

Topography of Map Selection in Google Earth Engine

Technical Documentation

TABLE OF CONTENTS

TECHNICAL DOCUMENTATION FOR THE FINAL VERSION (30.12.2021)...page 1-12

- ISSUE & OVERVIEW ON THE PROJECT
- TITLE
- TECHNICALITIES:

I. Functions of the map

II. Program overview

III. Details (includes code snippets, documentation and explanation for each point mentioned below)

A. CENTERING MAP (page 2)

B. ENABLE USER TO DRAW GEOMETRY (page 2-3)

C. BUILD CONTROL PANEL (page 4)

D. FUNCTION THAT MAKES PROGRAM RUN AUTOMATICALLY (page 4-10)

E. MAKE RESULTS VISIBLE ON DRAW/EDIT (page 10)

IV. Application details (creating and publishing application) (page 10-12)

DATE: 30.12.2021

ISSUE & OVERVIEW ON NEW PROJECT: The initial plan was too complex to implement – it is a research issue that has not been solved yet. Landslide prediction is a complex subject and predicting it requires a lot more parameters than I have included. I have included elevation, slopes, soil humidity and soil composition. From what I have learnt, elevation (which I initially thought would be the most important for landslide prediction) is not a crucial factor to landslides. In fact, geology contributes more to landslides. I have learnt that I do not have the necessary Geographical background for implementing this project. Since I want to learn how to work in Google Earth Engine, I changed the topic of the project, but kept the working environment. Also, the User Interface for the project remains unchanged – the user has the possibility to do a selection of the map. Only this time, the program will not show the user the landslide susceptibility map, but a topographic map.

The final version of the application can be found at the following link:
<https://francescadragut01.users.earthengine.app/view/topography-of-map-selection>

The source code of the application can be found at the following link:
<https://code.earthengine.google.com/76a66928b34128b4c99aed820eddbae5>

TITLE: Topography of Map Selection in Google Earth Engine

TECHNICALITIES:

I. Functions of the map:

- The functions already available in Google Earth Engine: zoom in and zoom out
- A User Interface function, that allows the user to select a rectangle on the map

II. Program overview (each subpoint – detailed at point C, below):

- Set map center on Romania (the program will work on any area on the globe, but this detail is for convenience)
- Create functions to enable the user to draw a geometry on the map
- Build the control panel (with the “Select Area” button)
- A function that makes the results visible after drawing, making the program run by itself after the user selects something on the map
- Make results visible on drawing/editing

III. Details:

A. CENTERING MAP:

- i. Documentation: <https://developers.google.com/earth-engine/apidocs/map-setcenter>
- ii. Code: `Map.setCenter(24.5, 45.5, 6.2);`

B. ENABLE USER TO DRAW GEOMETRY

i.Documentation: <https://developers.google.com/earth-engine/tutorials/community/drawing-tools-region-reduction>

ii.Code:

```
var drawingTools = Map.drawingTools();
drawingTools.setShown(false);

while (drawingTools.layers().length() > 0) {
  var layer = drawingTools.layers().get(0);
  drawingTools.layers().remove(layer);
}

var dummyGeometry =
  ui.Map.GeometryLayer({geometries: null, name: 'geometry', color: '23cba7'});

drawingTools.layers().add(dummyGeometry);
function clearGeometry() {
  var layers = drawingTools.layers();
  layers.get(0).geometries().remove(layers.get(0).geometries().get(0));
}
```

Figure 1. The preparatory steps in enabling the user to draw geometries.

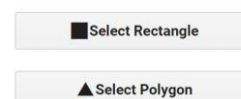
```
function drawRectangle() {
  clearGeometry();
  drawingTools.setShape('rectangle');
  drawingTools.draw();
}
function drawPolygon() {
  clearGeometry();
  drawingTools.setShape('polygon');
  drawingTools.draw();
}
```

Figure 2. The specific drawing functions for each particular shape.

iii. Explanation:

1. Define a variable, drawingTools, which is a widget object (Figure 1)
2. Make the default drawingTools (offered by GEE) invisible, so I can customize the selection (Figure 1)
3. Clear all previously defined geometries (Figure 1)
4. Create a dummyGeometry, that acts as a placeholder for drawn geometries (Figure 1)
5. Define a function that clears geometries on the map (later, it will be used to clear the geometry, in order for the topographical layer to be visible) (Figure 1)
6. Define 2 functions (Figure 2): drawRectangle() and drawPolygon() in order to enable the user to draw specific shapes. They are the underlying functions of the buttons “Select Rectangle” and “Select Polygon”, as seen in Figure 3.

