

Reference document for SEAMCAT-3 Wiki help database
**Example of Smart Antenna Algorithm implementation using the Post-processing
plug-in functionality of SEAMCAT-3**

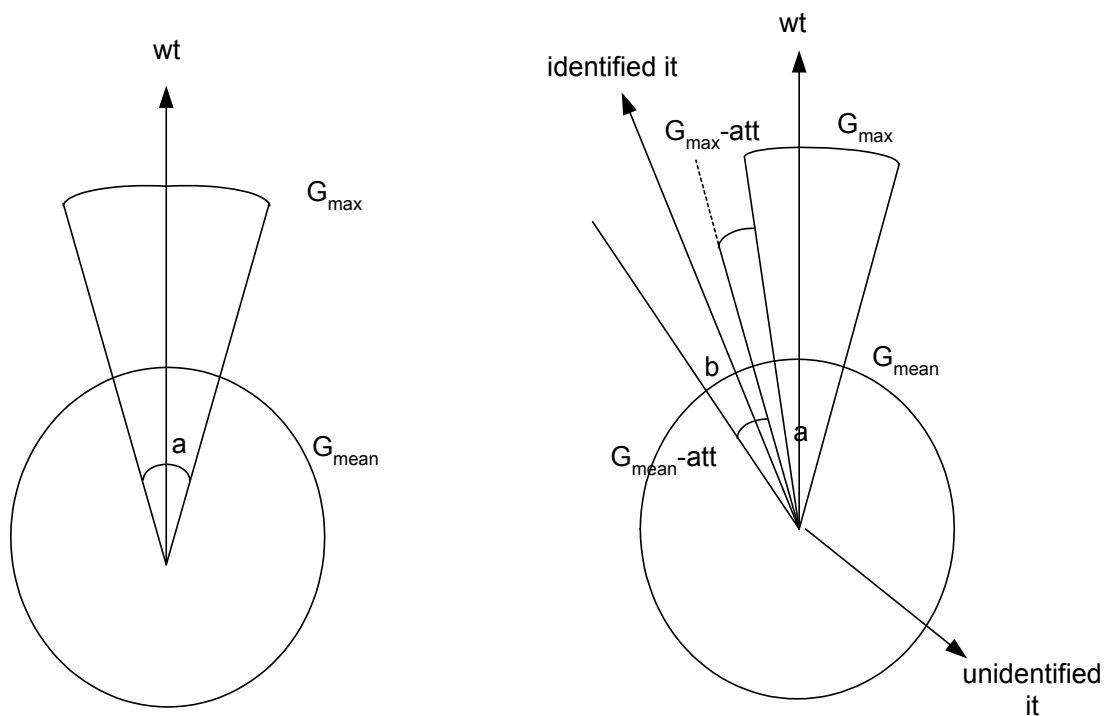
1. Introduction

This note describes a real-life example of Smart Antenna algorithm implementation as an SMC-3 post-processing plugin. As such it should be used as an example provided for educational purposes only and should not be understood to refer to any specific existing antenna type.

As the implementation of algorithm will be dependant upon which of transceivers (VR, WT, IT, WR) is equipped with such antenna, the proposed in this note algorithm assumes that Smart Antenna is used at the Victim Receiver only.

2. Concept description

This example is based on a concept of Smart Antenna as illustrated in figure below. It shows a Victim Receiver (VR) antenna radiation pattern with some mean value of G_{mean} . Then the budget of useful link is improved by VR antenna by increasing its gain towards the Wanted Transmitter (WT) by some extra gain, which added to G_{mean} results in some G_{max} .



**FIGURE: SMART ANTENNA CONCEPT USED IN THE EXAMPLE
(left hand side: no interference, right hand side: with interference)**

When interference is identified by the VR, it tries to mitigate the interference by locating and tracking one or more of the Interfering Transmitters (ITs), and applying certain attenuation to antenna gain in the direction towards the identified IT. Naturally, if the direction of IT coincides or close to the direction towards the WT, the applied attenuation might be also reducing the effect of extra gain towards WT, as shown in the right

hand side part of the above figure. Since identification and tracking of ITs requires certain resources at VR, the number of tracked ITs is finite, and any ITs in excess of this number will not be tracked and no attenuation will be applied towards them. Therefore another important part of functioning for such Smart Antenna algorithm is identification of those ITs, which create strongest interference, so that the extra attenuation is applied to the most critical ITs.

