

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN



Trabajo Fin de Grado

SNOW COVER MAPPING



AUTORA: *Ester García Riege*

PROJECT: SNOW COVER MAPPING

AUTHOR: ESTER GARCÍA RIEGE
DIRECTOR: MAURO DALLA MURA
(GIPSA-LAB, INP)

GRENOBLE, 2015-2016

INDEX

LIST OF FIGURES.....	iv
1. INTRODUCTION	1
2. REMOTE SENSING: BASIC CONCEPTS.....	1
2.1. Remote sensing	1
2.2. Snow properties	7
3. PROGRAMS	8
4. IMPLEMENTATION	11
4.1. Creating the Scene	12
4.1.1. Snow simple	14
4.1.2. Importing materials	17
4.1.3. Capturing images	19
4.2. Imported Scenario.....	19
4.2.1. Importing images	22
4.3. Unmixing	24
4.3.1. Simulating ice and soil	25
4.3.2. Simulating ice and snow	28
5. CONCLUSION.....	32
6. BIBLIOGRAPHY.....	33
Annex A. Simulation procedure for simulating images.....	34
BlackBox:	34
Dart Options:.....	36
Execute Dart:	38
Sequence:	40
Land Cover Map:	41
Open all images:	44
Annex B. Simulation procedure for importing images.....	46
BlackBox:	46
Annex C. Simulation of sensors	47
Sensor Call:.....	47
Simulator Pansharpening:	48
Simulating Sensor:	52
Build Cube:	54
Annex D. Others	54
Load All Spectres:.....	54
Annex E. Unmixing at a pixel level.....	55
Unmixing	55
Range selection for sequence.....	59
Range selection for unmixing	59
Annex F. Scenarios	64
Scenario Snow Simple:.....	64
Snow Simple:	66
Scenario Unmixing:	71

LIST OF FIGURES

Figure 1. Mapping using remote sensing	1
Figure 2. Old and new technologies for obtaining images from afar	2
Figure 3. Sensoring applications.....	2
Figure 4. The three major components of radiation.....	3
Figure 5. Hyperspectral imaging	5
Figure 6. Linear surface reflectance vs lambertian surface	6
Figure 7. Atmosphere levels	7
Figure 8. Snow and ice landscapes	8
Figure 9. Reflectance of snow depending on the grain's size	8
Figure 10. Default Coeff_diff.xml	9
Figure 12. Pinhole camera	11
Figure 13. Example of the input files	12
Figure 14. A txt in one of DART's database (Lambertian.db) describing coarse snow	12
Figure 15. Reflectances of different types of material.....	13
Figure 16. Materials of the scene.....	14
Figure 17. Topography of the scene for a flat surface	14
Figure 18. Hyperspectral Image at three different wavelength values	15
Figure 19. Reflectance of an original snow scene vs the simulated scene	15
Figure 20. Different topographies.....	15
Figure 21. Topography seen from DART	16
Figure 22. Simulating topography without atmosphere: a) Hyperspectral image b) Spectrum original scene vs simulated scene	16
Figure 23. Simulating topography and atmosphere: hyperspectral image.....	17
Figure 24. Simulating topography: a) Without atmosphere b) With atmosphere	17
Figure 25. DART's Database Manager	18
Figure 26. Variable distance.....	18
Figure 27. Example of the output image.....	19
Figure 28. Urban scene in 3D and in 2D aerial view.....	19
Figure 29. Sensors: a) Ikonos sensor b) Quickbird sensor c) WV2 sensor (world view 2).....	20
Figure 30. View of urban area with three different sensors: a) Panchromatic b) Multispectral c) Global	21
Figure 31. Relative responses normalized of the sensors: a) WV2 b) Quickbird c) Ikonos....	22
Figure 32. Image seen in 3D from DART	22
Figure 33. Image seen in 2D from DART	23
Figure 34. Image seen by one channel of the sensor: a) Panchromatic b) Multispectral c) Global.....	24
Figure 35. Linear unmixing	25
Figure 36. Spectrums of the different inputs and outputs of the simulation, in μm	26
Figure 37. Spectrums of the different inputs and outputs of the simulation at a short wavelength range, in μm	27
Figure 38. Matrix of endmembers	27
Figure 39. Abundancy of each material present in both simulations.....	28

Figure 40. Pixel reflectance value from DART (S) and from MATLAB (ans)	28
Figure 41. Input and output reflectance for two types of snow, in μm	29
Figure 42. Spectra of different types of snow at a short wavelength range, in μm	30
Figure 43. Effects of scattering in snow and ice.....	30
Figure 44. Abundancy of each material present in the scene for both simulations	31

1. INTRODUCTION

This project is about remote sensing. Remote sensing consists on the acquisition of information, that is, of electromagnetic radiation in the form of hyperspectral images. This will allow one to detect the different materials that are present in a given scene. This is used in many fields such as coastal mapping and erosion prevention, to study the impact of natural disasters, or controlling the urban growth so that the damage of natural resources is minimal.

The main purpose of this paper is the implementation in MATLAB of a set of functions that will optimize the process of simulating different scenarios using the software DART. So MATLAB will become an easy tool for others to use with any kind of scene made up of any kind of material. Once this is done, it will be followed by the study of the properties of different materials in different scenarios that are found on the Earth's surface. In this particular case, the study will be focused only on environments with a high presence of snow.

After that, an unmixing of the images will be done in order to see the amount of materials present in the scene.

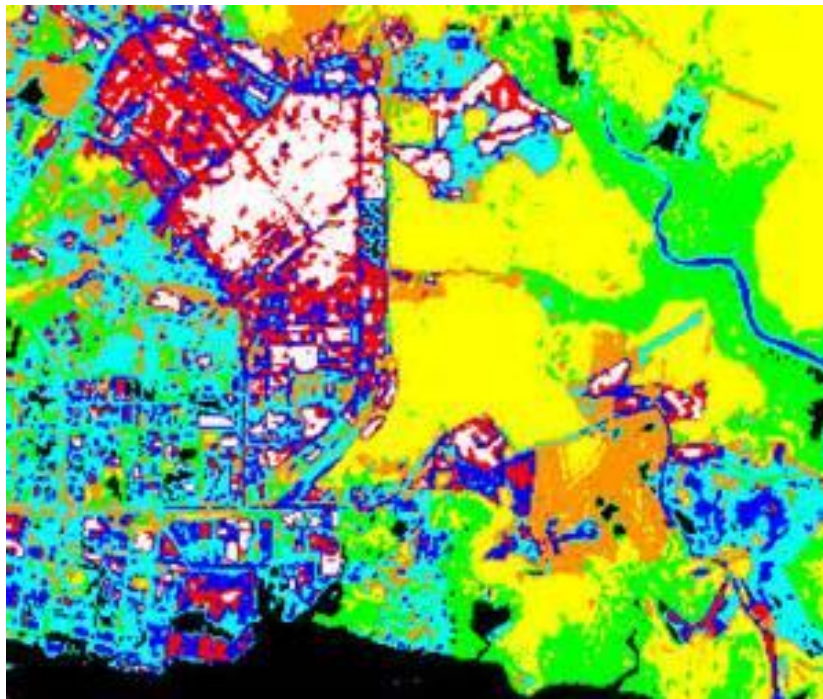


Figure 1. Mapping using remote sensing

2. REMOTE SENSING: BASIC CONCEPTS

2.1. Remote sensing

As it was said before, remote sensing is the collection of the electromagnetic radiation from a distance. There are many ways to obtain the data. It can be gathered with cameras based on the ground, on aircrafts, or on satellites. This data is to be stored in computers for

the upcoming study.

One of the first uses of remote sensing was during the US Civil War where cameras were hid on kites, balloons, and even pigeons for gathering information about enemy territory. On Figure 2, there is an example this technique (the photograph at the left), in this case, of the use of pigeons. This was called aerial photography. It wasn't until 1960, after the evolution of the methods and technologies used for obtaining the information, that the term *remote sensing* was introduced. Satellite sensors were developed during the late 20th century and now data can be obtained from a global scale and even from other planets. Also on Figure 2, (the photograph at the right) there is an example of this. The photograph shows a satellite monitoring the North American continent.



Figure 2. Old and new technologies for obtaining images from afar

The use of remote sensing has grown rapidly through the years although it is mainly for image processing and interpretation. Nowadays it is used for all kinds of fields since users can collect, interpret and manipulate different types of data over large (and often not easily accessible) areas. It can be for coastal applications such as monitoring shorelines changes and current systems. It is also used for natural resource management. It is now possible to monitor land use and chart wildlife habitats in order to try and secure natural resources. There is also the ability to study the impact that natural disasters such as earthquakes, erosion, hurricanes and floods, to name a few, have created.

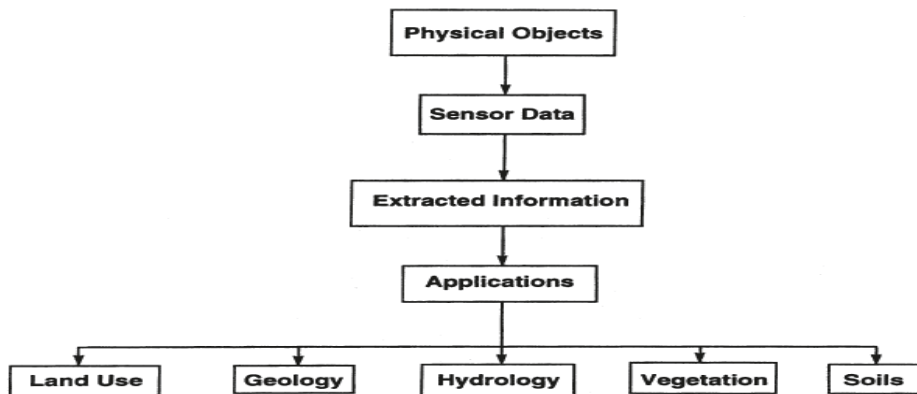


Figure 3. Sensing applications

The process of obtaining and reading data has become more sophisticated. Hyperspectral remote sensing is one of the many fields found in remote sensing. It combines two sensing modalities which are imaging and spectrometry. On one hand, imaging is the capturing of pictures from a remote distance by recollecting the reflected (or emitted) electromagnetic radiation. There are two kinds of radiation that will be considered by the sensors: the sunlight which is reflected by an object and the thermal emission from that same object and others nearby. This last source will be minimal from the visible to the shortwave infrared spectra. Multiple spectral bands can be used for the measurement so that pixels store the same scene view from different places (multispectral imaging). The use of the multiple spectral bands provides additional information that will change with the reflectance¹, or emissivity², of the pixels as a function of wavelength. Multispectral imagery supports image classification and land mapping. On the other hand, spectrometry measures the chemical composition of the materials.

The most common radiation that is used in remote sensing is solar radiation. The sun emits radiation at nearly the maximum efficiency possible.

Since one of the programs that will be used for the simulations works from the visible to thermal infrared domain, the radiation that will be studied would be just the reflectance of an object. It is important to keep in mind that the reflectance measured by the sensor won't be as pure as the radiation emitted from the sun. This modified radiation will be made up, in the most part, as three major components:

- *Unscattered, surface-reflected radiation* (generated at the visible shortwave range), this is the part of the radiation that is neither absorbed nor scattered by the atmosphere; the radiation hits directly the surface and bounces off towards the sensor.
- *Down-scattered, surface-reflected radiation* (known as the skylight component), this is the part of the radiation scattered by the atmosphere and redirected towards the Earth's surface, after it will be reflected towards the sensor.
- *Up-scattered path radiation*, this portion of radiation doesn't reach the surface of the Earth; it is scattered and redirected directly towards the sensor.

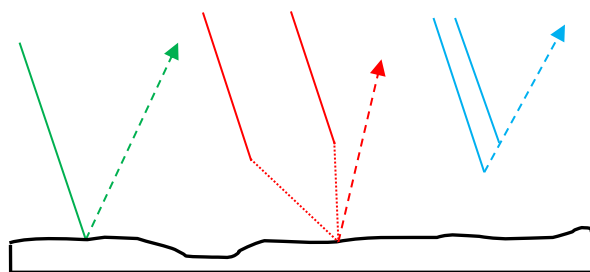


Figure 4. The three major components of radiation

¹ Reflectance of the surface of a material is the fraction of incident electromagnetic power that is reflected at an interface.

² Emissivity of the surface of a material is the ratio of the thermal radiation from the surface to the radiation from an ideal black surface at the same temperature as given by the Stefan-Boltzmann law.

Remote sensors measure the energy that is reflected from the Earth's surface. These sensors can be passive or active. Passive sensors measure natural energy, which most commonly would be sunlight. On the other hand, active sensors create their own energy source as form of measurement. Meanwhile, there are three different models for remote sensing:

- The first model records the reflection of solar radiation from the Earth's surface; the energy is mainly used in the visible and near-infrared portions of the spectrum.
- The second one records the radiation emitted from the Earth's surface; since emitted energy is strongest in the far infrared spectrum, to record these wavelengths it is required the use of special instruments; the emitted energy from the sun is mainly made of short waves which are absorbed and reemitted as longer wavelengths.
- The third one generates its own energy with the use of active sensors and then records the reflection of that energy; although this might be more complex, it is very useful since it can operate at night and in cloudy weather.

The type of remote sensing depends on what the sensors measure. Some sensors measure the vertical distributions of atmospheric parameters such as temperature, humidity, or pressure in a given area (these are called *Sounders*). Others measure the acceleration of the target or the electromagnetic energy. In this case, the sensors used will be those which detect incoming solar radiation. This type of sensor is called *Optical Remote Sensors*. They usually cover the region from visible and near-infrared to shortwave infrared.

In the Optical Remote field, there are different kinds of sensors depending on the number of spectral bands used:

- *Panchromatic*: one single channel detector; if it's configured in the visible area, the result would be that of a black and white image.
- *Multispectral*: multichannel detector with few narrow spectral bands; the result would be an image which contains both brightness and color of the target.
- *Superspectral*: same as a multispectral sensor but with many more spectral channels (more than ten); the bands are narrower too, achieving a finer spectral image.
- *Hyperspectral*: also known as imaging spectrometer, this sensor has more than a hundred continuous spectral bands which enables a finer image than the one obtained with superspectral sensors.

DART will make of if hyperspectral sensors. On the figure from below there is a better visualization of what this type of sensor generates, as it is the creation of multilayer images.

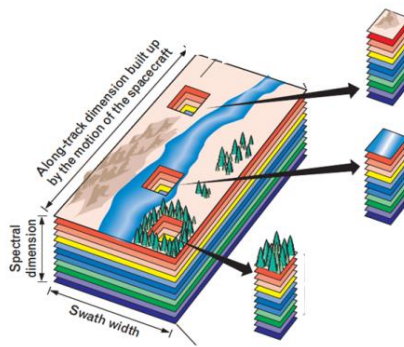


Figure 5. Hyperspectral imaging

In order to interpret the images created by the sensor spectral signatures are used. The spectral signatures are spectral measurements for identifying objects when resolution is too low for the recognition of an object. These measurements may be defined in different regions. The most common would be the solar reflective region by its reflectance as a function of wavelength. Other spectral regions would be temperature and emissivity (TIR) and surface roughness (radar).

For every type of material, there is a different signature. It is important to keep in mind that there are some factors that, regrettably, may interfere with the characterization of a material. It can be caused by the very nature of the material, its natural variability. Another factor would be the design of the remote sensing systems that quantify a given image since they are not able to retrieve an image with that high resolution.

Resolution is the sensitivity of the instrument and the existing contrast in the scene between objects and their backgrounds are always issues of significance in remote sensing investigations. These minimal areal units, known as pixels, are the smallest areal units identifiable on the image. This parameter is influenced primarily by the choice of sensor and the altitude at which it is used to record images of the Earth. As the altitude increases, the resolution of the sensor decreases.

Each sensor has its own spectral response. The spectral response is one of the parameters that characterize a sensor. It is the response of the sensor at different wavelengths. The response of an imaging system to a point source must also be measured. It comes down to the *Point Spread Function* to indicate the degradation of the imaging process. For an object to be considered a point source, it has to be smaller than the central maximum of the diffraction function, so that the diffraction may appear. As a result, the response of that point is not a point. It actually consists on a bright central disc surrounded by fainting concentric rings.

The sensors will be generated by the use of the bi-directional reflectance albedo³ values. The product function is used to study the variation of reflectance of an object for the illumination's geometry and viewing angles at a given wavelength. This function reads the directions of the light as it hits the object and as it bounces off of it. The directions will be functions of azimuth and zenith angles making, this function a four variable one. The

³ Albedo, or reflection coefficient, is the fraction of solar energy reflected from the Earth back into space. It is measured on a scale from 0 to 1, where 0 is a idealised black surface with no reflection, and 1 represents a white surface that has perfect reflection.

downside of the albedo is that it doesn't take into account the scattering properties of a surface therefore it would only be used as a first approximation.

There are many ways to obtain the bidirectional reflectance distribution function (BRDF). One can either directly measure it using calibrated cameras or using analytic models. One of this would be the Lambertian reflectance model.

A lambertian reflection defines an ideal reflecting surface, also called perfectly diffused surface. This surface exhibits equal radiance in all directions. This phenomenon depends on the wavelength and the view angle. Many natural surfaces are approximately Lambertian within a short range of view angles. It is most approximate at 20-40 degrees. As the angle passes this range and continues to increase, the materials will become less Lambertian and show different reflectance values at different directions. Nevertheless it is often assumed in remote sensing.

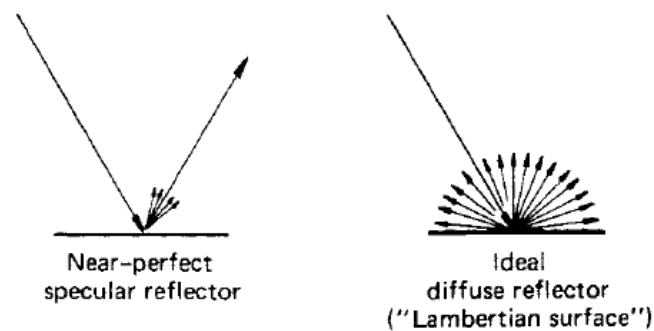


Figure 6. Linear surface reflectance vs lambertian surface

There are two types of sensors that can be used on their location. They can either be satellite sensors which are found at the top of the atmosphere (TOA) or they can be airborne sensors which are found at the bottom (BOA). The different atmospheric levels depending on the height are displayed in figure 7. For the first type of sensor, the electromagnetic energy received by the sensor is only a portion of the energy transferred by the sun (passive sensing) or by a given material (active sensing). The radiation is transmitted through space without being much altered. As it approaches the Earth's surface, some particles from the atmosphere absorb part of that energy. The most dominant components are ozone (O_3), water (H_2O), and carbon dioxide (CO_2). This absorption takes place in the bands 1.5-3 μm and 5-8 μm . There is also a minor water absorption in the microwave region near 1.36 cm. The most significant absorption, though, takes place at 0.375 cm. It is important to highlight that water vapor, unlike ozone and carbon dioxide, varies greatly with time and location. Others just divert the travelling wave (scattering). If the atmosphere wasn't taken into account both TOA and BOA sensors would generate the same images.