Select shape of the selection:



Wait for results to render.

Figure 3. The control panel in the app. The button “Select Rectangle” enables the user to draw a rectangle on the map, while the button “Select Polygon” enables the user to draw a polygon on the map.

C. BUILD CONTROL PANEL

i. Documentation: <https://developers.google.com/earth-engine/tutorials/community/drawing-tools-region-reduction>

ii. Code:

```
/*C. Build the control panel*/

// 1. Define symbols for drawing shapes in the control panel
var symbol = {
  rectangle: '■',
  polygon: '▲',
  point: '●'
}

// 2. Create label for title of the app
var title = ui.Label({
  value: 'Topography of Map Selection',
  style: {width:'700px', height:'70px',fontSize:'30px',color: '484848',textAlign:'center'}
})
Map.add(title);

//3. Create the control panel
var controlPanel = ui.Panel({
  widgets: [
    ui.Label('Select shape of the selection:'),
    ui.Button({
      label: symbol.rectangle + 'Select Rectangle',
      onClick: drawRectangle,
      style: {stretch: 'horizontal'}
    }),
    ui.Button({
      label: symbol.polygon + 'Select Polygon',
      onClick: drawPolygon,
      style: {stretch: 'horizontal'}
    }),
    ui.Label('Wait for results to render.'),
  ],
  style: {position: 'bottom-left'},
  layout: null,
});

// 4. Add control panel to the app
Map.add(controlPanel);
```

Figure 4. Building the control panel.

iii. Explanation:

1. In Figure 4, the selection button has a black square in front of it, in order to visually guide the user that the button will allow him to select a rectangle, while the polygon has a triangle in front of it.
2. Create a widget for the title, with the ui.Label() function
3. Define a variable, controlPanel, which is created as a User Interface Panel. Being a UI Panel, it includes the following widgets: labels and buttons. In the “style” property, the panel is set to appear in the bottom-left corner.
4. Add control panel to the map.

D. THE FUNCTION THAT MAKES THE PROGRAM RUN AUTOMATICALLY.

Note: This section will be the longest one in the program, since this function contains all the features that makes the program functional. This function is called `resultsGeneration()` and it is made up of five main steps.

1. Turn the geometry selected by the user into an object

i. Documentation: <https://developers.google.com/earth-engine/tutorials/community/drawing-tools-region-reduction>

ii. Code:

```
/* 1. Turn the geometry selected by the user into an object to be handled by the program*/
// 1.1. Get the drawn geometry; it will define the reduction region.
var aoi = drawingTools.layers().get(0).getEeObject();

// 1.2. Set the drawing mode back to null; turns drawing off.
drawingTools.setShape(null);
```

Figure 5. Turn the geometry into an object.

iii. Explanation:

- 1.1. Declare variable “aoi” (area of interest), which spans over the geometry selected by the user (situated on layer 0 – `layers().get(0)`) and transforms it into an object, by using the function `getEeObject()`.
- 1.2. Set the drawing mode back to default, in case the user needs to draw something else.