3. Description of the algorithm

Plugin configuration parameters

The following Smart Antenna parameters need to be defined by the user before running plugin:

- „Extra gain towards Wanted Transmitter, dB“ - denoted below as *ExtraWantedGain*;
- „Opening angle for applied extra gain, degrees“ – denoted as *ExtraGainAngle*;
- „Attenuation towards identified Interfering Transmitter, dB“ – denoted as *AttenuationUnwanted*;
- „Opening angle for applied attenuation, degrees“ – denoted as *AttenuationAngle*;
- „Maximum number of tracked Interfering Transmitters“ – denoted as *MaxNmbTrackedITs*

Assumed corresponding settings in general simulation scenario:

- When using this plugin, the victim receiver’s antenna should be specified in the scenario as non-directional, with antenna gain specified as to correspond to the Mean Gain of Smart Antenna;
- Antenna pointing azimuths should be specified as Constant=0 deg, to simplify the complex calculation of mutual pointing azimuths in the Smart Antenna algorithm.

The algorithm concept

During simulations in a given snapshot, the dRSS and iRSS signals are computed using the mean gain of VR antenna, as specified in the scenario. When the snapshot data is passed on to the Smart Antenna plugin, the plugin should do the following:

- Create a temporary VR antenna pattern vector, called *VRAntGainVector*, consisting of 360 elements (to represent one degree step), initially all elements made equal to mean antenna gain value, as taken from the scenario;
- Find out how many ITs are in the system (number of ILKs, each of them may contain more than one active IT), if the number is more than *MaxNmbTrackedITs* then find out the *MaxNmbTrackedITs* number of the ITs with largest impact (i.e. producing strongest original $iRSS_{unw}^i$ signal);

- Assume that VR antenna aligned directly towards WT, so add *ExtraWantedGain* value to *VRAntGainVector* positions with indexes falling within $[0 + \text{INT}(\text{ExtraGainAngle}/2); 360 - \text{INT}(\text{ExtraGainAngle}/2)]$;
- Run a cycle for each trackable IT_i : note the $\text{InterferenceLink}[j].\text{VR} \rightarrow \text{ITAntennaAzimuth}$, then subtract *AttenuationUnwanted* from all *VRAntGainVector* positions with indexes falling within $[\text{azimuthVR} \rightarrow IT_i - \text{INT}(\text{AttenuationAngle} / 2); \text{azimuthVR} \rightarrow IT_i + \text{INT}(\text{AttenuationAngle} / 2)]$. Check that attenuation had not been applied twice (e.g. if previously some other IT had been already identified with fully or partially overlapping azimuth+/- (*AttenuationAngle*/2)).
- As a result of this, the VR antenna gain vector *VRAntGainVector* will contain values that may be equal to either:
 - Mean gain, as per original scenario, or
 - Mean gain plus *ExtraWantedGain*, or
 - Mean gain minus *AttenuationUnwanted*, or
 - Mean gain plus *ExtraWantedGain* minus *AttenuationUnwanted*.
- Now take the final value of *VRAntGainVector*[0], add the difference between original mean gain and new value to the dRSS value (ideally this would be equal to extra gain, but may be less by *Attenuation* if WT is co-azimuth with one of ITs);
- Run a cycle for ALL ITs (i.e. no longer limiting to previously identified subset of tracked ITs), for each IT note the corresponding interference link VR antenna azimuth, pick up a corresponding value of VR antenna gain, add the difference between original mean gain and new value to the respective iRSS value (ideally this difference would be negative due to applied *Attenuation*, but may be 0 if this IT was not one of tracked, or even positive due to applied extra gain if e.g. IT is co-azimuth with WT);
- Return modified dRSS, iRSS values;
- End of plugin.

In addition, also the function for logging of final VR antenna gain vector per each snapshot could be added to plugin in order to help the initial checking of the algorithm. This may be e.g. complemented with having a user selectable check-box in plugin configuration interface, e.g. saying „Log final VR antenna pattern into a file“.