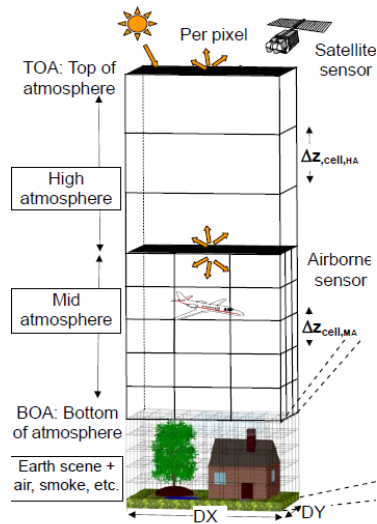


Figure 7. Atmosphere levels

2.2. Snow properties

The Earth's surface is mainly composed of water surfaces at different states. One of them is ice and snow which covers a sixth of the surface. Snow is a mixture of ice crystals, liquid water and air. It may also contain dust or dirt if the atmosphere wasn't clean during the creation of snow. Either way, the most characterizing parameter of snow is the grain size which is the radius of the ice crystals. The typical grain size is between 0.1 and 3 mm, although crystals with 0.01 mm radius can also be found. A snow pack, on the other hand, is harder to characterize. This is because of the inhomogeneities generated by the melting and refreezing that the snow can suffer.

The snows albedo depends greatly on the type of snow which is considered. When it comes to a freshly fallen snow (pure) the albedo can be as high as 0.9. On the contrary, when dirty snow is found it can be as low as 0.2. In the case of melting snow it gets to 0.4. The reflectance of the snow varies little over the range of wavelengths 0.4 to 0.65 μm , which is in the visible band. The reflectance depends on the grain size of the particular set of snow. As this value increases, the reflectance decreases. The impurity of the snow, that is, the amount of dust that can be found also decreases its reflectance. It is also important the amount of snow accumulated. If the snowpack is not thick enough, most photons will travel through the material without scattering. The scattering effect is the process in which part of the radiation is forced to change direction into different paths when they hit non uniformities. In the thermal infrared region, the reflectance decreases considerably. In this spectrum, the absorption of ice is very high.

The snow landscapes are one of the most difficult and dangerous places to access. Because of this, it has come down to remote sensing to obtain any kind of information.



Figure 8. Snow and ice landscapes

Depending on the dimensions of the snow grains, there are several types of snow. Each type has their own signature, that is, their own reflectance. On DART's database there are four types of snow: *fine*, *coarse*, *granular medium* and *medium to fine*. This classification is based on the size of the snow grain. Although they all have different values of reflectance, they do follow the same tendency. At a low wavelength, as the wavelength increases, the reflectance decreases.

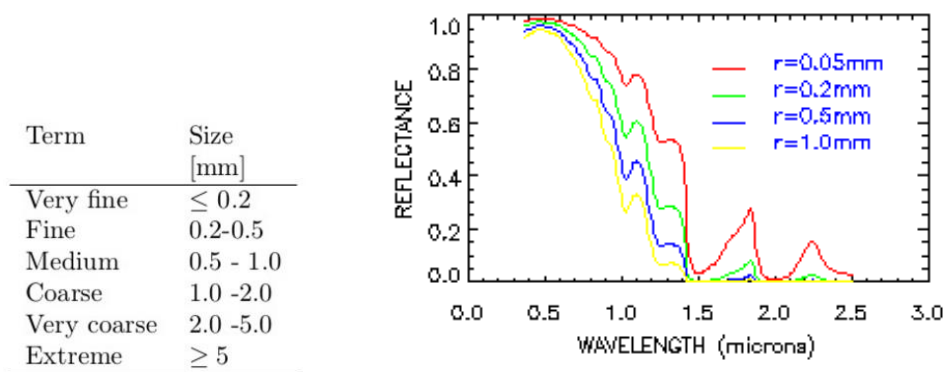


Figure 9. Reflectance of snow depending on the grain's size

3. PROGRAMS

For this project, two different programs will work alongside one another for the study of these scenarios. The first program is MATLAB that is very well known in the scientific field. This first program is the one the user will have to work on manually if there were to be changes either in the scene or in the simulation's properties. This program will also display the results of the different simulations.

The second program is DART created by CESBIO. It models the radiative transfer on the Earth's surface in visible to thermal infrared domain. It is used for simulating measurements of different types of off the ground sensors. This will come in handy in the study and monitoring of land surfaces from remote sensing measurements. That way, the user will be able to study different scenarios with different materials at different angles.

The main goal is to make the two programs interact. That is, to find a way to control DART by using MATLAB. Therefore, the user will not have the need to change the

parameters manually from DART. This will be achieved by reading and writing from *.xml* files which are found in DART's output folders of each specific simulation. The *.xml* files are the files where all the parameters needed to run the simulations can be found. According to the needs of the user, three different files will have the biggest interest and will be tampered with:

- *Phase.xml*: this file will be used to describe the type of radiation and atmospheric properties DART will simulate; there must be also written on this file the different wavelengths where the simulation will take place; here it's also found the different sensors with its corresponding coordinates and angle.
- *Coeff_diff.xml*: here there will be written the different materials that will make the simulated scene.
- *Wavelength.xml*: this file contains details of the simulation; it is needed in order to execute the sequence module since it has the wavelength values that will be simulated.

There is already a default folder in MATLAB with all the *.xml* files. These files will be copied in each scenario and altered with the desired characteristics. Doing so, there won't be no need to start from scratch in each simulation. On the following image it's displayed one of these files (*coeff_diff.xml*).

```
<?xml version="1.0" encoding="UTF-8"?>
<DartFile version="5.5.3">
  <Coeff_diff>
    <LambertianMultiFunctions>
      <LambertianMulti ModelName="reflect_equal_1
_trans_equal_0_0"
      databaseName="Lambertian.db"
      ident="Lambertian_Phase_Function_1" roStDev="0.000"
      useMultiplicativeFactorForLUT="1" useSpecular="0">
        <ProspectExternalModule
useProspectExternalModule="0"/>
        <lambertianNodeMultiplicativeFactorForLUT
          diffuseTransmittanceFactor="1"
          directTransmittanceFactor="1"
          reflectanceFactor="1"
          specularIntensityFactor="1"
          useSameFactorForAllBands="0">
          <lambertianMultiplicativeFactorForLUT
            diffuseTransmittanceFactor="1"
            directTransmittanceFactor="1"
            reflectanceFactor="1"
            specularIntensityFactor="1"/>
          </lambertianNodeMultiplicativeFactorForLUT>
        </LambertianMulti>
      </LambertianMultiFunctions>
      <HapkeSpecularMultiFunctions/>
      <RPVMultiFunctions/>
      <UnderstoryMultiFunctions integrationStepOnPhi="10"
integrationStepOnTheta="1" outputLADFile="0"
specularEffects="0"/>
      <AirMultiFunctions/>
      <WaterInterfaceMultiFunctions/>
      <Temperatures>
        <ThermalFunction deltaT="20.0"
          idTemperature="ThermalFunction290_310"
          meanT="300.0" override3DMatrix="0"/>
      </Temperatures>
    </Coeff_diff>
  </DartFile>
```

Figure 10. Default Coeff_diff.xml

DART can choose among three different methods for simulating light propagation. The radiative methods will be:

- *Flux-tracking*: this is the default method used in DART; a set of photons is emitted and the sensor measures how many of them return.
- *Monte Carlo*: a set of photons are tracked from the Earth's surface; the only measurement made is that of the reflectance.
- *LIDAR (Light Detection and Ranging)*: this method is most used for weapons ranging but it can also be thought as a tool in the measure of chemicals in the atmosphere and heights of objects in the ground; with this method, a light pulse, or a set of a number spaced light pulses, is emitted in the direction of a target and the sensor measures the light that bounces off the target (reflected light).

The modules that can be called on by MATLAB will be the following:

- *Dart*: this module simulates the rest of modules in sequential order, this is, the module will not be executed, instead MATLAB will directly call onto the other modules one by one.
- *Vegetation*: it generates the landscape of the scene.
- *Directions*: this module divides the space into a number of discrete directions.
- *Phase*: here it is set the optical properties of the given landscape.
- *Maket*: this one is in charge of simulating the scene with the different elements that can make up the scene; all the information will be gathered in the form of *.txt* file (maket.txt).

In order to generate the image, DART has two types of scanners to choose from:

- *Whiskbroom*: this type of scanner uses several detector elements which will be found aligned in track (or along track); by structuring them this way, the scanner will perform several scans in parallel and with each scan the direction can be reversed; it typically has a higher resolution than the pushbroom scanner.
- *Pushbroom*: this other type of scan has their detector elements aligned cross-track, as a result, this device scans the entire width of a surface in parallel as the platform moves; DART will make use of this last one.

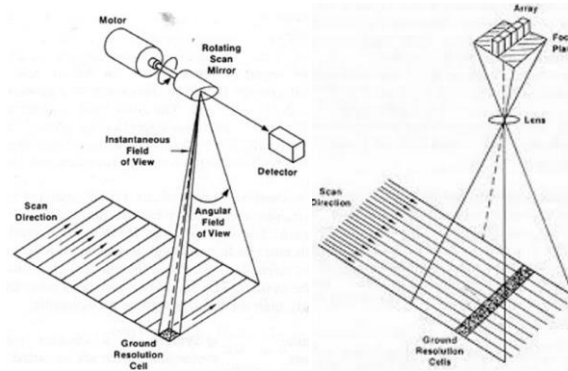


Figure 11. Types of scans in DART: a) Whisbroom b) Pushbroom

DART also can be configured to simulate pinhole cameras. These consist on a light tight box with a pinhole at one end and on the other the light sensitive material being watched. In the program, a matrix of locations will be generated followed by the generation of an image per location. When simulating LIDAR, this matrix will be a single value since there would be only one location. In any case, the computation time will be longer than with the pushbroom sensor.

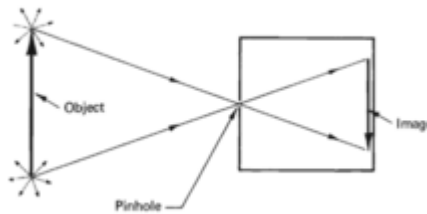


Figure 11. Pinhole camera

4. IMPLEMENTATION

The aim of this project is to implement a code in MATLAB so that DART may be controlled in the most optimized way possible. A code will be developed to run DART simulations directly through MATLAB. By doing this, the performance of the simulation will be optimized. Any parameter may be changed and any module may be executed directly from this program according to the needs of the user. All the information and images will be stored in the MATLAB folder so they may be visualized and manipulated if needed. This will be convenient for the study of the different materials that may compose the scene, as for the study of the effects caused by the topography and atmosphere.

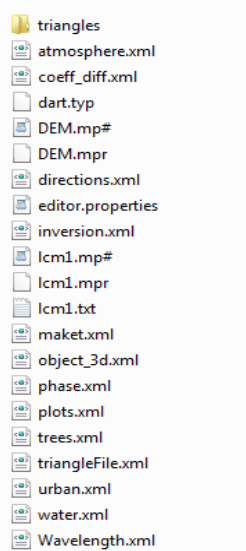


Figure 12. Example of the input files

As it was said before, all the input parameters from the DART modules will be stored in *.xml* files. The input files that are set by default in the folder *Default Simulation* will be copied and loaded in MATLAB in order to have a base to work with. For adding or changing specific parameters, these *.xml* files will be read and then written on according to the needs of the simulation. The files will be written with a tree hierarchy, therefore it is important to pay close attention to the position of the parameter being created or changed. In the figure 14, there is an example of the input folder with all these files. In this case, the file belongs to the folder of a snow scenario.

```
# Name: Coarse Granular Snow
# Type: FROST, Snow and Ice
# Class: Snow
# Subclass: Coarse Granular
# Particle Size: 178 micrometers effective size.
# Sample No.: Original filename COARSE.SNW.
# Owner: Dept. of Earth and Planetary Science, John Hopkins University
# Origin: collected at John Hopkins University IR Spectroscopy Lab.
# Description: Coarse snow. The spectrum from 0.3 to 2.08 micrometers was modeled, while that from 2.08 to 14 micrometers was measured
(see Introductory text)
# Measurement: Directional (10 Degree) Hemispherical Reflectance
# X Units: wavelength (micrometers)
# Y Units: Reflectance (percent)
#
wavelength;reflectance;direct_transmittance;diffuse_transmittance
0.3;95.2613;0.0;0.0
0.31;95.437;0.0;0.0
0.32;95.725;0.0;0.0
0.33;95.883;0.0;0.0
0.34;96.0984;0.0;0.0
0.35;96.2896;0.0;0.0
0.36;96.4234;0.0;0.0
0.37;96.5494;0.0;0.0
0.38;96.7268;0.0;0.0
0.39;96.843;0.0;0.0
0.4;96.9874;0.0;0.0
0.41;97.152;0.0;0.0
0.42;97.2646;0.0;0.0
0.43;97.4454;0.0;0.0
0.44;97.5054;0.0;0.0
0.45;97.849;0.0;0.0
0.46;97.9078;0.0;0.0
0.47;97.8732;0.0;0.0
0.48;97.7808;0.0;0.0
0.49;97.7362;0.0;0.0
0.5;97.634;0.0;0.0
0.51;97.5818;0.0;0.0
0.52;97.5276;0.0;0.0
0.53;97.462;0.0;0.0
```

Figure 134. A txt in one of DART's database (Lambertian.db) describing coarse snow

4.1. Creating the Scene

Different scenarios have already been provided in this project in order to validate the performance of the code. Since snow landscapes are the main interest of this work, the

elements used will be of snow and of clouds. In order to create a scenario and be able to execute the DART modules, the first step is to provide to the scene a set of input parameters. It needs to be specified the range of wavelengths where the simulation will work on, the topography of the scene, and the type of atmosphere that will be taken into account. It is also required to specify the different materials that will appear in the scene. Since this all has to do with long distance sensing, the study of the materials will be done by the use of the signature that each material has. These measurements are stored in pixels. Each pixel will contain a single material.

The signatures of the materials assigned have to be in the DART's database in order for the program to carry out the execution. These materials are separated into groups depending on their nature. Each group has its own set of parameters that describe the material. In DART, there are already installed databases that describe the different properties of some materials. The materials are divided in different groups: *vegetation*, *fluids*, and *soil and urban structures*. This last group is made up of different types of houses, roads, and plain walls that will be simulated.

Even though each material has its own spectral signature, the overall behavior of the material will depend on the group as it is clearly seen in the following figure:

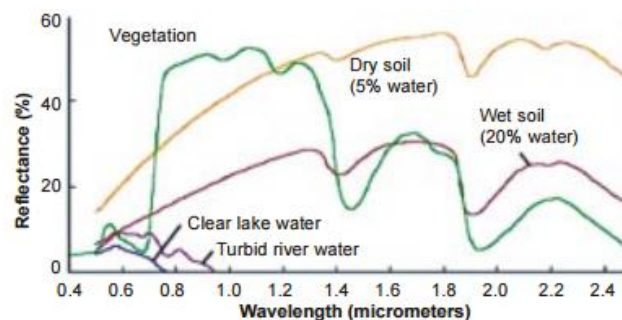


Figure 14. Reflectances of different types of material

Once this has been implemented, the next step will be to simulate the radiative transfer. That is, to read the radiation that is reflected from the Earth's surface. All the materials will be assumed as Lambertian. Since these are just testing simulations to see if the code has been well implemented, the simulation will be done with 20 wavelength bands. For a big number of wavelengths, a computer with lots of processing power will be needed for there are many commands being executed, each one having to perform numerous calculations and the amount of data is too great.

The code in MATLAB will also be altered in order to simulate the sensors at different angles. The user will be able to choose the exact angle or range of angles for the scene. For that, the file *phase.xml* must be renovated.

For visualizing the results it is important to install in MATLAB the hyperspectral tool *imshow3D*. This tool displays 3D grayscale images from three perpendicular views.

4.1.1. Snow simple

The study of the scenarios will begin with the execution of the simplest scene. This image will consist of three different materials distributed consecutively along the scenario.

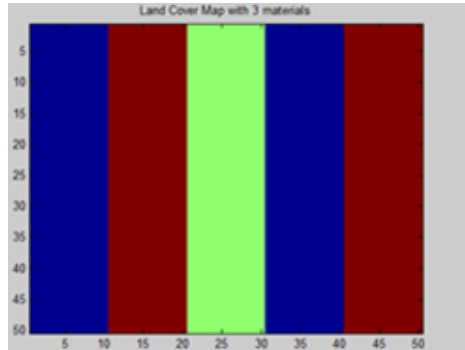


Figure 15. Materials of the scene

As it can be seen in figure 16, there are three different materials in the scene. The first approach will be the study of the materials without taking into account neither the topography nor the atmosphere. The topography of the scene will also be displayed, as it is seen in the figure 17. Since it is null, there will be only a flat surface.

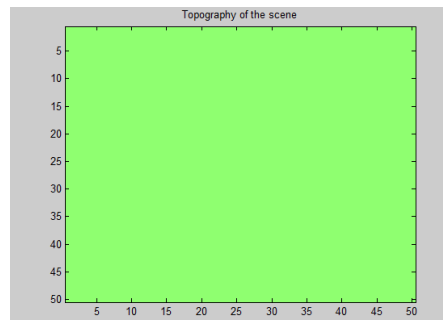


Figure 16. Topography of the scene for a flat surface

In figure 18, it is shown how each material has its own reflectance value for each wavelength in grayscale. Each pixel has a sampled spectrum of the material in the scene. In this case, the different materials can clearly be distinguished from one another. It is also noticeable that they react the same way to the value of the wavelength. As stated before, the figure also shows how the maximum reflectance is found at the first wavelengths. As the wavelength value increases, the reflectance decreases, since the program is simulating types of snow.

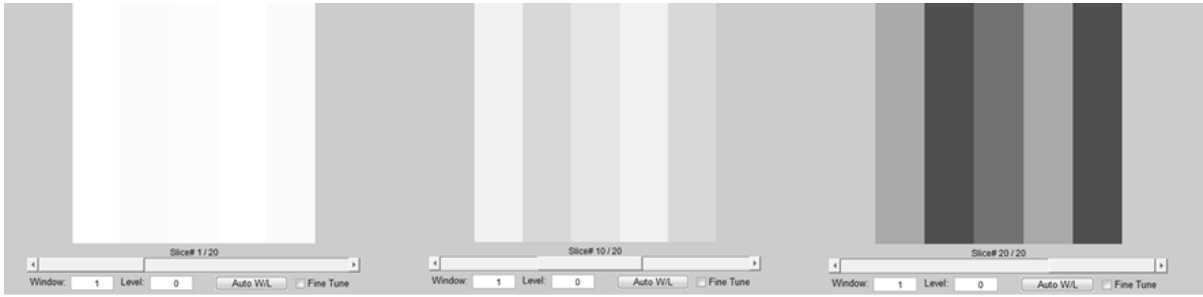


Figure 17. Hyperspectral Image at three different wavelength values

In figure 19, it is displayed both the original spectrum of the scene and the spectrum of the scene that DART simulates. The spectrum is the distribution of the reflectance of the snow of the scene as a whole as a function of the wavelength.