2. Create the layers to build the topography of the area

i. Code:

```
/* 2. Create layers to build topography*/
/* 2.1. ALOS DSM: Global 30m */
var ALOS = ee.Image('JAXA/ALOS/AW3D30/V2_2').multiply(4);
/* 2.2. Traditional Hillshade (input, azimuth, altitude).multiply(weight) */
var N = ee.Terrain.hillshade(ALOS, 0, 36).multiply(0);
var NE = ee.Terrain.hillshade(ALOS, 45, 44).multiply(0);
var E = ee.Terrain.hillshade(ALOS, 90, 56).multiply(0);
var SE = ee.Terrain.hillshade(ALOS, 135, 68).multiply(0);
var S = ee.Terrain.hillshade(ALOS, 180, 80).multiply(0.1);
var SW = ee.Terrain.hillshade(ALOS, 225, 68).multiply(0.2);
var W = ee.Terrain.hillshade(ALOS, 270, 56).multiply(0.2);
var NW = ee.Terrain.hillshade(ALOS, 315, 44).multiply(0.5);
/* 2.3. Multidirectional Hillshade */
var MULTI = N
  .add(NE)
  .add(E)
  .add(SE)
  .add(S)
  .add(SW)
  .add(W)
  .add(NW)
  .visualize({
    min: 0,
    max: 255,
    palette: [
      '#000000',
      '#ffffff'
    ],
  })
  .resample('bicubic')
  .updateMask(0.5);
```

```

/* 2.4. Slope */
var SLOPE = ee.Terrain.slope(ALOS)
  .multiply(2)
  .visualize({
    min:100,
    max:180,
    palette:[
      'ffffff',
      '000000'
    ]
  })
  .resample('bicubic')
  .updateMask(1);
/* 2.5. Shaded Relief */
var SHADED_RELIEF = ee.ImageCollection([
  SLOPE,
  MULTI
])
  .mosaic()
  .reduce(
    ee.Reducer.median()
  )
  .updateMask(1);

/* 2.6. Elevation */
var ELEVATION = ALOS
  .visualize({
    bands:['AVE_DSM'],
    min:0,
    max:12500,
    palette:[
      '#386641',
      '#6a994e',
      '#a7c957',
      '#fdf7d6',
      'ffffff'
    ]
  })
  .resample('bicubic')
  .updateMask(0.4);
/* 2.7. Surface Water */
var SURFACE_WATER = GSWM
  .visualize({
    bands:['occurrence'],
    min:0,
    max:100,
    palette:[
      '#B9E9E7'
    ]
  })
  .resample('bicubic');

/* 2.8. Sea */
var SEA = ALOS
  .updateMask(ALOS.lte(0))
  .visualize({
    bands:['AVE_DSM'],
    min:0,
    max:0,
    palette:[
      'B9E9E7'
    ]
  })
  .resample('bicubic');
/* 2.9. Bathymetry */
var BATHYMETRY = NOAA
  .updateMask(NOAA.lte(-10))
  .visualize({
    bands:['bedrock'],
    min:-5000,
    max:0,
    palette:[
      '#8ECCCB',
      '#ABE0DF',
      'B9E9E7'
    ]
  })
  .resample('bicubic');

```

ii. Explanation:

2.1.Import Image “JAXA/ALOS/AW3D30/V2_2”.

- 2.2. Set specific hillshade for every cardinal point, in order to create the contrast and shade effect on the topographical map.
- 2.3. Create a multidirectional hillshade, which connects and blends the hillshades for the cardinal points, making a layer that depicts a uniform hillshade for the map.
- 2.4. Create the layer for slope, which works on the ALOS image, imported in section 2.1. It applies terrain properties and visualization features (min, max, palette, resample, etc)
- 2.5. Create the layer for shaded relief. This layer works on the slope layer created previously and on the multishade layer. Basically, this layer applies the contrast of the hillshades to the altitude differences obtained in the slope layer
- 2.6. Create the elevation layer from the band “AVE_DSM”, using colors for elevation level differentiation
- 2.7. Create layer for surface water (lakes and rivers mainly). This layer uses the ‘occurrence’ band, which contains information about places which are covered with surface water
- 2.8. Create the layer for the sea sections
- 2.9. Create the bathymetry layer, which will differentiate in the shades of sea nuances, based on the depth of the sea water from the selection (if case)

3. Compute the mean, maximum and minimum elevation for the area of interest

i. Code:

```

/* 3. Compute mean, max and min elevations for the results panel*/
/* 3.1. Create dictionaries of image reduction*/
var dict = image.reduceRegion({
  reducer: "mean",
  geometry: aoi,
  scale: 900
});
var dictt = image.reduceRegion({
  reducer: "max",
  geometry: aoi,
  scale: 900
});
var dicttt = image.reduceRegion({
  reducer: "min",
  geometry: aoi,
  scale: 900
});

/* 3.2. Create UI labels in which to store each information obtained in 3.1.*/
var mean = ui.Label();
mean.setValue('Mean elevation: '+JSON.stringify(dict.get('elevation')).getInfo());
var max = ui.Label();
max.setValue('Maximum elevation: '+JSON.stringify(dictt.get('elevation')).getInfo())+'m');
var min = ui.Label();
min.setValue('Minimum elevation: '+JSON.stringify(dicttt.get('elevation')).getInfo())+'m');