Consistency check function

It would be also advantageous if the following consistency checks are required by plugin:

A. Consistency check, for parameters in General scenario:

- VR antenna is specified as nondirectional, some real gain value is given, no patterns defined – if not, the simulation is stopped and message produced saying „Wrong VR antenna settings in scenario“;
- VR, WT, IT, WR antennas pointing azimuths are set to Constants=0 – if not, the simulation is stopped and message produced saying „Pointing azimuths of all antennas should be set constant=0“;

B. Consistency check, for internal plugin configuration settings:

- Check that the entered numbers for *ExtraWantedGain*, *ExtraGainAngle*, *AttenuationUnwanted*, *AttenuationAngle* are more than 0;
- Check that the value of *MaxNmbTrackedITs* is more or equal 1 – if any of these conditions is not satisfied, discontinue simulations, produce message „Wrong settings of plugin configuration parameters“.

3. Programming of the algorithm

The algorithm described above could be programmed using Java programming language as shown in the attached annex.

The process of programming, compiling and connecting the Postprocessing plugin is described in detail in the Help file of the SEAMCAT-3.

Annex

Example of programming Smart Antenna algorithm as SEAMCAT-3 Post-processing plugin

```
import java.io.*;
import java.util.Arrays;
import java.util.List;

import org.seamcat.model.plugin.*;

/**
 * @author <a href="mailto:christian.petersen@ciber.dk">Christian Petersen, CIBER Denmark A/S</a>
 *
 */
public class EroSmartAntenna implements PostProcessingPlugin, ConsistencyCheck {

    private double extraWantedGain = 10;
    private double extraGainAngle = 10;
    private double attenuationUnwanted = 20;
    private double attenuationAngle = 5;
    private int maxNmbTrackedITs = 3;
    private boolean logVrAntennaPattern = false;
    private String filename = "C:\\\\vr_antenna_pattern.log";

    private PrintStream output;

    public void init(ParameterFactory arg0) {

    }

    public void cleanUp() {
        try {
            output.close();
        } catch (Exception e) {
            //Do nothing
        } finally {
            output = null;
        }
    }
}
```

```

public String getDescription() {
    return "Victim Receiver Smart Antenna example by, developed to demonstrate SEAMCAT 3 post processing
feature";
}

public void process(ScenarioInfo info) throws Exception {
    //Create temporary VR antenna pattern
    double[] vrAntennaPattern = new double[360];
    //Initialize temp pattern with mean antenna gain from scenario
    Arrays.fill(vrAntennaPattern, info.getVictimLink().getReceiver().getAntennaGain());

    //Initialize list of tracked ITs
    List<InterferingLink> ilks = info.getInterferingLinks();
    double irssThreshold = 0;
    boolean useIRSSThreshold = false;
    if (info.getNumberOfInterferingLinks() > maxNmbTrackedITs) {
        //Scenario has more ITs than we want to track -> identify iRSS low threshold
        double[] irssValues = new double[info.getNumberOfInterferingLinks()];
        int i = 0;
        for (InterferingLink link : ilks) {
            irssValues[i++] = link.getIRSSUnwanted();
        }
        Arrays.sort(irssValues);
        irssThreshold = irssValues[info.getNumberOfInterferingLinks() - maxNmbTrackedITs - 1];
        useIRSSThreshold = true;
    }
    //Add extra wanted gain
    int angleOffset = (int) extraGainAngle / 2;
    for (int i = 0; i < angleOffset; i++) {
        vrAntennaPattern[i] += extraWantedGain;
    }
    for (int i = 360-angleOffset; i < 360; i++) {
        vrAntennaPattern[i] += extraWantedGain;
    }
    //
    boolean[] modified = new boolean[360];
    Arrays.fill(modified, false);
    for (InterferingLink link : ilks) {
        if (!useIRSSThreshold || (useIRSSThreshold && (irssThreshold < link.getIRSSUnwanted())))) {
            double aziIndex = link.getTransmitter().getVrItAzimuth();

```

```

int start = (int) aziIndex - (int) (attenuationAngle / 2);
int stop = (int) aziIndex + (int) (attenuationAngle / 2);

if (start < 0) {
    start += 360;
}
if (stop >= 360) {
    stop -= 360;
}
boolean startOver = false;
if (start > stop) {
    startOver = true;
}

for (int i = start; ((i < stop) || startOver); i++) {
    if (!modified[i]) {
        vrAntennaPattern[i] -= attenuationUnwanted;
        modified[i] = true;
    }
    if (startOver && (i == 359)) {
        i = -1;
        startOver = false;
    }
}

}

//Calculate and store new dRSS
double drss = info.getVictimLink().getDRSSValue();
drss += vrAntennaPattern[0] - info.getVictimLink().getReceiver().getAntennaGain();
info.getProcessingResults().setDRSS(drss);

//Calculate and store new iRSS values
for (int i = 1, stop = ilks.size(); i <= stop; i++) {
    double irss = ilks.get(i-1).getIRSSUnwanted();
    InterferingTransmitter it = ilks.get(i-1).getTransmitter();
    irss += vrAntennaPattern[(int) it.getVrItAzimuth()] - it.getAntennaGain();
    info.getProcessingResults().setIRSSUnwanted(i, irss);
}

//Logging
if (logVrAntennaPattern) {
    if (output == null) {