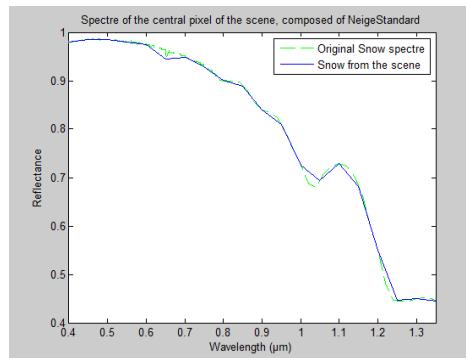


Figure 18. Reflectance of an original snow scene vs the simulated scene

Now certain topography will be given to the previous scenario. There are many types of topography that can be set. In figure 20, three different topographies will be displayed. The highest places will be represented by the color red. As the height decreases this color travels from dark red to dark blue when a minimal height is found. One of the examples used (the first one) has two high points separated by a low surface. This corresponds to a scene with two mountains. The other two examples only have one mountain, each one with a different size. The scene with two mountains will be also displayed in DART using the 3D viewer, as figure 21 shows.

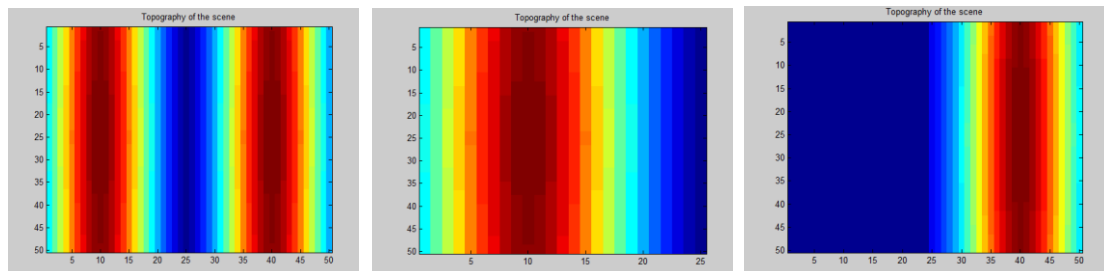


Figure 19. Different topographies

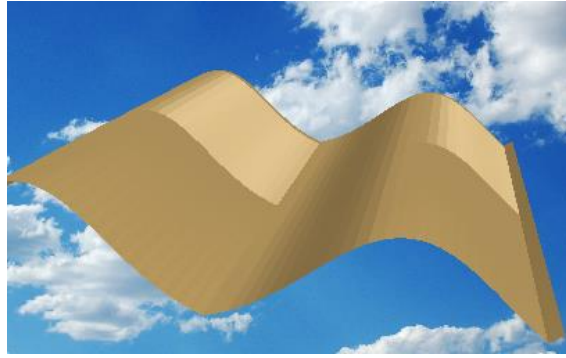


Figure 20. Topography seen from DART

In figure 22, it can be seen how a not null topography influences on the reflectance of the materials. But the behavior of snow remains the same. The reflectance is still highest at the first wavelength and after that it has a decreasing tendency.

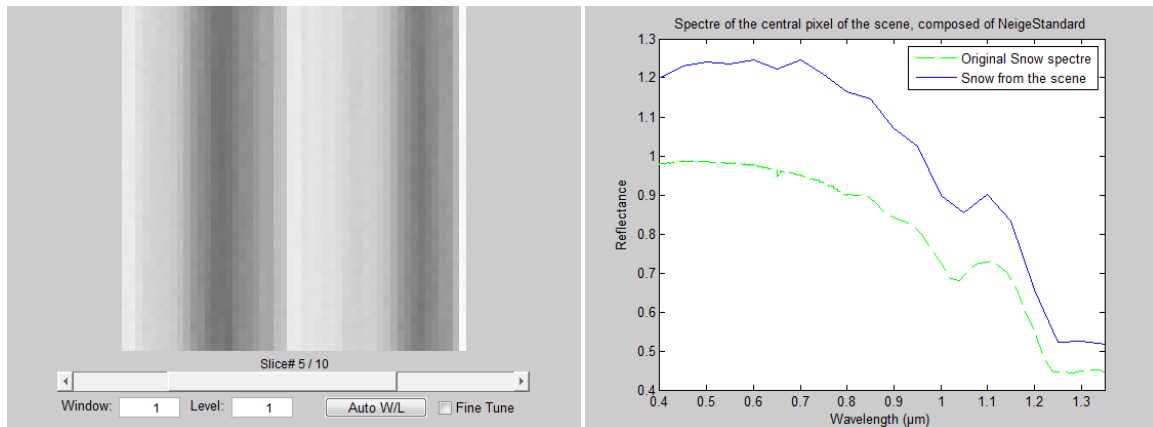


Figure 21. Simulating topography without atmosphere: a) Hyperspectral image b) Spectrum original scene vs simulated scene

After adding the effects of the topography, it can clearly be seen how the reflectance has the same tendency as before, but with an offset. The scenario has been set with topography but null atmosphere. The next step will be to add the atmospheric effects.

As the effects of the atmosphere are added, the resulting image becomes more distorted. It no longer has that smooth view of the bands. It is also visible how the resulting scene spectre differs when the atmosphere is taken into account. Due to this, whenever an unmixing is to be performed, by using the spectral signatures library of the different components that make the scene, these effects must be corrected beforehand.

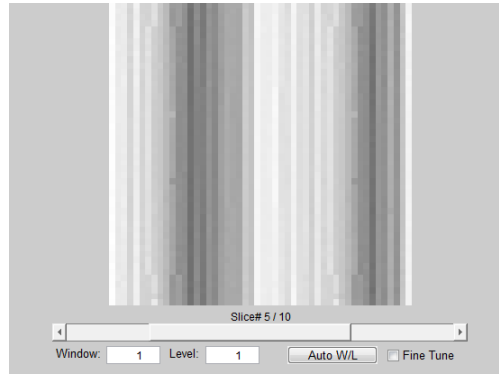


Figure 22. Simulating topography and atmosphere: hyperspectral image

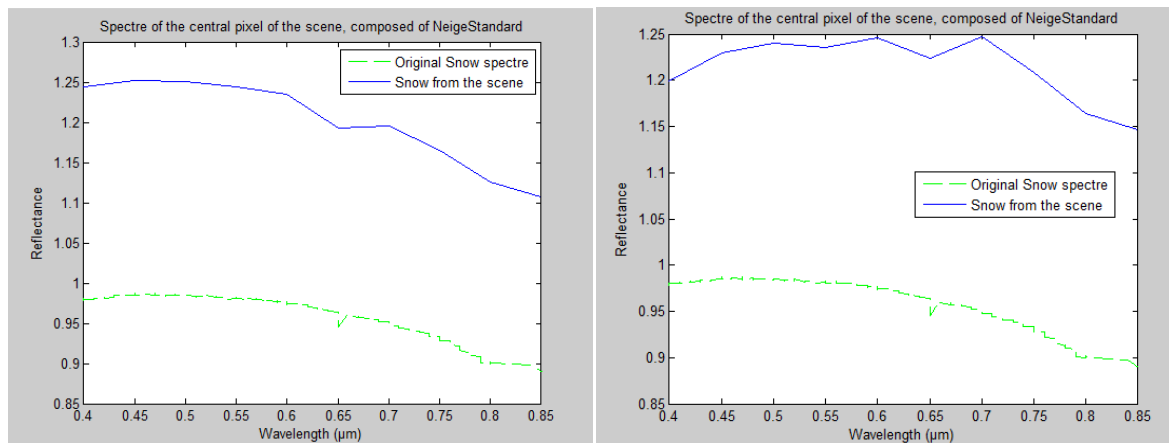


Figure 23. Simulating topography: a) Without atmosphere b) With atmosphere

4.1.2. Importing materials

Once the program has been tested, the next step will be to generate the most coherent scenes possible, that is, to upload the information of different materials if needed.

For now, MATLAB has only been working with one of the databases already created in DART's database manager, *Lambertian.db*. The next step will be to try to generate a new set of materials, that is, a new database using MATLAB. That way, scenes can be simulated with real measurements that have been taken from a specific location. The information will be provided to MATLAB in the form of an image. It will be the information about different kinds of snow, four to be precise. These four types of snow will form different combinations. At the end, what is given to the program is an $N \times N$ pixel image. The amount of snow contained will depend on where the element is found in the matrix. The purest elements will be those found in the four corners. As one travels towards the middle of the matrix, the mixture of the elements tends to be homogeneous. Each element will have a certain reflection value for the different wavelengths. This image will be displayed in MATLAB and then compared to the simulated imaged created by using DART.

Since the main focus of this project is to work with snow, the parameters needed for our database will be the wavelength, the reflectance, the direct transmittance, and the indirect transmittance. These last two values will be null in this case.

The new database will be placed in the folder *database*, which can be found in the folder *user_data* with the name *OurOwnDatabase*. When any of the elements of this database is called on, they must be called on by its proper name, that is, with the name given in the database *Manager*. The name that will be given to the element and the name of the element while loading it to the new data differ. This last name is used to determine the type of data that the database Manager will receive. For example, while uploading the data the program will call onto the snow *2D-LAM_Samples_layers610*. With this name it is meant to call a two dimensional lambertian object. Meanwhile, DART will store it as *samples_layers610*.

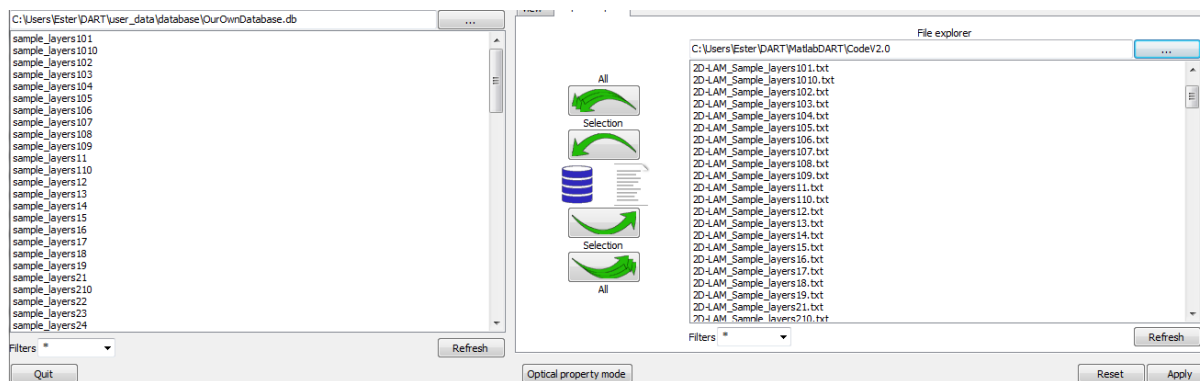


Figure 24. DART's Database Manager

The resulting image that the program will generate will not be an exact replica of the given image. This is shown in the variable *distance*. This variable is equivalent to the difference between the given image imported and the image simulated with the given data. This significant difference in some of the bands is because the DART module is working with sequences of wavelengths equally spaced, as opposed to the data provided. This means that some of the wavelengths that DART is simulating would not be the same as the wavelengths of the material. It is also important to keep in mind that there may also be a small error generated by the simulation. For a more visual outlook, figure 26 shows the variable distance as a 3D hyperspectral image.

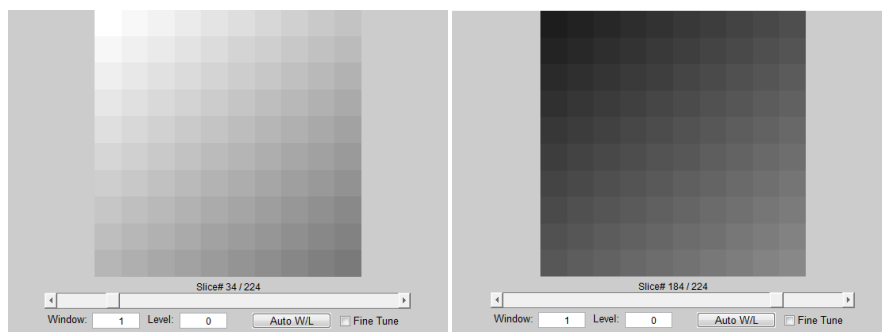


Figure 25. Variable distance

In the 3D plot, it is found the difference between the real image and the one that has been created. In most cases there will be a slight error (dark pixels) but in some wavelengths

the values will be quite big (light pixels).

4.1.3. Capturing images

DART uses the *ILWILS image* format. That means, a binary file will be created in the form of *.mp#* and also a *.mpr* header file. MATLAB will use the *.mp#* file for the display of the different images.





	ima01_VZ=000_0_VA=000_0.gr#	28/02/2016 17:24	Archivo GR#	1 KB
	ima01_VZ=000_0_VA=000_0.grf	28/02/2016 17:24	Archivo GRF	2 KB
	ima01_VZ=000_0_VA=000_0.mp#	28/02/2016 17:24	Archivo MP#	20 KB
	ima01_VZ=000_0_VA=000_0.mpr	28/02/2016 17:24	Archivo MPR	1 KB

Figure 26. Example of the output image

The next step will be to gather information from an image already defined from DART. That way, the user will be able to manipulate from MATLAB the outputs that had already been stored in DART. For that, a scene must be previously generated from DART. This will be done manually. The scene created will be that of an urban scene made by houses and trees.



Figure 27. Urban scene in 3D and in 2D aerial view

As it is appreciated in figure 28, the second image (which is the 2D aerial view of the scene) does not have such high resolution. Nevertheless, it is easy to distinguish the different elements in the scene. There are the darkest elements, which belong to the trees that have been implanted. There is also another vegetation plot which can be recognized as the dark grey rectangle. Finally there are also the small rectangles which belong to the different houses. Some houses have a flat roof; therefore it will be seen as a single color rectangle. The other houses have different shades of grey; these belong to particular shapes of the roof. They can either have a classic roof (two slopes) or a complex roof (four slopes).

4.2. Imported Scenario

In this scenario hyperspectral imaging sensors will be simulated. These sensors simultaneously capture both the spatial and spectral content of remote scenes. The result will be a hypercube (3D dataset) composed of layers of grayscale images.

Until now, several images have been recreated from afar in order to be able to study the effects that may counteract with the image created. This is done keeping in mind that it falls down to the sensor to obtain these images from afar.

As opposed to the image that was generated by DART, sensors see a limited version of the image. DART can be thought of as an ideal sensor. Therefore, in order to simulate an image captured by a sensor, the number of samples that will be taken into account will be compressed from infinite to finite.

Firstly, a scene will be created manually in DART, this image will have full resolution. Then a sensor simulated by MATLAB will acquire the image from DART. That sensor will be the one that will compress the samples, after windowing. There are many specific sensors that can be simulated. Sensors can also be simulated directly from DART, but only a small set of specific sensors. In order to simulate others that are not available in DART, it is required the use of MATLAB. This program will need a set of parameters from the specific sensors that can be found in the data sheets of each real sensor. For this project, three different sensors will be configured:

	IKONOS	QUICKBIRD	WORLDVIEW-2
Panchromatic Resolution⁴	1 m	0.61 m	0.46 m
Multispectral Resolution	4 m	2.44 m	1.84 m
Temporal Resolution	3-5 days	1-3.5 days	1.1-3.7 days
Radiometric resolution Collect	11-bit	11-bit	11-bit



Figure 28. Sensors: a) Ikonos sensor b) Quickbird sensor c) WV2 sensor (world view 2)

Firstly, the different sensors will display a scene that the program will create itself from scratch. This image will that of a street viewed perpendicularly to the ground. The sensor would have to distinguish an urban area with man-made constructions such as cement or rooftops.

These sensors will use the *pansharpening* method. Pansharpening is the process of

⁴ The corresponding resolutions are those found at nadir. Nadir is the point found directly below the observer. This point is diametrically opposite the zenith.

combining high-resolution panchromatic and lower resolution multispectral imagery in order to create a single high resolution color image.

After executing the simulations, the program will display three different images of the scene.

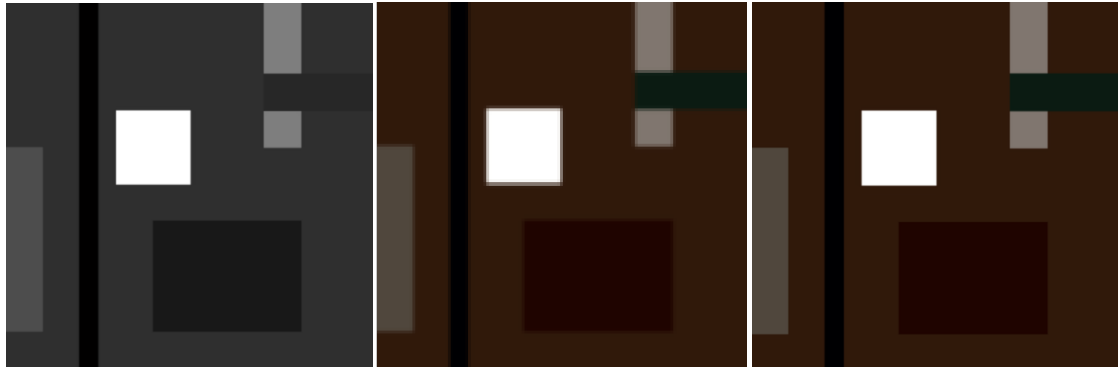
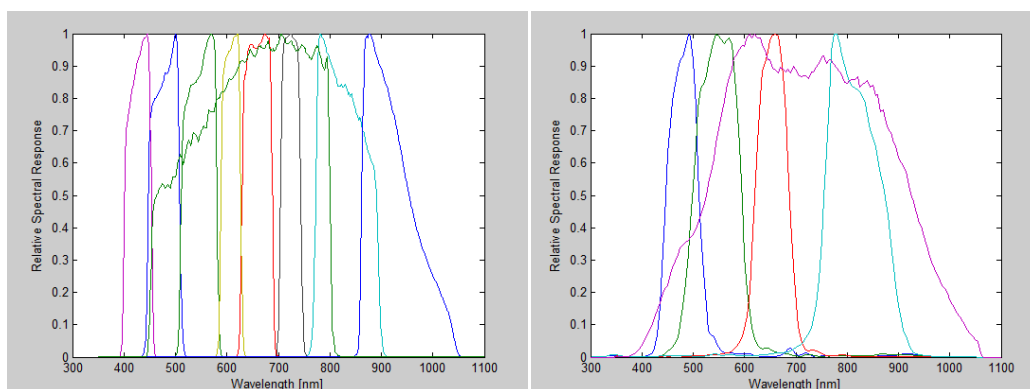


Figure 29. View of urban area with three different sensors: a) Panchromatic b) Multispectral c) Global

As it can be seen in figure 30, the first sensor (panchromatic) sees the image in grayscale. Though there is a loss of color, the image has high resolution (the scene is very clear). On the other hand, the second sensor (multispectral) sees the image in color, but it is blurry since there is less resolution.