```



```

/* 3.3. Compute the highest point, its coordinates define highest point on map */
var im = image.clip(aoi);
var pixel_pos = ee.Image.pixelLonLat();
var high = im.reduceRegion(ee.Reducer.max(), aoi, 900).get('elevation');
var highest = im.eq(ee.Image.constant(high));
var highest_point_image = im.addBands(pixel_pos).mask(highest);
var highest_point = highest_point_image.reduceRegion(ee.Reducer.mean(), aoi, 900);
var x_max = highest_point.get('longitude');
var y_max = highest_point.get('latitude');
var pt_max = ee.Geometry.Point([x_max,y_max]);

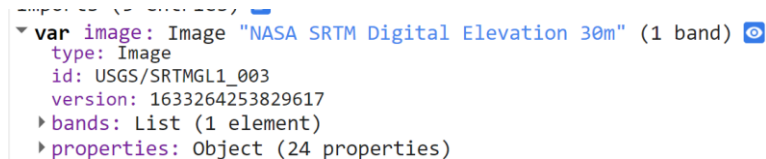
/* 3.4. Compute the lowest point, its coordinates define lowest point on map */
var pixel_pos_min = ee.Image.pixelLonLat();
var low = im.reduceRegion(ee.Reducer.min(), aoi, 900).get('elevation');
var lowest = im.eq(ee.Image.constant(low));
var lowest_point_image = im.addBands(pixel_pos).mask(lowest);
var lowest_point = lowest_point_image.reduceRegion(ee.Reducer.mean(), aoi, 900);
var x_min = lowest_point.get('longitude');
var y_min = lowest_point.get('latitude');
var pt_min = ee.Geometry.Point([x_min,y_min]);

```

ii. Explanation:

3.1. Create dictionaries for image reduction. First dictionary is for mean value, the second is for the max value and the third is for the min value. All of the dictionaries are based on the same idea: they reduce the image (see details about the image in Figure 6) to the area of interest, based on a reducer criterion: mean/max/min and on a scale, which determines the precision and the capabilities of selection.

3.2. For each of the obtained values, create a UILabel. For each label, set the value for the respective data. Later, these labels will be added in the results panel.



```

var image: Image "NASA SRTM Digital Elevation 30m" (1 band) ⓘ
  type: Image
  id: USGS/SRTMGL1_003
  version: 1633264253829617
  ▶ bands: List (1 element)
  ▶ properties: Object (24 properties)

```

Figure 6. Declaration of 'image', which is a SRTM Image, with elevation data.

3.3. Compute the highest point in the selection. Start by clipping the image to the area of interest and store it in variable 'im'. Then, reduce the image with the ee.Reduce.max() function. Out of the reduction, get the elevation data. Then, create the graphics for the point and declare it as a geometry of type point in 'pt_max'.

3.4. Compute the lowest point in the selection, similar to 3.3.

4. Add layers created in section 3 to the map.

i. Code

```

/* 4. Add layers to create topography*/
Map.addLayer(
  SHADED_RELIEF.clip(aoi),{
    min:0,
    max:255,
    gamma:1
  },
  'Shaded Relief',
  true
);

Map.addLayer(
  ELEVATION.clip(aoi),{},
  'Elevation',
  true
);

Map.addLayer(
  SURFACE_WATER.clip(aoi),{},
  'Surface Water',
  true
);

Map.addLayer(
  SEA.clip(aoi),{},
  'Sea',
  true
);

Map.addLayer(
  BATHYMETRY.clip(aoi),{},
  'Bathymetry',
  true
);

/*4.1. Add the layers corresponding to the lowest and highest points*/
Map.addLayer(pt_max, {color:'red'}, 'Highest point');
Map.addLayer(pt_min, {color:'green'}, 'Lowest point');

```

ii. Explanation:

Add each layer to the map. In order for topography to be visible, we need to see all the layers overlapped. We use the function `Map.addLayer()`. Clip each layer to the area of interest, using `layer_name.clip(aoi)` and set true for the visibility of each layer. In 4.1, the layers for the maximum and the minimum points, were also added to the map.

5. Create the results panel and display it.

i. Code:

```

/* 5. Create the results panel, clear geometry and add results panel to map*/
/* 5.1 Create the results panel*/
var resultsPanel = ui.Panel({
  widgets: [
    ui.Label('ELEVATION DATA'),
    mean,
    max,
    min,
  ],
  style: {position: 'bottom-right'},
  layout: null,
});
/* 5.2. Clear the geometry from the map, in order to make topography visible*/
clearGeometry();
/* 5.3. Add results panel to the map*/
Map.add(resultsPanel);

```

ii. Explanation:

5.1.Create the results panel as a UI Panel object. Set a label widget, which will serve as a title to the panel and add the mean, max and min widgets created in section 3. Set the panel to be visible in the bottom-right part of the map.

5.2.Clear the drawn geometry from the map. If it wasn't cleared, it would cover the topography results.

5.3.Display the results panel on the map.