```

```

        output = new PrintStream(new BufferedOutputStream(new FileOutputStream(filename)));
    }
    output.print(info.getCurrentEventNumber() + ":\t {");
    for (int i = 0;i < vrAntennaPattern.length;i++) {
        output.print(vrAntennaPattern[i] + (((i+1) < vrAntennaPattern.length) ? ", " : ""));
    }
    output.println("");
}

}

public int getNumberOfParameters() {
    return 7;
}

public ParameterType getParameterType(int index) {
    switch (index) {
        case 5: {
            return ParameterType.Integer;
        }
        case 6: {
            return ParameterType.Boolean;
        }
        case 7: {
            return ParameterType.String;
        }
        default : {return ParameterType.Double;}
    }
}

public void setParameterValue(int index, Object value) {
    switch (index) {
        case 1: {
            extraWantedGain = (Double) value;
            break;
        }
        case 2: {
            extraGainAngle = (Double) value;
            break;
        }
        case 3: {
            attenuationUnwanted = (Double) value;

```

```

        break;
    }
    case 4: {
        attenuationAngle = (Double) value;
        break;
    }
    case 5: {
        maxNmbTrackedITs = (Integer) value;
        break;
    }
    case 6: {
        logVrAntennaPattern = (Boolean) value;
        break;
    }
    case 7: {
        filename = (String) value;
        break;
    }
}
}

```

```

public Object getParameterValue(int index) {
    switch (index) {
        case 1: {
            return extraWantedGain;
        }
        case 2: {
            return extraGainAngle;
        }
        case 3: {
            return attenuationUnwanted;
        }
        case 4: {
            return attenuationAngle;
        }
        case 5: {
            return maxNmbTrackedITs;
        }
        case 6: {
            return logVrAntennaPattern;
        }
        case 7: {

```

```

        return filename;
    }
    default: {
        throw new IndexOutOfBoundsException("Unknown Parameter Index");
    }
}

public String getParameterName(int index) {
    switch (index) {
        case 1: {
            return "Extra gain towards Wanted Transmitter, dB";
        }
        case 2: {
            return "Opening angle for applied extra gain, degrees";
        }
        case 3: {
            return "Attenuation toward identified Interfering Transmitter, dB";
        }
        case 4: {
            return "Opening angle for applied attenuation, degrees";
        }
        case 5: {
            return "Maximum number of tracked Interfering Transmitters";
        }
        case 6: {
            return "Log final VR Antenna pattern to file";
        }
        case 7: {
            return "Filename";
        }
        default: {
            throw new IndexOutOfBoundsException("Unknown Parameter Index");
        }
    }
}

public boolean check(ScenarioInfo info) {
    boolean result = true;
    if (!(extraWantedGain > 0)) {
        info.addConsistencyWarning("Extra Wanted Gain should be positive");
        result = false;
    }
}

```

```

    }
    if (!(extraGainAngle > 0)) {
        info.addConsistencyWarning("Extra Gain Angle should be positive");
        result = false;
    }
    if (!(attenuationAngle > 0)) {
        info.addConsistencyWarning("Attenuation Angle should be positive");
        result = false;
    }
    if (!(attenuationUnwanted > 0)) {
        info.addConsistencyWarning("Attenuation opening angle should be positive");
        result = false;
    }
    if (!(maxNmbTrackedITs > 0)) {
        info.addConsistencyWarning("Maximum number of tracked Interfering Transmitters should be 1 or more");
        result = false;
    }
    if (!(info.getVictimLink().getReceiver().getAntennaGain() > 0 )) {
        info.addConsistencyWarning("Victim Receiver antenna should have positive gain");
        result = false;
    }
    return result;
}
}

```