The combination of the two sensors (one with high resolution and the other with color) will result in a very clear colored image as it is shown in the third image.

As stated before, there is a large variety of sensor to choose from, but in this case the program will be configured to simulate only three sensors.



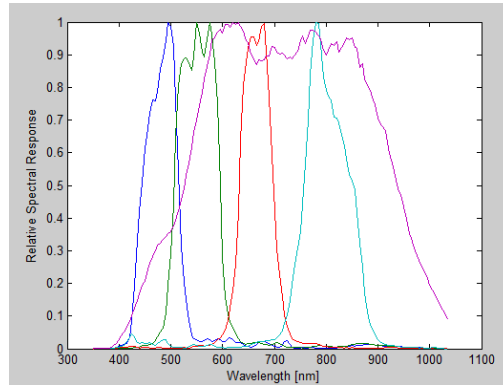


Figure 30. Relative responses normalized of the sensors: a) WV2 b) Quickbird c) Ikonos

From the point of view of spectral responses, the Ikonos and Quickbird sensors are the most similar, as it is appreciated in figure 31. The third one, WV2 has the most number of channels.

After seeing that the program simulates effectively the different sensors, the next step will be to simulate a sensor from an imported image from DART.

4.2.1. Importing images

Now the images studied will have been already created directly in DART instead of having to create a scene from scratch. In this case, it will be a simple image with a few houses and trees. With DART the houses generated will be able to have different sizes and there will be three types of roof: plain, simple and complex. In this case, all three types will be introduced in the plot. There will also be a rectangular vegetation plot besides the incorporation of trees. These trees can either be set in a determined location or the location can be random. For the location doesn't affect the upcoming result, the option random location will be chosen.

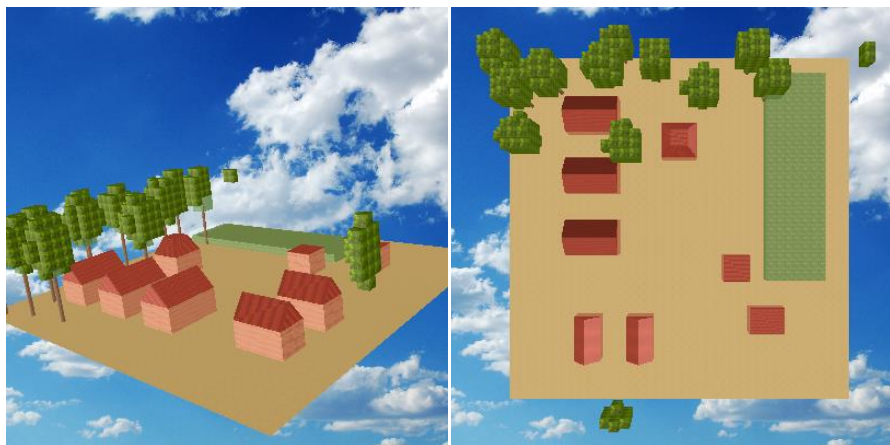


Figure 31. Image seen in 3D from DART

The sensors will capture the image from an areal view in 2D. This image has been made simulating an ideal sensor, this will be followed by the simulation of the real sensor.

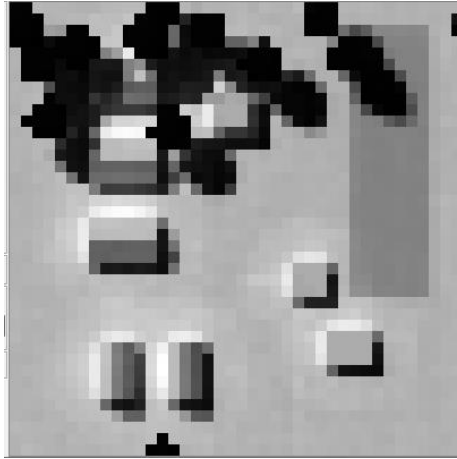


Figure 32. Image seen in 2D from DART



Figure 33. Global image seen by the sensor Global

In the figure above it's shown the image seen by the WorldView II sensor. As it has the most spectral bands (8 bands), the resolution will be best than with the other two sensors which only have half the number of bands. It is clear that this image has quite low resolution. Nevertheless, the components that make up the scene can be somewhat distinguished. In order to simulate in DART, 300 bands have been used along the entire sensor spectra, that is from 0.4 to 1.0 μm .

For obtaining a finer image, DART would need to work with a larger number of bands. Therefore, the same simulation has been created at a smaller range. This range only covers one of the sensor's channel (the red channel which is at 700nm). With this, the number of bands simulated per wavelength has been increased. As a result, the multispectral image has this purplish color. In this new case, the images created are finer than the ones which covered the entire spectrum. As it is shown on the figure below, more characteristics can be appreciated like the different shades generated by the trees and houses and also the roofs' form. It still won't be as clear as the images which are visualized directly from DART. In any case, since this simulation was done at the red channel, it can be well appreciated how the combination on the panchromatic and the multispectral image result in a high quality colored

image.



Figure 33. Image seen by one channel of the sensor: a) Panchromatic b) Multispectral c) Global

4.3. Unmixing

For accurate estimation it is required an unmixing. Pixels are assumed to be mixtures of different materials, these are called *endmembers*. The unmixing will be done with a scene with four different materials, which are all a different kind of snow. Out of the combination of the four, 100 new materials have been made with different percentages of each material. The respective spectral signatures of this new set of materials will be stored in a new database in DART.

Unmixing can be challenging because of numerous factors such as observation noise, the materials variability, environment conditions, etc. Since DART uses linear mixing when simulating, for obtaining the different endmembers the procedure used will be linear unmixing algorithms. Assuming the linear observation model, the program will calculate the abundance of the different endmembers. The linear observation model is an approximation of the spectral functions in the wavelength domain. In this case, it would be the reflectance function and the abundance of the material.

In order to obtain the amount of each material in the scene, the first approach will be by directly using the equations:

$$P(\lambda) = \sum_{i=1}^m R_i(\lambda) \cdot a_i$$

where m is the number of materials in the scene, $R_i(\lambda)$ is the reflectance of the material i at the specific wavelength λ , a_i is the abundance of the material i in the scene, and $P(\lambda)$ is the value of the global reflectance of the pixel at the wavelength λ . The $[R_i(\lambda)]$ matrix will be that of a 4x4 since there four materials present. If the number of materials were N , a $N \times N$ matrix would be used. A simple way to obtain the scene proportions would be using the inverse matrix.

Since the input images are mere matrix with a certain reflectance value for each coordinate, for unmixing on a pixel level, the amount of different pixel values found in the image would be counted. The result will be the number of materials present in the scene. After, the number of pixels that have each value will be also counted. This will provide the amount of each material.

Hyperspectral unmixing refers to any process that separates the pixel spectra from a hyperspectral image into multiple endmembers (single spectral signatures). Although, as it has been stated before, the method used will be the linear unmixing method. It is important to remember that the reflectance is usually a non linear function of the mass of the material. Therefore, the unmixing will be an estimation of the amount of material in a given space.

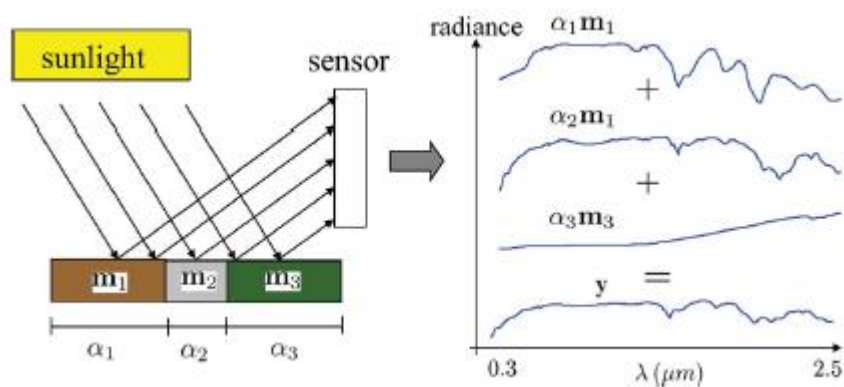


Figure 34. Linear unmixing

4.3.1. Simulating ice and soil

The first unmixing will be of materials whose spectral signatures differ greatly will be carried out in first place. This way, the efficiency that DART has at simulating mixed pixels will be tested. To begin, the materials from one of DART's databases (Lambertian.db) will be used. MATLAB refers to these materials as it follows: Sandy loams⁵ (A), weathered asphalt (B), red brick (C), and ice (D).

In DART, a material for the entire scene plot should be set. It is essential that this material doesn't interfere with the calculations. Therefore the material used will be transparent. It will be created and loaded to the database, covering the entire scene.

For this test trial, two different combinations of the four materials in the pixel will be set. If expressed as vectors (% of A, % of B, % of C, % of D), one combination will be (25, 25, 25, 25) and the other will be (50, 25, 25, 0).

⁵ It is a type of soil used for gardening. It is normally made up of sand along with varying amounts of silt and clay.

In this case, there will be different types of materials with different wavelength ranges. As it is seen on figure 36, the spectra are different from one another and the reflectance of the different kinds of soil differs greatly that from ice. It is also found that one of the materials (weathered asphalt) presents a very small wavelength range. Because of that, the unmixing will take place at a range where all the materials have a given reflectance value in DART's database.

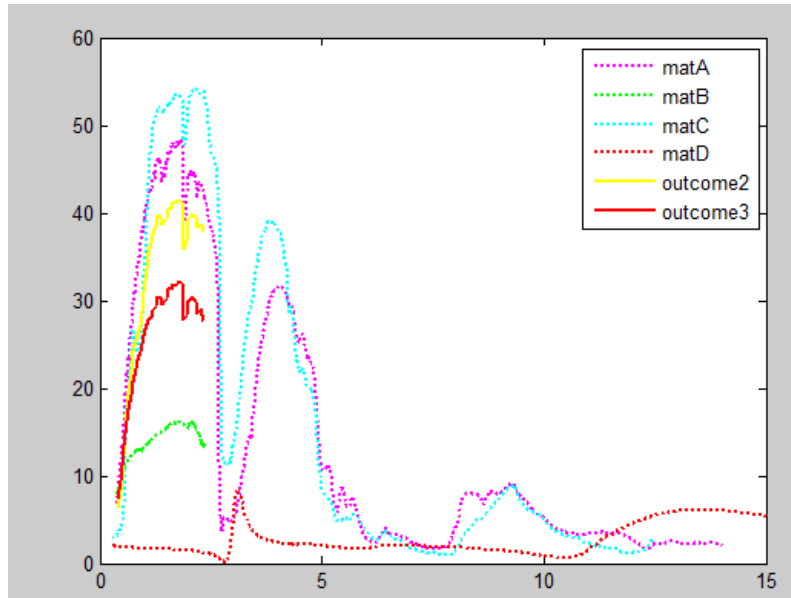


Figure 35. Spectra of the different inputs and outputs of the simulation, in μm

As it is seen on figure 37, an optimum range of simulation should be between 0.5 and 2.4 μm . This range is that of the visible and infrared regions. At this range, the four materials (dotted lines) follow the same tendency. The two continuous lines represent the reflectance of the scene for different combinations of the materials.

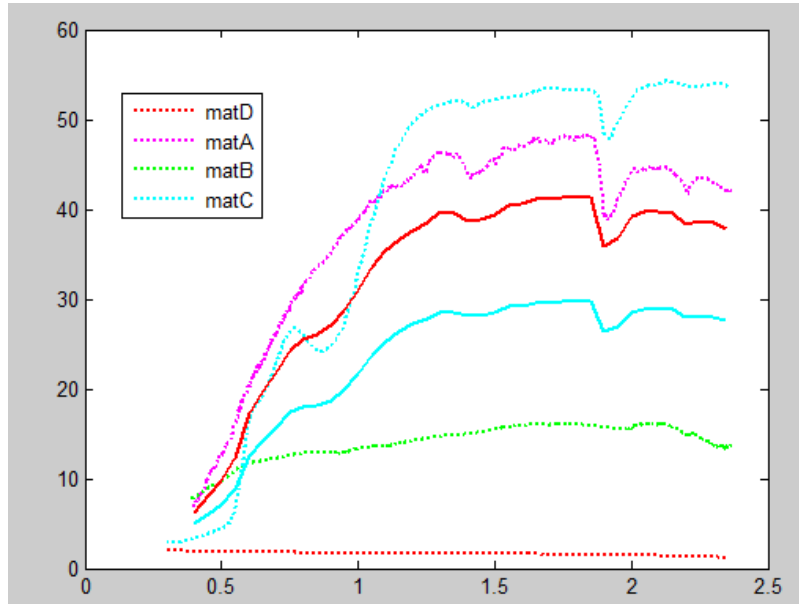


Figure 36. Spectrums of the different inputs and outputs of the simulation at a short wavelength range, in μm

Out of all these values, only four will be selected for the linear equation solving. Those values which are more accurate will be looked for. In order to do this, the MATLAB function $\text{cond}(A)$ will be used. This function has as an output a single value that measures the accuracy which the specified matrix will have when calculating its inverse or when solving a linear equations system. The shorter this value is, the better is the performance. The points that will give a value of a 10^3 factor will be chosen. On figure 38, one of the possible matrix that have this condition can be seen.

A =

6.5500	10.9200	1.8100	15.9669
26.4800	12.6100	1.7600	29.0960
43.6000	13.6700	1.7000	42.0095
52.7400	15.8700	1.6000	46.8460

Figure 37. Matrix of endmembers

Once all the inputs are loaded in MATLAB, the next step will be to solve the equations system. The resulting proportion that the program calculates is practically the same as the one that was placed in the scene. For both simulations the values presented on figure 39, the results are (25.00, 25.01, 25.00, 24.99) and (49.98, 24.98, 24.99, 0.04).


```

prob3 =      prob2 =
      0.4998      0.2500
      0.2498      0.2501
      0.2499      0.2500
      0.0004      0.2499

```

Figure 38. Abundancy of each material present in both simulations

There is another way to confirm the programs accuracy thanks to MATLAB because it can calculate the pixel reflectance value using the endmember matrix and the proportions obtained. It can be seen on figure 40 that the input value loaded from DART and the one calculated from MATLAB after the unmixing are the same.

```

S =
      8.8117      17.4865      25.2449      29.2640
ans =
      8.8117
      17.4865
      25.2449
      29.2640

```

Figure 39. Pixel reflectance value from DART (S) and from MATLAB (ans)

4.3.2. Simulating ice and snow

The next step will be to change the materials to those most interested in. In this case, those materials will be different types of snow. The reflectance of different types of snow is found in DART's database (also in Lambertian.db). The snow provided is that of different radius grain size. This data is equally spaced therefore it will be easier to select points in the spectrum for the signature matrix.

Since dealing with another type of material, it will be necessary to change the points in the matrix of spectral signatures so they will match the wavelengths of the scene spectrum. It is important to find the exact points that are contained in all the materials. Even if the wavelength error is minimal (0.0001 μm), the results obtained would be too far off.

In this simulation the different materials will be referred as: snow (A), snow fine (B), snow coarse (C), and ice (D). A scene with four materials equally distributed will be created.

As it was seen on figure 39, there are three of the materials which are very similar to one another. These are the three different types of snow which are categorized by the size of the grain. For they are snow, they have the same tendency. And when bigger grain radiuses are

taken into account, the absorption area increases. As a result, the decrease of reflectance will be greater for snow with a bigger grain radius. This difference in the radius is most significant at the range of 1.5 to 2.5 μm . The maximum reflectance is found at the shortest wavelength. As the wavelength increases, the reflectance decreases. There are some peaks at two wavelengths, 1.82 and 2.24 μm . In contrast, the fourth material is ice. Ice has a very low reflectance which is almost constant in the majority of the spectrum. The reflectance behavior changes considerably in the infrared region. To be more precise, the sudden change is found at the mid-infrared region (2.5 – 25 μm). At 2.84 μm , it is found that the reflectance drops to lower than 10% of the mean average. This is due to the effects of absorption which is lowest at this wavelength. Right after, there is the maximal absolute value at 3.08 μm , which is considerably low compared with the snow.

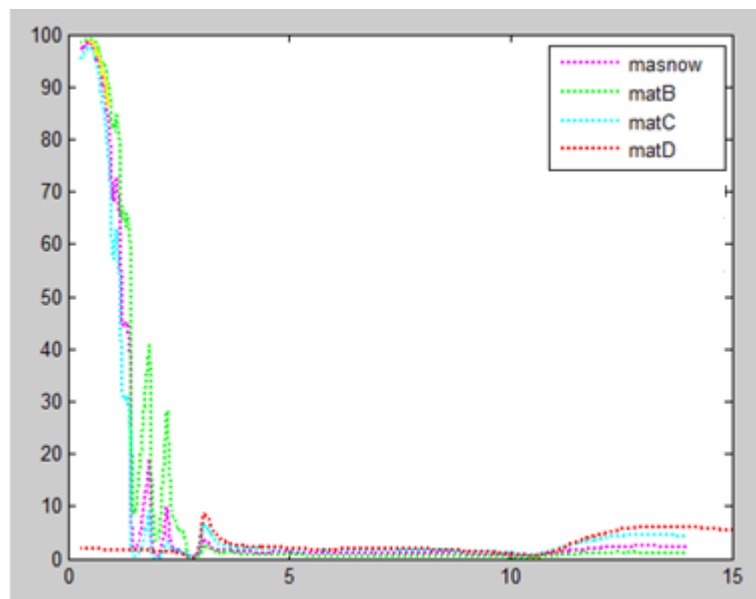


Figure 40. Input and output reflectance for two types of snow, in μm

On figure 42, the ice and the different types of snow can be seen at a shorter range at wavelengths corresponding to visible light and infrared light.

