ targetNodeId->sourceNodeId).

The basic idea of the ATMF is to select the newest n values of the data and sort this data in a nondecending order. Using a given alpha value you can then determine to amount of points to be trimmed at each side using the formula: (alpha\*n)/2. After removing this amount of points you take the average of the remaining.

This value will then be given to the clustering algorithm.

## III. Clustering

As mentioned we uses the KNN algorithm to cluster the given values to determine the correct state.

Basically this is achieved by collecting training data for 3 possible situation pattern ( standing, walking, empty) and divide all processed data into packets sized p. For each packet we will determine the mean and variance and put the results in an 2 dimensional graph with x being the mean value and y being the variance. We do receive a graph as shown: <Bild hier>

After the system has been trained with the data we can start the observation of the room; Incoming link information use there mean and variance based on the ATMF to be temporarily placed inside the graph. We do then determine the nearest data Points and retrieves there type (walking, standing, empty). This classification is then passed to the combining link phase.

## IV. Combining link information

We have implemented a couple of different methods to determine the current state of the system. Some of them were removed, as they did not fulfil the demand for our task.

One of the measurement method that were removed was a heatmap like display of the current data, where the heat of each link was based on the current variance of this link.

Unfortunately the variance can’t be used for a proper state estimation without considering the current signal strength; that’s why we decided to use the KNN cluster algorithm to match this requirement.

The results of this link based KNN clustering is used by our system estimation methods:

1. Basic method:  
   We look at the KNN result for each link and pick the most frequent one. This is used as the current system state.
2. Advanced method:

We still look at the KNN results for each link but will use the certainty of the current assumption. The certainty is calculated by dividing the amount of the nearest cluster points of the same type though K (which is set to 10 in our basic measurements). We used this approach in order to get better detection results by less relying on bad link clusters.

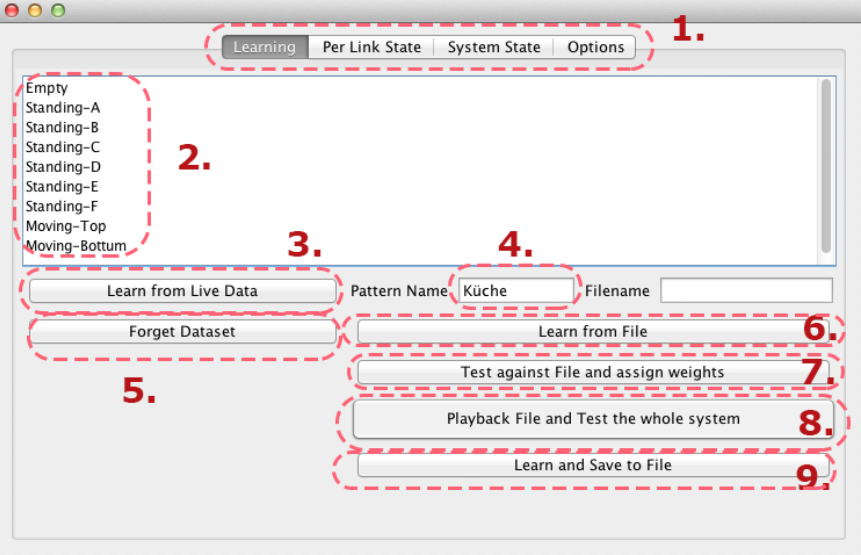
1. Trained weight method:

For the last method we have to train our System in advance (see next chapter) to distinguish good links from bad links. As some of the links may create very distinctive clustering that are easy to use for the KNN ,others might create clusters that cannot be clearly classified. Moreover some links create clusters that can be used to distinguish to system states very good, but lacks in detecting other states correctly. The trained weight method will use training data to test each link: if the link does compute the training data correctly, meaning that the cluster state result matches the training state, it will increase the trust in this link for this state; otherwise it will decrease the trust for this state.

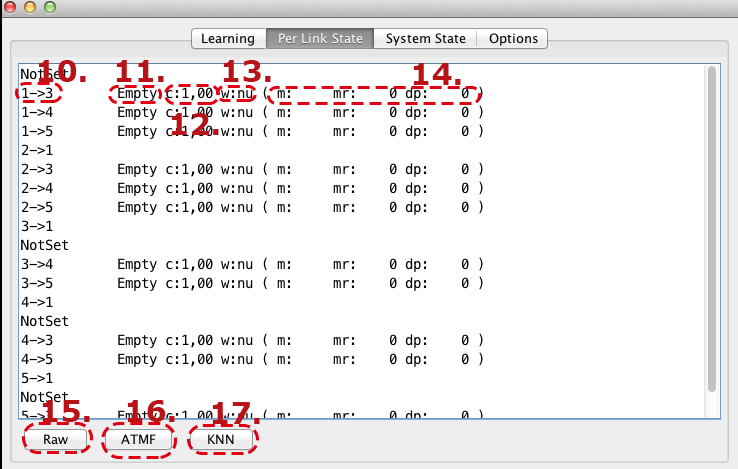
After training is completed the system can be given real data.  
The current state of the whole system is calculated by grouping each link states together and then sum up there trust level for there current detected state; the state with the highest value is displayed by the system.

# Program Usage

We created a handy java application to test our implementation. We will give an introduction to the different aspects in the following:



1. Panel to change the view
2. All states that the algorithm should be able to detect and to be trained
3. After selecting a state in (2.) you can use the current data to learn this state
4. The current Pattern name to refere to
5. Will erase the current selected learning data from (2.) in the KNN
6. Filters and clusters the data from a given Pattern (labelled <Pattern Name (4.)>-<Selected Situation (2.)>
7. Use the current selected Situation (2.) to assign weight to the KNN Results
8. Playback the current selected Pattern and Situation to test the system against this Situation. Displays the results in (1.) System State. Does also calculates the hit and miss rate.
9. Record the current messurments and store in a file labelled <Pattern Name (4.)>-<Selected Situation (2.)> . Use (3.) To stop recording.



1. Displays the current connection. E.g from Sensor id 1 to sensor id 3.
2. The current prejumted state this link has detected. (Here it detected that the room is empty right now). If no state is shown, then this link was not trained with enough data to get a meaningful result.

12.The current certainty of this Link. Is calculated by the number of K-nearest neighbours of the assumed state divided by the number of included neighbours

13. The calculated weight of this link. Will be trained with (7.)

14. <<<<>>>>>

15. Displays the future arriving raw data of the selected link

16. Displays the future arriving ATMF filtered data of the selected link

17. Displays the future arriving KNN generated data of the selected link

# Results

# Discussion

# Summary

# Acknowledgements

# References