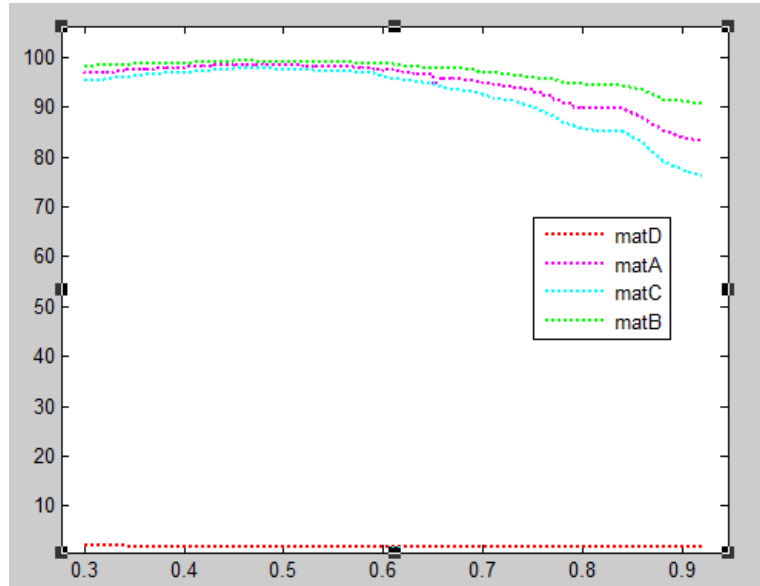


Figure 41. Spectra of different types of snow at a short wavelength range, in μm

At the first range, visible light, all the snow has almost the same reflectance, which is very high. More than 95% of the snow is reflected at this range, giving them that pure white color. Meanwhile, by coming across the red color of the visible light and heading towards the infrared region it is noticed a decrease of the snow reflectance although it depends on the different types of snow. The reflectance is still very high (above 80% of the light is reflected). When dealing with thick snowpack crashed together, there is a high amount of light scattered through the snow. A lot of the red light is absorbed making the blue light dominant. As a result, snow tends to have a bluish color as it suffers more and more of scattering effects. It is appreciated this in places where large amounts of snow are compressed, as we can see on figure 43.



Figure 42. Effects of scattering in snow and ice

On the other hand, the reflectance of the ice can be considered as constant and very low. This is due to the fact that ice is a translucent material. The light passes through without difficulty but changing its direction. The reflectance falls considerably close to the

wavelength 320 nm. This is due to the fact that a maximum absorption at this wavelength is found. The reflectance then quickly stabilizes.

Once again a pixel will be generated but now with the different types of snow. The first simulation will be with just two of the materials and the second one will be with all four, in both cases with identical proportions. Then the linear unmixing will be implemented.

prob4 =	prob4 =
0.5000	0.2494
0.5000	0.2496
-0.0000	0.2497
0.0000	0.2513

Figure 43. Abundancy of each material present in the scene for both simulations

By multiplying by 100 each data presented on figure 44, the results are (50.00, 50.00, 0.00, 0.00) and (24.94, 24.96, 24.97, 25.13). The abundances obtained in each simulation are almost the same as the abundances that had been placed in the scene. The drawback is that there is a lot of data that must be set manually. The next step will be to try to speed up the process. To reduce the amount of data that the user must import will be the priority now. The wavelengths that all materials share in common will be found. If chosen other materials, the unmixing would generate wrong results since the wavelength values could be different. Keeping in mind that the parameters from the simulated image can be changed, the best values for step size and central wavelength will be found. This will be done focusing on the materials present in the scene.

By working with any set of values, it is necessary to change the sequence properties so that the four of the resulting wavelengths match the material wavelength. DART will be generating consecutive wavelengths. In order to find the maximum number of wavelengths present in all the materials and the simulation, both the step size should be smaller and the spectrum range amplified. By doing so, in a lot of cases, the same wavelength values as the materials would be obtained. But the computational cost would be too great, not only when simulating in DART but also reading the files in MATLAB. Therefore, it will be necessary to find out the wavelengths shared by all the materials without adding more wavelength values to the simulation. The sets of data of the endmembers will be divided into groups. The most common step size for those short ranges of wavelengths will be calculated along with the first wavelength that encounters that value. With this, it will be easy to choose a set of values for the simulation which will generate the most coherent unmixing results.

For the first set of materials, and in order to get the most accurate results, it is seen that it is necessary to start the sequence at the wavelength 0.648 μm and it should have a size step of 0.002 μm .

5. CONCLUSION

For this project, two different programs (MATLAB and DART) have worked together for recreating the radiance exchange of information between the Earth's surface and the receiving sensor. DART can be executed with the use of MATLAB. This simulation done has been as if from an ideal sensor.

The program has been able to both generate scenes from MATLAB and import the scenes from DART. By doing this, the study of different scenarios has taken place. The main focus of this study has been of those scenes that involve snow. This study makes emphasis on the impact parameters such as the topography of a scene and assuming or taking for granted the atmosphere that is caused in the recollection of information.

Furthermore, a more realistic set of sensors has been successfully been simulated. Since the ideal sensor would be just a theory, the effects different that the specific sensors have on the images have been recreated for the display of those images.

Once all this had been achieved and the images had been set in MATLAB, the next part of the project could be carried on. This part corresponds to the unmixing of the resulting image. The method used was at a subpixel level. This program is effective when simulating ideal virtual sensors as does DART. When obtaining simulations from real sensors, effects of the intrinsic and extrinsic phenomena of the sensor will degrade the simulation's results. This should be addressed in future studies. Besides, this method can be used when there is a complete knowledge of the endmembers' matrix which it is difficult to obtain in most cases.

Due to the little time available for this study, the simplest methods have been used for the unmixing. With this method, a lot of data must be uploaded to MATLAB in the case of having more endmembers and the number of valid responses will be decremented as more endmembers are present in the scene. For future references, a more complex method should be worked with.

6. BIBLIOGRAPHY

- [1] <http://calval.cr.usgs.gov/wordpress/wp-content/uploads/pRanga14.pdf>
- [2] http://www.cesbio.ups-tlse.fr/index_us.htm
- [3] <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.403.6043&rep=rep1&type=pdf>
- [4] <http://www.umbc.edu/rssipl/people/aplaza/Papers/Conferences/2013.WHISPERS.Topography.pdf>
- [5] <http://www.crisp.nus.edu.sg/~research/tutorial/intro.htm>
- [6] <http://www.satimagingcorp.com/satellite-sensors>
- [7] <http://eoedu.belspo.be/en/satellites/ikonos.htm>
- [8] http://alienryderflex.com/sub_pixel
- [9] Schowengerdt - Remote Sensing, Models and Methods for Image Processing (Elsevier 3rd ed. 2007)
- [10] Michael Eismann-Hyperspectral Remote Sensing. PM210-SPIE (2012)
- [11] Elachi, van Zyl - Introduction to the Physics and Techniques of Remote Sensing (Wiley 2ed 2006)
- [12] [W._Gareth_Rees]_Remote_Sensing_of_Snow_and_Ice (BookZZ.org)
- [13] [James_B._Campbell_PhD,_Randolph_H._Wynne]_Intro (BookZZ.org)
- [14] RapportStageEN_GIPSA_RS_LAURENT_BLIN.pdf
- [15] <https://nsidc.org/cryosphere/snow/climate.html>
- [16] <http://cse21-iiith.virtual-labs.ac.in/exp3/index.php?section=Theory>

ANNEXES: MATLAB CODE

After every scene that is generated, the same line of code will be used for all the different simulations, although there are two different types of simulations: the simulation that creates the scene from scratch and the simulation already generated from DART.

Annex A. Simulation procedure for simulating images

BlackBox:

This function is used for the simulation of the scene. It doesn't matter what type scene it is, the procedure will always be the same. First, some parameters must be set, if they haven't been set value yet. Even if there is no specific value for the parameter, the parameters need to have been created beforehand. Once it's all set, the program will call on the DART to execute all the modules one by one.

Input:

- ➔ Structure Opt: This variable stores information from the scene. In this case, the needed information will be the name of the scene, the first wavelength values, the step and size of the wavelengths, the number of wavelengths in our simulation (opt.name, opt.spband.first, opt.spband.step, opt.spband.delt, opt.spband.nb). It is also required the number of pixels of the squared matrix plot and the size of each pixel(opt.N, opt.Celldim).
- ➔ mat: This variable consists on a vector with the different materials that make up the scene.
- ➔ lcm: an indexed matrix with the different materials in their respective coordinates.

Output:

- ➔ bands: the different wavelengths that will be simulated.
- ➔ IMG: this variable stores the matrix values that make the different images that DART has created.

```
function [ bands, IMG ] = BlackBox(opt, mat, lcm, topo, mat2, lcm2)%  
    %%%when generating a scene from scratch  
    %BlackBox Main function creating the whole simulation  
    %    Input elements  
    %    -opt : structure with all the necessary options :  
    %           "name" of the simulation  
    %           "N" size of the scene in pixels  
    %    "Celldim" Size of a pixel in meters in the scene (1 if not  
    %           specified)  
    %    "spband" structure with spectral bands options  
    %    -mat : array of structures  
    %    -lcm : indexed matrix with the occupation of materials (paired with  
    %           mat).  
    %    -topo : elevation matrix map
```

```

%
%      OPTIONAL INPUT :
% %      -lcm2
% %      -mat2
%
% % Version 2.0

% %% Falculative Arguments
if nargin<=5
    lcm2=0;
    mat2=0;
end

if nargin<4
    topo=0; end

if ~isfield(opt,'Nmat')
opt.Nmat=size(mat,2); end %Supposing mat is only an array

if ~isfield(opt,'Celldim')
    opt.Celldim=1; end

if ~isfield(opt,'N')
    opt.N=size(lcm);
    opt.Nx=opt.N(1);
    opt.Ny=opt.N(2);
else %En supposant que la scène est carrée.
    if ~isfield(opt,'Nx')||~isfield(opt,'Ny')
        opt.Nx=opt.N;
        opt.Ny=opt.N;
        end
    end
end

if ~isfield(opt,'atm')
opt.atm=1; end % Presence of atm by default

if ~isfield(opt,'nbiter')
opt.nbiter=3; end % Reduce number of iterations to speed up calculation

if ~isfield(opt,'cloud')
    opt.cloud=0; end %No clouds

if ~isfield(opt,'pinhole')
    opt.pinhole='0';end
if ~isfield(opt,'pushbroom')
    opt.pushbroom='0';end
%
% % Creation of the scene
opt.dartpath='../..\..\..\DART'; %% PATH TO CHANGE IF MATLAB FILES ARE MOVING
opt.defpath='../\DARTdefaultsimulation'; %% PATH TO CHANGE IF FILES ARE
MOVING

opt.path=[opt.dartpath,'\user_data\simulations\',opt.name];
copyfile([opt.defpath,'\DefaultSimulation'],opt.path);

DARToptions(opt); % Configure other DART options

if ~isempty(lcm)

```



```

        opt.lcm=1; % First Land Cover Map
LandCoverMap(opt,mat,lcm); % Launch Vegetation module to create plots
ExecuteDART(opt,'vegetation'); %Optional if only one lcm (redundant)
        end

        if topo~=0
Topography(opt,topo);
        end

        if opt.atm==1;
        Atmosphere(opt);
        end

        if opt.cloud~=0;
        Clouds(opt);
        end

        %% Execution of DART
        ExecutedART(opt,'full');

        % Sequence (spectral bands)
        Sequence(opt);

        % Create an hyperspectral image
        IMG=OpenAllImages(opt);
        % Array of Spectral Bands
        band = opt.spband.first;
bands=zeros(1,opt.spband.nb);

        for i=1:opt.spband.nb
            bands(i) = band;
            band = band+opt.spband.step;
        end

        end

```

Dart Options:

This function is used to set parameters in the *phase.xml* file. It will write the number of iterations will be used in the specific simulation. This function also controls the different sensors that can be configured to be at different positions and at different angles. There will be two types of sensors:

- *Pinhole*: As default, this sensor captures the entire plot, but the camera's orientation and position can be changed. The images can be visualized either with the camera field of view or with the sub-Earth scene.
- *Pushbroom*: This sensor manipulates the azimuth's angle and the platform azimuth.

Input:

- ➔ Structure Opt: This variable stores information from the scene. In this case the needed information will be the name of the scene and the number of iterations. (opt.name, opt.nbiter).

Output:

→ There will not be any outputs in MATLAB, but *phase.xml* will be modified.

```
function [ output_args ] = DARToptions( opt )
%DARToptions parameters some options for the simulation
%    can configure :
%        the number of iterations(opt.nbiter)
%        the captation from pinhole or pushbroom

%% Configure Number of Iterations (phase.xml)
xmlpathpl=[opt.path, '\input\phase.xml'];
xDoc= xmlread(xmlpathpl);

phase=xDoc.getElementsByTagName('Phase');
phase=phase.item(0);
dainpa=phase.getElementsByTagName('DartInputParameters');
dainpa=dainpa.item(0);
flutra=dainpa.getElementsByTagName('nodefluxtracking');
flutra=flutra.item(0);
flutra.setAttribute('numberOfIteration',int2str(opt.nbiter));

xmlwrite(xmlpathpl,xDoc);

%% Pinhole/Pushbroom (phase.xml)
dapr=phase.getElementsByTagName('DartProduct');
dapr=dapr.item(0);
damopr=dapr.getElementsByTagName('dartModuleProducts');
damopr=damopr.item(0);
brfprpr=damopr.getElementsByTagName('BrfProductsProperties');
brfprpr=brfprpr.item(0);

senimssim = xDoc.createElement('SensorImageSimulation');
phase.appendChild(senimssim);
senimssim.setAttribute('importMultipleSensors','0');

xmlwrite(xmlpathpl, xDoc);

if opt.pinhole=='1'
pinhole = xDoc.createElement('Pinhole');
senimssim.appendChild(pinhole);
pinhole.setAttribute('setImageSize','0');
pinhole.setAttribute('defCameraOrientation','0');
pinhole.setAttribute('cameraRotation','0');

senpin=xDoc.createElement('Sensor');
pinhole.appendChild(senpin);
senpin.setAttribute('sensorPosZ','1000');
senpin.setAttribute('sensorPosY','0');
senpin.setAttribute('sensorPosX','0');
end

if opt.pushbroom=='1'
push = xDoc.createElement('Pushbroom');
senimssim.appendChild(push);
```

```

push.setAttribute('importThetaPhi','0');

plat = xDoc.createElement('Platform');
senimsim.appendChild(plat);
plat.setAttribute('PlatformDirection','0');
plat.setAttribute('PlatformAzimuth','0');

senpu = xDoc.createElement('Sensor');
senimsim.appendChild(senpu);
senpu.setAttribute('PlatformDirection','0');
senpu.setAttribute('PlatformAzimuth','0');
end

xmlwrite(xmlpathpl,xDoc);
end

```

Execute Dart:

This function is used to execute any of DART's modules. There can be one of the following: Vegetation, Directions, Phase, Maket, Dart, Full (this is the module that will be used during the project since it runs all of the previous modules consecutively), Sequence (this module will be called alongside full).

Input:

- ➔ Structure Opt: This variable stores information from the scene. In this case the needed information will be the name of the scene and the location of the program DART (opt.name, opt.dartpath).
- ➔ The module that must be executed from the ones described above.

Output:

- ➔ There will not be any output in MATLAB. It will simulate the different modules and load the results in the output folder.

```

function [ out ] = ExecuteDART( opt, module)
%ExecuteDART Execution of the DART Modules
% The function executes DART modules specified by name module
(vegetation,
% directions, phase, maket, dart and full (and the modules) in the
% simulation folder (namefolder).
% The functions can not use dart-full.bat correctly, so it execute all
% modules successively
% Version 1.0
name=opt.name;
curdir = pwd;
cd([opt.dartpath, '\tools\windows\']);
switch module
case 'vegetation'
command=[opt.dartpath, '\tools\windows\dart-vegetation.bat ',name];
system(command);

```

```

        out=1;
        cd(curdir)
        return

        case 'directions'
command=[opt.dartpath, '\\Tools\windows\dart-directions.bat ', name];
        system(command)
        out=1;
        cd(curdir)
        return

        case 'phase'
command=[opt.dartpath, '\\Tools\windows\dart-vegetation.bat ', name];
        system(command)
command=[opt.dartpath, '\\Tools\windows\dart-phase.bat ', name];
        system(command)
        out=1;
        cd(curdir)
        return

        case 'maket'
command=[opt.dartpath, '\\Tools\windows\dart-maket.bat ', name];
        system(command)
        out=1;
        cd(curdir)
        return

        case 'dart'
command=[opt.dartpath, '\\Tools\windows\dart-only.bat ', name];
        system(command)
        out=1;
        cd(curdir)
        return

        case 'full'
% Calling dart-full doesn't work, so the modules are executed one
% by one instead
command=[opt.dartpath, '\\Tools\windows\dart-vegetation.bat ', name];
        system(command)
command=[opt.dartpath, '\\Tools\windows\dart-directions.bat ', name];
        system(command)
command=[opt.dartpath, '\\Tools\windows\dart-phase.bat ', name];
        system(command)
command=[opt.dartpath, '\\Tools\windows\dart-maket.bat ', name];
        system(command)
command=[opt.dartpath, '\\Tools\windows\dart-only.bat ', name];
        system(command)
        out=1;
        cd(curdir)
        return

        case 'sequence'
command=[opt.dartpath, '\\Tools\windows\dart-sequence.bat ', name,
        '\\Wavelength.xml -start'];
        system(command)
        cd(curdir)
        return

        otherwise

```

```

print('Did you mean "vegetation", "directions", "phase", "maket",
      "dart", "sequence" or "full" ?');
      out=0;
      cd(curdir)
      return

end

```

Sequence:

This function is used not only to call on the module Sequence, but it first prepares the files for the execution of the module. Firstly, the file Wavelength.xml will be copied from the folder Default Simulation. This folder contains all the default values for the simulations can be found. The copied file Wavelength will be placed in the folder of the scene, outside the input and output folders. This file will be altered by changing the range of wavelengths to the required by the simulation at hand. The step size between wavelengths will also need to be configured as the actually size of the wavelength.

Input:

- ➔ Structure Opt: This variable stores information from the scene. In this case the needed information will be the name of the scene and the values that have previously been defined and the location of both the scene folder and the default simulation folder (opt.name, opt.spband.first, opt.spband.step, opt.spband.delt, opt.spband.nb, opt.path,opt.defpath).

Output:

- ➔ There will not be any output in MATLAB. It will simulate the different modules and load the results in the Sequence folder.

Any error generated during the execution of the module will be saved in Wavelength_Sequence_error.log which is found outside of the Sequence folder (The majority of the time this error will be for a misplaced parameter in the .xml files).

```

function [ output_args ] = Sequence( opt )
%Sequence Create and execute a sequence with the specified spectral bands
%
% Input parameters (inside 'opt' structure) :
%   namefolder
% firstmeanlambda : First spectral band of the sequence
% stepmeanlambda : Interval between each spectral band
%   deltalambda : Width of the spectral bands
%   numberbands : Number of Spectral Bands
% Version 1.0

% Creation of the sequence file
% For simplicity, we copy an existing xml file with default values (cf
% captures) : delete as many useless files as possible (save space) ; only
% execute main processes (direction, phase, maket, dart)
copyfile([opt.defpath, '\Wavelength.xml'],opt.path);

xmlpath=[opt.path, '\Wavelength.xml'];

```

```

xDoc= xmlread(xmlpath);

desc=xDoc.getElementsByTagName('DartSequencerDescriptorEntry');
meanLambda=[num2str(opt.spband.first),';',num2str(opt.spband.step),';',int2
str(opt.spband.nb)];
desc.item(0).setAttribute('args',meanLambda);
deltaLambda=[num2str(opt.spband.delt),';0;'];int2str(opt.spband.nb)];
desc.item(1).setAttribute('args',deltaLambda);

pref=xDoc.getElementsByTagName('DartSequencerPreferences');
initialNode = pref.item(0);
ids = char(initialNode.getAttribute('numberParallelThreads'))
pref.item(0).setAttribute('numberParallelThreads', '1');
ids = char(initialNode.getAttribute('numberParallelThreads'))    %%we check
to see if we changed the parameter

xmlwrite(xmlpath,xDoc);

%% Execution of the sequence

ExecutedDART(opt, 'sequence')

end

```

Land Cover Map:

This function generates a land cover map of the scene. It will be used to generate the different plots that are found in the scene as the overall Earth scene.

Input:

- ➔ Structure Opt: This variable stores information from the scene. In this case the needed information will be the name of the scene, the number of pixels in the scene, and the number of iterations (opt.name, opt.N, opt.nbiter).

```

function [ output_args ] = LandCoverMap( opt,mat,lcm )
%LandCoverMap Creation of a Land Cover Map with the defined materials
% The function create an occupation map from the input parameters:
%
% -opt : options of the scene
% -mat : Array composed of the different materials (struct)
% -lcm : Indexed matrix image
% Version 1.0

if ~isfield(opt,'overwriteplot')
opt.overwriteplot=0; %keep (0) or crush (1) the existing plots
end

opt.Nmat=size(mat,2);
if opt.lcm==2 %%%%%%%%%%%%%%lo quit 
N=size(lcm);
opt.Nx=N(1);
opt.Ny=N(2);

```

```

end

%% Create raster image .mp#
imgname=[opt.path, '\input\lcm', int2str(opt.lcm), '.mp#'];
fich=fopen(imgname, 'w');
fwrite(fich, lcm, 'double');
fclose(fich);

%% Create .mpr file

mini=min(min(lcm));
maxi=max(max(lcm));

mprname=[opt.path, '\input\lcm', int2str(opt.lcm), '.mpr'];
imgmpr=fopen(mprname, 'w');

fprintf(imgmpr, '[Ilwis]\r\nDescription=\r\n');
fprintf(imgmpr, 'Time=0\r\n');
fprintf(imgmpr, 'Version=5.5\r\n');
fprintf(imgmpr, 'Class=Raster Map\r\n');
fprintf(imgmpr, 'Type=BaseMap');
fprintf(imgmpr, '\r\n\r\n');
fprintf(imgmpr, '[BaseMap]\r\nCoordSystem=unknown.csy\r\n');
fprintf(imgmpr, 'CoordBounds=0 -%d %d
0\r\n', opt.Nx*opt.Celldim, opt.Ny*opt.Celldim); %Dimensions of the scene
(in m) !
fprintf(imgmpr, 'Domain=value.dom\r\n');
fprintf(imgmpr, 'DomainInfo=value.dom;Long;value;0;-
9999999.9:9999999.9:0.1:offset=0;\r\n');
fprintf(imgmpr, 'Range=1.000:%d.000:0.001:offset=0\r\n', opt.Nmat);
fprintf(imgmpr, 'Proximity=1.000000\r\nType=Map\r\n');
fprintf(imgmpr, 'MinMax=%f:%f\r\n', mini, maxi); %Min and Max of values
fprintf(imgmpr, '\r\n');
fprintf(imgmpr, '[Map]\r\nGeoRef=%s.grf\r\n', 'lcm');
fprintf(imgmpr, 'Size=%d %d\r\n', opt.Nx, opt.Ny); %Size of the image (in
pixels) !
fprintf(imgmpr, 'Type=MapStore');
fprintf(imgmpr, '\r\n\r\n');
fprintf(imgmpr, '[MapStore]\r\n');
fprintf(imgmpr, 'StoreTime=0\r\nData=%s.mp#\r\n', 'lcm');
fprintf(imgmpr, 'Structure=Line\r\n');
fprintf(imgmpr, 'StartOffset=0\r\nRowLength=%d\r\n', opt.Nx);
fprintf(imgmpr, 'PixelInterLeaved=Yes\r\nSwapBytes=No\r\n');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%change to yes
fprintf(imgmpr, 'Type=Real\r\nUseAs=No\r\nFormat=2\r\n');

fclose(imgmpr);

%% Create the descriptor file

descname=[opt.path, '\input\lcm', int2str(opt.lcm), '.txt'];
descip=fopen(descname, 'w');

for i=1:opt.Nmat
fprintf(descip, '%s\t%d\t%d\t', mat(i).name, mat(i).index, mat(i).plot);
fprintf(descip, '%d\t%d\t', mat(i).plottype, mat(i).soiltype);
fprintf(descip, '%s\t%s\t', mat(i).soildb, mat(i).soilsp);
fprintf(descip, '%s\t%s\tg\t', mat(i).vegdb, mat(i).vegsp, mat(i).LAI);

```

```

fprintf(descip, '%g\t%g\t%g\t', mat(i).Hbase, mat(i).H, mat(i).Hsigma);
fprintf(descip, '%g\t%g\t', mat(i).Tmean, mat(i).Tdelt);

fprintf(descip, '%g\t%g\t%g\t%g\t', mat(i).clumpmin, mat(i).clumpmax, mat(i).cl
uma, mat(i).clumb);

fprintf(descip, '%d\t%g\t%g\t%g\t', mat(i).LAD, mat(i).ALA, mat(i).eccen, mat(i)
.LeafSize);
fprintf(descip, '\r\n');
end

fclose('all');

% Modify the XML files to implement LCM
%Plots

xmlpathpl=[opt.path, '\input\plots.xml'];
xDoc= xmlread(xmlpathpl);
% Open XML

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

plots=xDoc.getElementsByTagName('Plots'); % Get to the node "Plots"
plots=plots.item(0);
plots.setAttribute('isVegetation', '1');

importRaster = plots.getElementsByTagName('ImportationFichierRaster');
importRaster = importRaster.item(0);

vegetprop = xDoc.createElement('VegetationProperties');
importRaster.appendChild(vegetprop);
vegetprop.setAttribute('OverwritePlots', int2str(opt.overwriteplot)); %keep
(0) or crush (1) the existing plots
vegetprop.setAttribute('coverLandMapDescFileName', ['lcm', int2str(opt.lcm), '
.txt']);
vegetprop.setAttribute('coverLandMapFileName', ['lcm', int2str(opt.lcm), '.mp#
']);
vegetprop.setAttribute('selectSubZone', '0');

if opt.lcm==2
N=size(lcm);
Nx=N(1);
Ny=N(2);

vegetprop.setAttribute('selectSubZone', '1');
subzone = xDoc.createElement('SelectSubZoneProperties');
vegetprop.appendChild(subzone);

```



```

        subzone.setAttribute('lineOfTopLeftPixel','5');
        subzone.setAttribute('lineNbSubZone',int2str(Nx));
        subzone.setAttribute('columnOfTopLeftPixel','5');
        subzone.setAttribute('columnNbSubZone',int2str(Ny));
        end

    rastercos = xDoc.createElement('RasterCOSInformation');
        importRaster.appendChild(rastercos);
        rastercos.setAttribute('nbColCOS',int2str(opt.Ny));
        rastercos.setAttribute('nbLiCOS',int2str(opt.Nx));
    rastercos.setAttribute('pixelByteSizeCOS','9'); %double (Little Endian)
    rastercos.setAttribute('pixelSizeCol',int2str(opt.Celldim));
    rastercos.setAttribute('pixelSizeLi',int2str(opt.Celldim));

    xmlwrite(xmlpathpl,xDoc);

    if opt.lcm==1;
        %%Maket
        xmlpathma=[opt.path,'\input\maket.xml'];
        xDoc= xmlread(xmlpathma);

        scene=xDoc.getElementsByTagName('Scene'); % Get to the node "Scene"
        scene=scene.item(0);
        cell=scene.getElementsByTagName('CellDimensions'); % Get to the node
            "CellDimensions"
        cell=cell.item(0);
        cell.setAttribute('x',int2str(opt.Celldim));
        cell.setAttribute('y',int2str(opt.Celldim));

        dim=scene.getElementsByTagName('SceneDimensions'); % Get to the node
            "CellDimensions"
        dim=dim.item(0);
        dim.setAttribute('x',num2str(opt.Nx,'%2f'));
        dim.setAttribute('y',num2str(opt.Ny,'%2f'));

        xmlwrite(xmlpathma,xDoc);
        end

    end
end

```

Open all images:

This function opens the images created by Sequence.

Input:

- ➔ Structure Opt: This variable must contain the name of the scene, the number of pixels in the scene, and the number of iterations (opt.name, opt.N).

→ lcm: an indexed matrix with the different materials in their respective coordinates.

```

function [ IMG ] = OpenAllImages( opt,lcm )
% Create a 3D image from all the image simulations
%
% Input parameters :
% -name
% -N : size of the image (in pixels)
% -numberbands : number of spectral bands (=number of images)
% Version 2.0 open brf,toa,sensor images (from v1.2)

if nargin<2
    lcm=0; end

if ~isfield(opt,'simtype')
    opt.simtype='brf'; end

%Default parameters : Useful if function called outside of Black Box
if ~isfield(opt,'path')
    opt.dartpath='../..\DART';
opt.path=[opt.dartpath,'\user_data\simulations\',opt.name];
end

if ~isfield(opt,'N')
    opt.N=size(lcm); opt.Nx=opt.N(1); opt.Ny=opt.N(2);
else %En supposant que la scène est carrée.
    if ~isfield(opt,'Nx')||~isfield(opt,'Ny')
        opt.Nx=opt.N; opt.Ny=opt.N; end
    end
    %%
    pathsim=[opt.path,'\sequence'];

IMG=zeros(opt.Nx,opt.Ny,opt.spband.nb); %Memory Allocation

switch opt.simtype
    case 'brf'
        endpath='BRF\ITERX\IMAGES_DART';
    case 'sensor'
        endpath='SENSOR\IMAGES_DART';
    case 'toa'
        endpath='TOA\IMAGES_DART';
    end

for i=0:opt.spband.nb-1
    path=[pathsim,'\Wavelength_',int2str(i),'\output\BAND0\',endpath];
    imgpath=[path,'\ima01_VZ=000_0_VA=000_0.mp#'];
    image=fopen(imgpath);
    F=fread(image,[opt.Nx,opt.Ny], 'double');
    IMG(:,:,i+1)=F;
    fclose(image);
end

end

```

Annex B. Simulation procedure for importing images

BlackBox:

This function will be used when opening an image that already exists. This image will have already been created using directly DART. The scene will already have in the output file the images generated by Sequence (also using exclusively DART). Therefore the only task that this function has is to located the files and open the images.

Input:

- ➔ Structure opt: Like in the other version of BlackBox, the needed information will be the name of the scene, the number of pixel of the scene, the first wavelength values, the step and size of the wavelengths, the number of wavelengths in our simulation and the number of iterations (opt.name, opt.N, opt.spband.first, opt.spband.step, opt.spband.delt, opt.spband.nb, opt.nbiter).

```
function [ bands, IMG ] = BlackBox1(opt)
%BlackBox Main function when the image haa already been created in DART
%      Input elements
%      -opt : structure with all the necessary options :
%
% % Version 2.0

% %% Falcultative Arguments

% % Creation of the scene
opt.dartpath='..\..\..\DART'; %% PATH TO CHANGE IF MATLAB FILES ARE MOVING
opt.defpath='..\DARTdefaultsimulation'; %% PATH TO CHANGE IF FILES ARE
MOVING

opt.path=[opt.dartpath, '\user_data\simulations\',opt.name];

xmlpathpl=[opt.path, '\input\phase.xml'];
xDoc= xmlread(xmlpathpl);

phase=xDoc.getElementsByTagName('Phase'); %we need to add this parameter in order to
execute in DART
phase=phase.item(0);
dainpa=phase.getElementsByTagName('DartInputParameters');
dainpa=dainpa.item(0);
flutra=dainpa.getElementsByTagName('nodefluxtracking');
flutra=flutra.item(0);
flutra.setAttribute('numberOfIteration',int2str(opt.nbiter));

xmlwrite(xmlpathpl,xDoc);
%%%%%%%%%%%%%% import sensors

senimsim = xDoc.createElement('SensorImageSimulation');
phase.appendChild(senimsim);
senimsim.setAttribute('importMultipleSensors','0');

xmlwrite(xmlpathpl, xDoc);
```

```

% % Create an hyperspectral image

        Sequence( opt );
        IMG=OpenAllImages(opt);

% Array of Spectral Bands
        band = opt.spband.first;
        bands=zeros(1,opt.spband.nb);

        for i=1:opt.spband.nb
            bands(i) = band;
            band = band+opt.spband.step;
        end

        end

```

Annex C. Simulation of sensors

Sensor Call:

This function establishes some of the values needed in order to simulate the sensor, if they have not been set already. To be more precise, the function establishes the sensor that will be used and the dimensions of the cell that make the image. Once this is established, the program will start with the deformation of the image.

Input:

- ➔ Structure Opt: In this particular function the information need will be the name of both the simulation and the sensor used (opt.name, opt.sen.sensor). The size of the cell will be set by default if this value is not set, but it has to have been created (opt.Celldim).

```

function [ output_args ] = SensorCall( opt )
%SensorCall Call the Sensor Simulation scripts (Pansharpening)
% The parameters useful for the simulation will be contained in the
%     structure opt.sen
%
        cd('../\Simulator')
        % addpath('../\Simulator\Images')
        % addpath('../\Simulator\Outputs')
        % addpath('../\Simulator\Relative Spectral Responses')
        % addpath('../\Simulator\Signatures')

%% Parameters useful for the simulation

        %%% Type of Sensor

        sensor = opt.sen.sensor;
        % sensor = 'IKONOS';
        % sensor = 'QUICKBIRD';
        % sensor = 'WV2';
        % sensor = 'none';

```

```

        %%%% Type of Sensor

        if ~isfield(opt.sen, 'sensor')
            opt.sen.sensor='IKONOS'; end

            % sensor = 'IKONOS';
            % sensor = 'QUICKBIRD';
            % sensor = 'WV2';
            % sensor = 'none';

        %%%% Dimension of the simulated image

        if ~isfield(opt, 'Celldim')
            opt.Celldim=1; end

% Dim_Pixel_Simulated_Image = opt.Celldim; % in [meter] usually << 1 meter

        SimulatorPansharpening( opt )

            cd('..\CodeV2.0')
            end

```

Simulator Pansharpening:

This function deforms the image as the real sensor would (since the image created in DART would be that of an ideal sensor). In order to do so, some information about the sensor such as their spectral response, their spatial resolution and their MTF gain value at the Nyquist frequency must be uploaded into the program.

Input:

- ➔ Structure Opt: In this particular function the information needed is the name of both the simulation and the sensor used (opt.name, opt.sen.sensor). The size of the cell will be set by default if we don't set a value, but it has to have been created (opt.Celldim).

Output:

- ➔ This function generates the images that the specific sensor would produce.

```

function[output_args] = SimulatorPansharpening( opt )
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Sensor Choice %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Sensor to simulate
        sensor = opt.sen.sensor;
        % sensor = 'IKONOS';
        % sensor = 'QUICKBIRD';
        % sensor = 'WV2';
        % sensor = 'none';

        % Plot Relative Spectral Responses of the Sensor
        plotRelativeSpectralResponse = 1;

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Parameter Setting %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    % Simulated Image  
Dim_Pixel_Simulated_Image = opt.Celldim; % in [meter] usually << 1 meter
```

```
    if strcmp(sensor,'IKONOS')  
        % Load the IKONOS Relative Spectral Response  
load('Relative Spectral Responses/IKONOS_Spectral_Responses.mat'); %  
        The order is: Blue, Green, Red, NIR, PAN
```

```
    % Degradation and Resampling IKONOS sensor
```

```
        % Cutoff frequency PANchromatic channel  
        Spatial_Resolution_PAN = 1; % in [meter]  
PAN_fcutoff = Dim_Pixel_Simulated_Image/Spatial_Resolution_PAN;  
        PAN_fsampling = PAN_fcutoff;  
        PAN_GNyq = 0.17;  
    % Cutoff frequency MultiSpectral/HyperSpectral (MS/HS) channel  
        Spatial_Resolution_MS = 4; % in [meter]  
MS_fcutoff = Dim_Pixel_Simulated_Image/Spatial_Resolution_MS;  
        MS_fsampling = MS_fcutoff;  
MS_GNyq = [0.26,0.28,0.29,0.28]; % Band Order: B,G,R,NIR
```

```
    % % % IKONOS with bicubic interpolator to degrade  
        %     PAN_GNyq = [];  
        %     MS_GNyq = [];
```

```
        % Quantization  
L = 11; % Radiometric Resolution IKONOS
```

```
    elseif strcmp(sensor,'QUICKBIRD')
```

```
        % Load the QUICKBIRD Relative Spectral Response  
load('Relative Spectral Responses/QUICKBIRD_Spectral_Responses.mat'); %  
        The order is: Blue, Green, Red, NIR, PAN
```

```
    % Degradation and Resampling QUICKBIRD sensor
```

```
        % Cutoff frequency PANchromatic channel  
        Spatial_Resolution_PAN = 0.61; % in [meter] at nadir  
PAN_fcutoff = Dim_Pixel_Simulated_Image/Spatial_Resolution_PAN;  
        PAN_fsampling = PAN_fcutoff;  
        PAN_GNyq = 0.15;  
    % Cutoff frequency MultiSpectral/HyperSpectral (MS/HS) channel  
        Spatial_Resolution_MS = 2.44; % in [meter] at nadir  
MS_fcutoff = Dim_Pixel_Simulated_Image/Spatial_Resolution_MS;  
        MS_fsampling = MS_fcutoff;  
MS_GNyq = [0.34 0.32 0.30 0.22]; % Band Order: B,G,R,NIR
```

```
    % % % QUICKBIRD with bicubic interpolator to degrade  
        %     PAN_GNyq = [];  
        %     MS_GNyq = [];
```

```

                                % Quantization
L = 11; % Radiometric Resolution QUICKBIRD

                                elseif strcmp(sensor, 'WV2')

                                % Load the WV-2 Relative Spectral Response
load('Relative Spectral Responses/WV2_Spectral_Responses.mat');

                                % Degradation and Resampling WV-2 sensor

                                % Cutoff frequency PANchromatic channel
Spatial_Resolution_PAN = 0.46; % in [meter] at nadir
PAN_fcutoff = Dim_Pixel_Simulated_Image/Spatial_Resolution_PAN;
PAN_fsampling = PAN_fcutoff;
PAN_GNyq = 0.11;
                                % Cutoff frequency MultiSpectral/HyperSpectral (MS/HS) channel
Spatial_Resolution_MS = 1.8; % in [meter] at nadir
MS_fcutoff = Dim_Pixel_Simulated_Image/Spatial_Resolution_MS;
MS_fsampling = MS_fcutoff;
MS_GNyq = [0.35 .* ones(1,7), 0.27]; % Band Order: All bands, NIR2

                                % % % WV-2 with bicubic interpolator to degrade
                                %     PAN_GNyq = [];
                                %     MS_GNyq = [];

                                % Quantization
L = 11; % Radiometric Resolution WV-2

                                else
                                fprintf('Wrong Selection\n');
                                return;
                                end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Other Parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                                coreg_shift_x = 0;
                                coreg_shift_y = 0;
                                sigma_gauss_noise = 0;
                                sigma_mul_noise = 0;
                                bandwidth_mul_noise = 10;
                                magnitude_stripping = 100;
                                ratio_BD_N = 0;
                                prob_bad_pixel = 0;
                                spectral_offset = [];
                                smile_width = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                                if plotRelativeSpectralResponse == 1
figure, plot(wavelength_nm, Spectral_Responses_Matrix)
                                xlabel('Wavelength [nm]');
                                ylabel('Relative Spectral Response');
                                end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Building a Test Image %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

        cd('..\CodeV2.0')
        LabeledImage = OpenAllImages(opt);
        cd('..\Simulator')
        % img = size(LabeledImage);
        % LabeledImage = Build_Image_Scenario_1();
        % LabeledImage = Build_Image_Scenario_2();
    % LabeledImage = Build_Image_Scenario_3(300,Dim_Pixel_Simulated_Image);
        % LabeledImage = Build_Image_Scenario_4(64);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Simulating Sensors %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        I_GT =
Simulating_Sensor(LabeledImage,wavelength_nm,Spectral_Responses_Matrix(1:en
        d-
1,:),PAN_fcutoff,MS_GNyq,PAN_fsampling,L,coreg_shift_x,coreg_shift_y,sigma_
gauss_noise,sigma_mul_noise,bandwidth_mul_noise,magnitude_stripping,ratio_BD_
        N,prob_bad_pixel,spectral_offset,smile_width);

        I_MS =
Simulating_Sensor(LabeledImage,wavelength_nm,Spectral_Responses_Matrix(1:en
        d-
1,:),MS_fcutoff,MS_GNyq,MS_fsampling,L,coreg_shift_x,coreg_shift_y,sigma_ga
uss_noise,sigma_mul_noise,bandwidth_mul_noise,magnitude_stripping,ratio_BD_N
        ,prob_bad_pixel,spectral_offset,smile_width);

        I_PAN =
Simulating_Sensor(LabeledImage,wavelength_nm,Spectral_Responses_Matrix(end,
        :),PAN_fcutoff,PAN_GNyq,PAN_fsampling,L,coreg_shift_x,coreg_shift_y,sigma_g
auss_noise,sigma_mul_noise,bandwidth_mul_noise,magnitude_stripping,ratio_BD_
        N,prob_bad_pixel,spectral_offset,smile_width);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Print Output RGB %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Adjustments to Print
        I_Printable_GT = zeros(size(I_GT));
        I_Printable_MS = zeros(size(I_MS));
        for ii = 1 : size(I_GT,3)
I_Printable_GT(:, :, ii) = imadjust(im2double(I_GT(:, :, ii)));
I_Printable_MS(:, :, ii) = imadjust(im2double(I_MS(:, :, ii)));
        end
        I_Printable_PAN = imadjust(im2double(I_PAN));

        % Print GT, MS and PAN
        figure, imshow(I_Printable_GT(:, :, [3 2 1]))
        figure, imshow(I_Printable_MS(:, :, [3 2 1]))
        figure, imshow(I_Printable_PAN)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Save Simulated Result %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        save 'Outputs/Simulation.mat' I_PAN I_MS I_GT

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```


Simulating Sensor:

This function is used in the SimulatorPansharpener function. This is where the actual degradation of the image takes place.

Input:

- LabeledImage: the image that has to be modified, it can either be created in MATLAB or simply opened from the Sequence folder.
- wavelength_nm: the range of Relative Spectral Responses of the specified sensor.
- Spectral_Responses_Matrix: contains the relative spectral response of the sensor for each of wavelength.
- fcutoff: cut frequency of the sensor.
- GNyq: stores the values of the MTF gain at the Nyquist frequency.
- fsampling: stores the sampling frequency.
- L: number of quantification.
- coreg_shift_x, coreg_shift_y: values of both the spatial shift in x and in y direction.
- sigma_gauss_noise, sigma_mul_noise: values of both standard deviation of gaussian and multiplication noise.
- bandwidth_mul_noise: bandwidth of the multiplicative noise.
- magnitude_stripping: magnitude stripping.
- ratio_BD_N: ration between bad and normal pixels.
- prob_bad_pixel: probability of bad pixels.
- spectral_offset: spectral offset.
- smile_width: width of the spectral smile effect.

Output:

- This function generates images that seems as if captured by the sensor we specified.

```
function SimulatedImage =  
Simulating_Sensor(LabeledImage,wavelength_nm,Spectral_Responses_Matrix,fcut  
off,GNyq,fsampling,L,coreg_shift_x,coreg_shift_y,sigma_gauss_noise,sigma_mu  
l_noise,bandwidth_mul_noise,magnitude_stripping,ratio_BD_N,prob_bad_pixel,sp  
ectral_offset,smile_width)  
  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
% % % INPUT  
% LabeledImage is the original scenario labeled. In library there will be  
% the relation between the labels and the spectral signatures of the  
% materials.  
% wavelength_nm is the vector of the range of Relative Spectral Responses  
% of the sensor to simulate.  
% Spectral_Responses_Matrix is the matrix of the Relative Spectral  
% Responses of the sensor to simulate. For each band we have a row in the  
% matrix.
```

```

    % fcutoff is the cut frequency of the sensor optics.
    % GNyq is the vector or a single value of the MTF gain at the Nyquist
        % frequency.
    % fsampling is the vector or a single value of the sampling frequency in
        % the downsampling phase.
    % L represents the levels in the quatizator.
    % coreg_shift_x: Spatial shift in x direction
    % coreg_shift_y: Spatial shift in y direction
    % sigma_gauss_noise: St.dev. Gaussian noise
    % sigma_mul_noise: St.dev. multiplicative noise
    % bandwidth_mul_noise: Bandwidth multiplicative noise
    % magnitude_stripping: Magnitude stripping
    % ratio_BD_N: Ratio between Bad/Dead pixels and Normal ones
    % prob_bad_pixel: Probability of bad pixels w.r.t. dead ones
    % spectral_offset: Spectral offset
    % smile_width: Width of the spectral smile effect
    % % % OUTPUT

    % SimulatedImage is the LabeledImage seen by the simulated sensor.
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if isempty(GNyq)
            flag_degradation = 'none';
        else
            flag_degradation = 'MTF';
        end

        flag_library = 'none';
        % flag_library = 'USGS';

        %%% Input checks
        if numel(fsampling) == 1
            fsampling = repmat(fsampling,[1 size(Spectral_Responses_Matrix,1)]);
        end

        if numel(GNyq) == 1
            GNyq = repmat(GNyq,[1 size(Spectral_Responses_Matrix,1)]);
        end

        if isempty(spectral_offset) || isempty(smile_width)
            % Quicker but no smile effect
            SimulatedImage =
            Build_Cube(wavelength_nm,Spectral_Responses_Matrix,LabeledImage);
        else
            SimulatedImage =
            Build_Cube_Smile(wavelength_nm,Spectral_Responses_Matrix,LabeledImage,spect
            ral_offset,smile_width);
        end
        % end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Degradation & Downsampling Phase %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        SimulatedImage =
        DegradationAndResampling(SimulatedImage,flag_degradation,fcutoff,GNyq,fsamp
        ling);

        SimulatedImage =

```

```

    coregistration_shift (SimulatedImage, coreg_shift_x, coreg_shift_y);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Noise Step %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    SimulatedImage =
noises (SimulatedImage, sigma_gauss_noise, sigma_mul_noise, bandwidth_mul_noise
        );

    SimulatedImage =
striping (SimulatedImage, magnitude_striping, ratio_BD_N, prob_bad_pixel);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Quatization Step %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    SimulatedImage = QuantizationAndConversionUint16 (SimulatedImage, L);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end

```

Build Cube:

This function is used to resize the image. That way, the program can simulate sensors which have different MTF vector lengths (like in this case are PAN and MS sensors) without having to change any of the code.

```

function SimulatedImage =
Build_Cube (wavelength_nm, Spectral_Responses_Matrix, LabeledImage)

SimulatedImage = zeros (size (LabeledImage, 1), size (LabeledImage, 2),
    size (Spectral_Responses_Matrix));

    b= size (SimulatedImage) %we add the spectral responses of the chosen sensor
SimulatedImage = LabeledImage (:, :, 1:size (Spectral_Responses_Matrix));

end

```

Annex D. Others

Load All Spectres:

In this function, information from a given file is uploaded. This information will be the properties of certain materials. This file comes from a specific location from a specific computer. Because of this it must be changed when using another computer. It will generate a .txt file with the value of the MxN different points of the matrix as reflectance value for each wavelength value. That is, it will generate MxN reflectance signatures. Two more parameters will need to be added to much DART’s database files which are the direct and the diffuse

transmittance. In this example, we will set these two values to zero.

```

        clc
        clear all
        close all
        %%%%%in this case we will upload an 10x10 image with 224 wavelengths
load('C:\Users\Ester\MaterialesSimulacion\simulated_image.mat') %Spectres
        of different materials
        spectre=zeros(size(wavelengths),4);

        N=size(img,1)

        for i = 1:N

            tagname = ['layer', int2str(i)];
            spectre(:,1)= wavelengths;

            for k= 1:N
                spectre(:,2) = img(i,k,:);
                spectre(:,3) = 0.0;
                spectre(:,4) = 0.0;

                header='# Name:  ';
header=[header, tagname]; %each material will have the same type of text file but with a different
                name and reflectance values
                header=[header, '\r\n#Sample\r\n'];
header=[header, '\r\nwavelength;reflectance;direct_transmittance;diffuse_tra
                nsmittance\r\n'];
                ident = ['layers', int2str(i)];
                finalIdent = [ident, int2str(k)];
                nameFile = strcat(finalIdent, '.txt');
                finalNameFile = strcat('2D-LAM_Sample_', nameFile)
                txtfile=fopen(finalNameFile, 'w');
                fprintf(txtfile,header);

                fprintf(txtfile, '%f;%f;%f;%f\r\n', spectre. ');
                end
            end
        end

```

Annex E. Unmixing at a pixel level

Unmixing

With this code the different materials present in the scene and the resulting reflectance after the DART simulation will be imported.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% Pixel with types of snow scenario %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        clc
        clear all
        close all

```

```

        %signatures of each material
        load('ice.mat')
        spD = ice(:,2);
        load('loams.mat')
        spA = loams(:,2);
        load('brickr.mat')
        spC = brickr(:,2);
        load('asph4.mat')
        spB = asph4(:,2);
        %reflectance of scene
        load('A3C.mat')
        sA3C = A3C(:,2)*100 ;
load('ABCA.mat')
        sABCA = ABCA(:,2)*100 ;
        load('AABB.mat')
        sAB = AABB(:,2)*100 ;
        load('ABAD.mat')

sABAD = ABAD(:,2)*100 ; %%%%%%mitad B mitad D

figure(1)

plot(loams(:,1),spA, 'm' , 'LineWidth', 2)

    hold on

plot(asph4(:,1),spB, 'g' , 'LineWidth', 2)

hold on

    plot(brickr(:,1),spC, 'c' , 'LineWidth', 2)

hold on

    plot(ice(:,1),spD, 'r' , 'LineWidth', 2)

hold on

    plot(ABCA(:,1),sABCA, 'y' , 'LineWidth', 2)

        hold on

plot(AABB(:,1),sAB , 'r' , 'LineWidth', 2)

    legend('matA', 'matB', 'matC', 'matD', 'outcome2', 'outcome3')

        %%%%%only focus on the range they all have data

contD = 0;
for i = 1:size(ice(:,1))

    if ice(i,1) < max(asph4(:,1))
        contD = contD +1;
    else

        break;

end;

```

```

end
figure(2)
plot(ice(1:contD, 1),spD(1:contD), '\:r' , 'LineWidth', 2)

hold on

contA = 0;
for i = 1:size(loams(:,1))
if loams(i,1) < max(asph4(:,1))
contA = contA +1;
end
end

plot(loams(1:contA,1),spA(1:contA), '\:m' , 'LineWidth', 2)

hold on
contC = 0;

plot(asph4(:,1),spB, '\:g' , 'LineWidth', 2)

hold on

for i = 1:size(brickr(:,1))
if brickr(i,1) < max(asph4(:,1))
contC = contC +1;
end
end

plot(brickr(1:contC, 1),spC(1:contC), '\:c' , 'LineWidth', 2)

hold on
contM1 = 0;

for i = 1:size(A3C(:,1))
if A3C(i,1) < max(asph4(:,1))
contM1 = contM1 +1;
end
end

plot(ABCA(1:contM1,1),sABCA(1:contM1), '\:y' , 'LineWidth', 2)

hold on

contM2 = 0; for i = 1:size(AABB(:,1))
if AABB(i,1) < max(asph4(:,1))
contM2 = contM2 +1;
end
end

```

```

end
contM4 = 0; for i = 1:size(ABCA(:,1))

    if ABCA(i,1) < max(asph4(:,1))

contM4 = contM4 +1;

end

end

plot(ABCA(1:contM4,1),sABCA(1:contM4) , 'r', 'LineWidth', 2)

hold on

contM3 = 0;
for i = 1:size(ABAD(:,1))

    if ABAD(i,1) < max(asph4(:,1))

contM3 = contM3 +1;

end

end

plot(ABAD(1:contM3,1),sABAD(1:contM3) , 'c', 'LineWidth', 2)

legend('matD', 'matA', 'matB', 'matC')

%
% for i= 1:contA
% if (loams(i,1)== AABB(:,1))
%     i
% end
% end

A = [spA(51), spB(31), spC(76), spD(76);
spA(351), spB(181), spC(226), spD(226);
spA(451), spB(306), spC(261), spD(261);
spA(576), spB(488), spC(286), spD(286)];
cond(A)

%
%
S = [sAB(2), sAB(8), sAB(13), sAB(23)]
F = [sABCA(2), sABCA(8), sABCA(13), sABCA(23)]

G = [sABAD(2), sABAD(8), sABAD(13), sABAD(23)]

p3=A\F';
prob3 = p3./sum(p3)

F
A*[0.5 0.25 0.25 0]'
%
p4=A\G';
prob2 = p4./sum(p4)

```

G
A*[0.5 0.25 0.00 0.25]'

Range selection for sequence

It will be evaluated which four points generate a more precise unmixing.

```
A = zeros(4,4);
for f = 1:20 %% (120)/4 = 20

    i = f+52;
    g = f + 252;

    A = [sA(g),sB(i),sC(i), sD(i); %%I want to start off where the
wavelength is now repeated

        sA(g+20),sB(i+20),sC(i+56),sD(i+20);

        sA(g+2*20),sB(i+2*20),sC(i+2*20), sD(i+2*20);

        sA(g+3*20),sB(i+3*20),sC(i+3*20),sD(i+3*20)];

                if cond(A) < 280

t = cond(A)
wavel= [snow(g,1);snow(i+20,1); snow(i+2*20,1);snow(i+3*20,1)]
end

end
```

Range selection for unmixing

With this code it will be possible to determine the best range for DART to simulate. The most common step and where to start the sequence will be found.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% Pixel with types of tA scenario %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        clc
        clear all
        close all

%signatures of each material
        load('loams.mat')
        spDs = loams(:,2);
        wD = loams(:,1);
        load('brickr.mat')
        spAs = brickr(:,2);
        wA = brickr(:,1);
        load('ice.mat')
        spCs = ice(:,2);
        wC = ice(:,1);
        load('asph4.mat')
        spBs = asph4(:,2);
        wB = asph4(:,1);

% %reflectance of scene
```



```

        load('ABAD.mat')
        ABs = ABAD(:,2)*100 ; %%%%%%mitad B mitad
        wres = ABAD(:,1);

        maximo = [size(wA),size(wB), size(wC), size(wD)];
        maxi = [maximo(1), maximo(3), maximo(5), maximo(7)];%%maxi =
            maxi(1:2:size(maximo))
        wG = -1*ones(max(maxi)+1,1);

        %%%%% I locate the wavelengths found in all materials
        %%%%% after I locate the wavelengths that are also in the resulting spectra
        cont1 = 1;
        wBB = [wB; -3]; %%%add a negative number so I can compare j and j+1
        for i = 1: size(wA)
            for j = 1: size(wB)
                if wBB(j)~=wBB(j+1)
                    if wA(i) == wBB(j)
                        wG(cont1) = wA(i);
                        cont1 = cont1 +1;
                    break;%%once found the value we stop looking
                end
            end
        end
        end

        cont1 = 1;
        wCC = [wC; -3];

        for i = 1: size(wG)
            for j = 1: size(wC)
                if wCC(j)~=wCC(j+1)
                    if wG(i) == wCC(j)
                        wG(cont1) = wCC(j);
                        cont1 = cont1 +1;
                    break;
                end
            end
        end
        end

        cont1 = 1;
        wDD = [wD; -3];

        for i = 1: size(wG)
            for j = 1: size(wD)
                if wDD(j)~=wDD(j+1)
                    if wG(i) == wDD(j)
                        wG(cont1) = wG(i);

                        cont1 = cont1 +1;
                    break;
                end
            end
        end
        end

        %%%%%we have seen the wavelengths on the different materials, now the
        %%%%%resulting image
        totalwav = zeros(1,cont1-1);

```

```

contador = 1;   %%      so no number is repeated and get the wavelengths that
                  I need and fit the profile range
                for i = 1:cont1
                  if wG(i) < wG(i+1)
                    totalwav(contador) = wG(i);
                    contador = contador +1;
                else break; % once found we don't need to keep on searching
                end
            end
%%i divide into four sections so that the steps won't be too far off each
%%other
    totalwav = totalwav';
    div = size(totalwav(:,1))/4
    redondeo = round(div(1))
if redondeo > div(1)    %%si round - div > 0 -> high number
    twav = redondeo-1;
else twav = redondeo;

    end

    steps = ones(twav,1);
    steps1 = ones(twav,1);
    steps2 = ones(twav,1);
    steps3 = ones(twav,1);
    steps4 = ones(twav,1);

    for n = 1: size(totalwav)-1
        steps(n) = totalwav(n+1,1) - totalwav(n,1);
        end
        size(steps)
        for n = 1:twav-1
            steps1(n) = totalwav(n+1,1) - totalwav(n,1);
            end
        for n = twav: 2*twav-1
            steps2(n-twav+1) = totalwav(n+1,1) - totalwav(n,1);
            end
        for n = 2*twav: 3*twav-1
            steps3(n-2*twav+1) = totalwav(n+1,1) - totalwav(n,1);
            end
        for n = 3*twav: 4*twav-1
            steps4(n-3*twav+1) = totalwav(n+1,1) - totalwav(n,1);
            end

        rang1=find(steps== mode(steps1));
        repetidos1=length(rang1);
        rang2=find(steps== mode(steps2));
        repetidos2=length(rang2);
        rang3=find(steps== mode(steps3));
        repetidos3=length(rang3);
        rang4=find(steps== mode(steps4));
        repetidos4=length(rang4);
        rep = [repetidos1, repetidos2, repetidos3, repetidos4];
        valorRep = max(rep);    %% range where we find the most common step

        if valorRep == repetidos1
            stepsize = mode(steps1);
            firstWave = rang1(1);

```

```

        end
        if valorRep == repetidos2
            stepsize = mode(steps2);
            firstWave = twav +rang2(1);
        end
        if valorRep == repetidos3
            stepsize = mode(steps3);
            firstWave = 2*twav+rang3(1);
        end
        if valorRep == repetidos4
            stepsize = mode(steps4);
            firstWave = 3*twav+rang4(1);
        end
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        cont2 = 1;
        wT = -1*ones(size(wres));
        wresult = [wres; -3];

        for i = 1: size(totalwav)
            for j = 1: size(wres)
                if wresult(j)~=wresult(j+1)
                    if totalwav(i) == wresult(j)
                        wT(cont2) = wresult(j);
                        cont2 = cont2 + 1;
                        break;
                    end
                end
            end
        end

        %%%now we insert the reflectance to those wavelengths

        sA = zeros(cont2-1,1);    %%%with the resulting spectrum
        for i = 1:size(spAs)
            for j = 1:cont2-1    %cont2 = number of equal wavelengths in all the
                materials
                    if wA(i)==wT(j)
                        sA(j) = spAs(i);
                    end
                end
            end
        end
        sB = zeros(cont2-1,1);
        for i = 1:size(spBs)
            for j = 1:cont2-1
                if wB(i)==wT(j)
                    sB(j) = spBs(i);
                end
            end
        end
        sC = zeros(cont2-1,1);
        for i = 1:size(spCs)
            for j = 1:cont2-1
                if wC(i)==wT(j)
                    sC(j) = spCs(i);
                end
            end
        end
    end
end

```

```

        end
        end
        sD = zeros(cont2-1,1);
        for i = 1:size(spDs)
            for j = 1:cont2-1
                if wD(i)==wT(j)
                    sD(j) = spDs(i) ;
                end
            end
        end

        sRes = zeros(cont2-1,1);
        for i = 1:size(ABs)
            for j = 1:cont2-1
                if wres(i)==wT(j)
                    sRes(j) = ABs(i);
                    break;
                end
            end
        end
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% find the best values

        div = size(sA(:,1))/4
        redondeo = round(div)
        if redondeo(:,1) > div(:,1) %%si round - div > 0 -> high number
            tram = redondeo-1;

        else tram = redondeo;

        end

        A = zeros(4,4);
        for f = 1:tram %%int(43/4)
            A = [sA(f),sB(f),sC(f), sD(f); %%I want to start off where the
                wavelength is now repeated
                sA(2*f),sB(2*f),sC(2*f),sD(2*f);
                sA(3*f),sB(3*f),sC(3*f), sD(3*f);
                sA(4*f),sB(4*f),sC(4*f), sD(4*f)];
            if cond(A) < 7.5*10^4
                t = cond(A)
                wavel= [wT(f);wT(2*f); wT(3*f);wT(4*f)]
                Prueba = A;
                S = [sRes(f),sRes(2*f), sRes(3*f), sRes(4*f)]

                p4=A\S';
                prob4 = p4./sum(p4)
            end

        end

        advice = ['for best results try in sequence step = ', num2str(stepsize)];
        advice = [advice, ' and first wavelength = '];
        advice = [advice, num2str(wG(firstWave))]

```

Annex F. Scenarios

Scenario Snow Simple:

This function generates a specific scene and then generates the output results by calling on the different modules in DART. As output, this function will display the hyperspectral image of the scene. It will also display the reflectance value of the scene simulated and the reflectance value of the measurements of the real scene.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SCENARIO 1 : SNOW %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Snow
%
%
% clc
% clear all
% close all

%% INPUT OPTIONS

% Options
opt.name='TestV2\Sc1Snow';
opt.N=50; % Size of the map (in pixels)
opt.Celldim=1; %Size of a cell (in m)

opt.spband.first=0.4; %First spectral band of the sequence
opt.spband.step=0.05; %Interval between each spectral band
opt.spband.delt=0.05; %Width of the spectral bands
opt.spband.nb=20; %Number of Spectral Bands

opt.nbiter=5; % More precise for the snow

%% INPUT MATERIALS

% Default Options
matdef.name='None'; % Name of the material
matdef.index=0; % Index
matdef.plot=1; % Presence of plot (should stay at 1)
matdef.plottype=0; % Plot Type : Ground=0, Vegetation=1, G+V=2, Air=3
matdef.soiltype=0; % Soil Type : Lambertian=0, lamb+specular=2, Hapke=3
matdef.soildb='0';
matdef.soilsp='0';
matdef.vegdb='0';
matdef.vegsp='0';
matdef.LAI=0;
matdef.Hbase=0;
matdef.H=0;
matdef.Hsigma=0;
matdef.Tmean=300;
matdef.Tdelt=0;
matdef.clumpmin=0;
matdef.clumpmax=0;
matdef.cluma=0;
matdef.clumb=0;
matdef.LAD=0;
matdef.ALA=0;
matdef.eccen=0;
```

```

matdef.LeafSize=0;

% Real Mats
mat(1)=matdef;
mat(1).name='NeigeFine'; mat(1).index=1;
mat(1).soildb='Lambertian.db';mat(1).soilsp='snow_fine';

mat(2)=matdef;
mat(2).name='NeigeStandard'; mat(2).index=2;
mat(2).soildb='Lambertian.db';mat(2).soilsp='snow';

mat(3)=matdef;
mat(3).name='NeigeEpaisse'; mat(3).index=3;
mat(3).soildb='Lambertian.db';mat(3).soilsp='snow_coarse';

clear matdef;

opt.Nmat=size(mat,2); %Supposing mat is only an array

%% CREATION OF MAPS

% Land Cover Map
lcm = randi([1 opt.Nmat],opt.N,opt.N); %random
lcm = zeros(opt.N,opt.N);
for j=1:10; lcm(:,j)=1; end
for j=11:20; lcm(:,j)=3; end
for j=21:30; lcm(:,j)=2; end
for j=31:40; lcm(:,j)=1; end
for j=41:50; lcm(:,j)=3; end

figure
imagesc(lcm);
title(['Land Cover Map with ',int2str(opt.Nmat),' materials']);

% Topography
topo=zeros(opt.N,opt.N);

sigma1=100;
sigma2=7;
mpic1=opt.N/5;
mpic2=4*opt.N/5;
m=opt.N/2;
for i=1:opt.N
    for j=1:m
topo(i,j) =10*exp(-(((i-m)^2)/(2*sigma1^2))/2+(((j-
mpic1)^2)/(2*sigma2^2))));
        end
        for j=m:opt.N
topo(i,j) =10*exp(-(((i-m)^2)/(2*sigma1^2))/2+(((j-
mpic2)^2)/(2*sigma2^2))));
        end
    end
    %% for i=1:opt.N
    %% for j=1:opt.N
%% topo(i,j) = 0; %if we want a scene without topography
    %% end
    %% end

```

```

% clear m; clear sigma1; clear sigma2; clear i; clear j;
%
figure
imagesc(topo);
title('Topography of the scene')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% BLACK BOX LAUNCHER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% LAUNCH THE BLACK BOX

opt.atm = 1;
[bands, IMG]=BlackBox(opt,mat,lcm,topo);

%% SAVE OUTPUT FILES
% DisplayImages(IMG,opt,mat,lcm,bands)

% Display the Hyperspectral image
figure
imshow3D(IMG); % need to install imshow3D in the toolbox
title('Hyperspectral image')

%% Generate a 2D spectre
spx=floor(opt.N/2);
spy=floor(opt.N/2);
spectre=zeros(1,opt.spband.nb);

for i=1:opt.spband.nb
spectre(i)=IMG(spx,spy,i);
end

snow=importdata('../\DARTdefaultsimulation\2D-LAM_snow.txt',';');
figure
plot(snow.data(:,1),snow.data(:,2)/100,'--g');
hold on;
plot(bands,spectre);
xlim([bands(1) bands(size(bands,2))]);
xlabel('Wavelength (µm)');
ylabel('Reflectance');
legend('Original Snow spectre','Snow from the scene')
spin=lcm(spx,spy);
title(['Spectre of the central pixel of the scene, composed of
',mat(spin).name])
clear i spx spy ;

```

Snow Simple:

Different versions of a scene can be simulated in one go. In this case, this function will simulate the same scene with different types of topography. It can also be configured so that it does not take into account the effects of the atmosphere.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SCENARIO 1: SNOW %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                % Snow

                                clc
                                clear all
                                close all

                                %% INPUT OPTIONS

                                % Options
                                opt.name='TestV1\Sc1Snow';
                                opt.N=50; % Size of the map (in pixels)
                                opt.Celldim=1; %Size of a cell (in m)

                                opt.spband.first=0.4; %First spectral band of the sequence
                                opt.spband.step=0.05; %Interval between each spectral band
                                opt.spband.delt=0.05; %Width of the spectral bands
                                opt.spband.nb=20; %Number of Spectral Bands

                                opt.nbiter=5; % More precise for the snow

                                %% INPUT MATERIALS

                                % Default Options
                                matdef.name='None'; % Name of the material
                                matdef.index=0; % Index
                                matdef.plot=1; % Presence of plot (should stay at 1)
                                matdef.plottype=0; % Plot Type : Ground=0, Vegetation=1, G+V=2, Air=3
                                matdef.soiltype=0; % Soil Type : Lambertian=0, lamb+specular=2, Hapke=3
                                matdef.soildb='0';
                                matdef.soilsp='0';
                                matdef.vegdb='0';
                                matdef.vegsp='0';
                                matdef.LAI=0;
                                matdef.Hbase=0;
                                matdef.H=0;
                                matdef.Hsigma=0;
                                matdef.Tmean=300;
                                matdef.Tdelt=0;
                                matdef.clumpmin=0;
                                matdef.clumpmax=0;
                                matdef.cluma=0;
                                matdef.clumb=0;
                                matdef.LAD=0;
                                matdef.ALA=0;
                                matdef.eccen=0;
                                matdef.LeafSize=0;

                                % Real Mats
                                mat(1)=matdef;
                                mat(1).name='NeigeFine'; mat(1).index=1;
                                mat(1).soildb='Lambertian.db';mat(1).soilsp='snow_fine';

                                mat(2)=matdef;
                                mat(2).name='NeigeStandard'; mat(2).index=2;
                                mat(2).soildb='Lambertian.db';mat(2).soilsp='snow';

```



```

        mat(3)=matdef;
        mat(3).name='NeigeEpaisse'; mat(3).index=3;
mat(3).soildb='Lambertian.db';mat(3).soilsp='snow_coarse';

        clear matdef;

opt.Nmat=size(mat,2); %Supposing mat is only an array

        %% CREATION OF MAPS

        % Land Cover Map
% lcm = randi([1 opt.Nmat],opt.N,opt.N); %random
        lcm = zeros(opt.N,opt.N);
        for j=1:10; lcm(:,j)=1; end
        for j=11:20; lcm(:,j)=3; end
        for j=21:30; lcm(:,j)=2; end
        for j=31:40; lcm(:,j)=1; end
        for j=41:50; lcm(:,j)=3; end

        figure(1)
        imagesc(lcm);
title(['Land Cover Map with ',int2str(opt.Nmat),' materials']);

        % Topography
        topo=zeros(opt.N,opt.N);
        % topo2=zeros(opt.N,opt.N);
        % topo3=zeros(opt.N,opt.N);

        sigma1=100;
        sigma2=7;
        mpic1=opt.N/5;
        mpic2=4*opt.N/5;
        m=opt.N/2;
        for i=1:opt.N
            for j=1:m
                topo(i,j) =10*exp(-(((i-m)^2)/(2*sigma1^2))/2+(((j-mpic1)^2)/(2*sigma2^2)));
                topo2(i,j) =10*exp(-(((i-m)^2)/(2*sigma1^2))/2+(((j-mpic1)^2)/(2*sigma2^2)));
            end
            for j=m:opt.N
                topo(i,j) =10*exp(-(((i-m)^2)/(2*sigma1^2))/2+(((j-mpic2)^2)/(2*sigma2^2)));
                topo3(i,j) =10*exp(-(((i-m)^2)/(2*sigma1^2))/2+(((j-mpic2)^2)/(2*sigma2^2)));
            end
        end
clear m; clear sigma1; clear sigma2; clear i; clear j;

        figure(2)
        imagesc(topo);
        title('Topography of the scene')
        figure
        imagesc(topo2);
        title('Topography of the scene')
        figure
        imagesc(topo3);

```

```

        title('Topography of the scene')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% BLACK BOX LAUNCHER %%%%%%%%%%%%%%%

        %% LAUNCH THE BLACK BOX

                opt.atm=1;
        [bands1, IMG1]=BlackBox(opt,mat,lcm,topo);
        % [bands2, IMG2notopo]=BlackBox(opt,mat,lcm);

                %% With Atmosphere
                opt.atm=1;
        % [bands3, IMG3atm]=BlackBox(opt,mat,lcm,topo);

[bands4, IMG4top2]=BlackBox(opt,mat,lcm,topo2); % non-reflecting mountain
[band5, IMG5top3]=BlackBox(opt,mat,lcm,topo3); % only reflecting mountain

        %% SAVE OUTPUT FILES
        % DisplayImages(IMG,opt,mat,lcm,bands)

        %% Display the Hyperspectral image
        figure(3)
        imshow3D(IMG1); % need to install imshow3D in the toolbox
        title('Hyperspectral image')
        % figure
% imshow3D(IMG2notopo); % need to install imshow3D in the toolbox
% title('Hyperspectral image')
% figure
% imshow3D(IMG3atm); % need to install imshow3D in the toolbox
% title('Hyperspectral image')
        figure(4)
        imshow3D(IMG4top2); % need to install imshow3D in the toolbox
        title('Hyperspectral image')
        figure(5)
        imshow3D(IMG5top3); % need to install imshow3D in the toolbox
        title('Hyperspectral image')

        %% Generate a 2D spectre
        spx=floor(opt.N/2);
        spy=floor(opt.N/2);
        % spectre1=zeros(1,opt.spband.nb);
        % spectre2=zeros(1,opt.spband.nb);
        spectre3=zeros(1,opt.spband.nb);
        spectre4=zeros(1,opt.spband.nb);
        spectre5=zeros(1,opt.spband.nb);

        for i=1:opt.spband.nb
                spectre1(i)=IMG1(spx,spy,i);
        % spectre2(i)=IMG2notopo(spx,spy,i);
        % spectre3(i)=IMG3atm(spx,spy,i);
        spectre4(i)=IMG4top2(spx,spy,i);
        spectre5(i)=IMG5top3(spx,spy,i);
        end

        snow=importdata('..\DARTdefaultsimulation\2D-LAM_snow.txt',';');
        atm=importdata('..\DARTdefaultsimulation\gTRANS_USSTD76_gTRANS.txt',';');

```

```

        figure(6)
        plot(snow.data(:,1), snow.data(:,2)/100, '--g');
            hold on;
            plot(bands1, spectre1); % Topo + No atm
            plot(bands1, spectre2, 'r'); % No Topo + No atm
            plot(bands1, spectre3, 'm'); % Topo + atm
            hold off
            plot(bands1, spectre4, 'o k'); % Topo un coté + atm
            plot(bands1, spectre5, '--c'); % Topo autre coté + atm
            xlim([bands1(1) bands1(size(bands1,2))]);
            xlabel('Wavelength (µm)');
            ylabel('Reflectance');
legend('Original Snow spectre', 'Snow with topography and no atm', 'Snow
without topography nor atm', 'Snow with atmosphere', 'Snow with non-
reflecting mountain', 'Snow with reflecting mountain')
            spin=lcm(spx, spy);
            title(['Spectre of the central pixel of the scene, composed of
', mat(spin).name])
            clear i spx spy ;

%% Atmosphere
spx=floor(opt.N/2);
spy=floor(opt.N/2);

opt.simtype='brf';
IMGbrf=OpenAllImages(opt, lcm);
opt.simtype='sensor';
IMGsen=OpenAllImages(opt, lcm);
opt.simtype='toa';
IMGtoa=OpenAllImages(opt, lcm);

spectre1=zeros(1, opt.spband.nb);
spectre2=zeros(1, opt.spband.nb);
spectre3=zeros(1, opt.spband.nb);

for i=1:opt.spband.nb
    spectre1(i)=IMGbrf(spx, spy, i);
    spectre2(i)=IMGsen(spx, spy, i);
    spectre3(i)=IMGtoa(spx, spy, i);
end

bands1=bands3;
snow=importdata('..\DARTdefaultsimulation\2D-LAM_snow.txt', ';');
        figure(7)
        plot(snow.data(:,1), snow.data(:,2)/100, '--g');
            hold on;
            plot(bands1, spectre1); %BRF
            plot(bands1, spectre2, ' r'); %Sensor
            plot(bands1, spectre3, ' m'); %TOA
            xlim([bands1(1) bands1(size(bands1,2))]);
            xlabel('Wavelength (µm)');
            ylabel('Reflectance');
            legend('Original Snow spectre', 'BRF', 'Sensor', 'TOA')
            spin=lcm(spx, spy);
            title(['Spectre of the central pixel of the scene, composed of
', mat(spin).name])
            clear i spx spy ;

```

figure

```

imshow3D(IMGbrf); % need to install imshow3D in the toolbox
title('Hyperspectral image BRf')
figure
imshow3D(IMGsen); % need to install imshow3D in the toolbox
title('Hyperspectral image Sensor')
figure
imshow3D(IMGtoa); % need to install imshow3D in the toolbox
title('Hyperspectral image TOA')

```

Scenario Unmixing:

In scenario unmixing, MATLAB will make use of images already created in DART. The image will be regenerated and saved in MATLAB in order to later modify it. Before opening the images the file phase.xml must be modified since the function *Sequence* will be called on. That is so for the module sequence will be executed in order to generate the images and, in order to do so, the file wavelength must be created. After the images are loaded in MATLAB, the program will recreate the image as if it were been seen by a specific sensor (in this case there are three sensor to choose from: Ikonos, Quickbird and WV2).

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SCENARIO 2 : UNMIXING %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Unmixing : using different spatial resolution to
close all
clc

% Options
opt.name='Simulation_randomHouses'; %The simulation is already done
opt.sen.sensor='IKONOS';
opt.N = 40; %%%%we need it for Blackbox
opt.nbiter = 5; %%%for DART options

opt.spband.first=0.56; %First spectral band of the sequence
opt.spband.step=0.02; %Interval between each spectral band
opt.spband.delt=opt.spband.step; %Width of the spectral bands

opt.spband.nb=4; %Number of Spectral Ban

BlackBox1(opt);

SensorCall(opt)

```