E. MAKE RESULTS VISIBLE ON DRAW/EDIT

i. Code:

```
/* 5. Make results visible on drawing/editing*/  
drawingTools.onDraw(ui.util.debounce(resultsGeneration, 500));  
drawingTools.onEdit(ui.util.debounce(resultsGeneration, 500));
```

ii. Explanation:

Use function `ui.util.debounce()` in order to run the `resultsGeneration()` function, with a delay of 500 milliseconds after the user draws the geometry on the map. Run the function also when the user edits something on the map.

IV. Application details

After finishing the program, the application needs to be published, in order to be used. This can be done from the Google Earth Engine code editor. Press the button „Apps” (highlighted in Figure 7). The program will show a window, „Manage Apps” (see figure 8). From here, there are more possibilities. One possibility is to create a new app (see Figure 9) and another one is to update an existing app (see Figures 10 and 11).

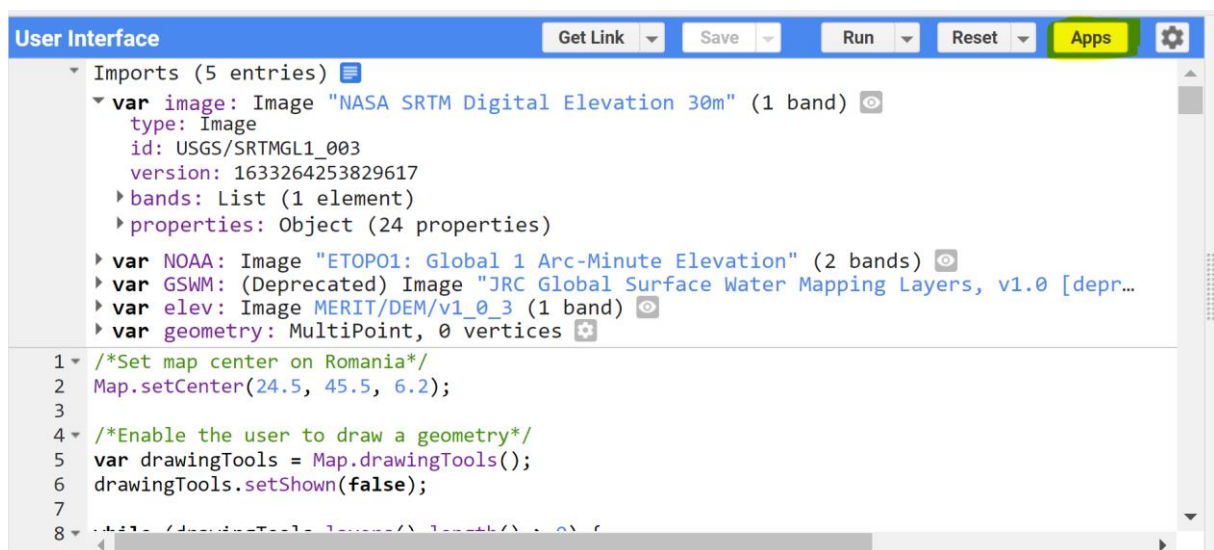


Figure 7. Highlighted „Apps” button in the GEE Code Editor.

App Name (click to launch)	ID (click to update app)	Delete
Beta Version	users/francescadragut01/beta-version	
Landslide Susceptibility	users/francescadragut01/landslide-susceptibility	
Topo	users/francescadragut01/topo	
var2	users/francescadragut01/var2	

Figure 8. Manage Apps. Can create a new app (top-right button „New app”) or can update existing app.

Publish New App

App Name

Beta Version #2

Your App ID will be beta-version-2 [Edit](#)

URL: <https://francescadragut01.users.earthengine.app/view/beta-version-2>

Google Cloud Project

ee-francescadragut01 [CHANGE](#)

Source Code

☒ Current contents of editor

☐ Repository script path

☒ **Gallery** ☐ Restriction ☐ Logo

☐ Feature this app in your [Public Apps Gallery](#)

Set Thumbnail (Optional) Description (Optional)

When the app is published, **it's public and anyone can view it**. The published source code will be publicly readable. All assets must also be shared publicly or with the app to display properly. See <https://developers.google.com/earth-engine/apps> for more information about publishing apps.

CANCEL **PUBLISH**

Figure 9. Publishing new app. Give title to app in section „App Name”. Next, press „Publish”.

Manage Apps			VIEW GALLERY	NEW APP
App Name (click to launch)	ID (click to update)	Update app	Delete	
Beta Version	users/francescadragut01/beta-version			
Landslide Susceptibility	users/francescadragut01/landslide-susceptibility			
Topo	users/francescadragut01/topo			
var2	users/francescadragut01/var2			

Figure 10. Editing existing app. Hover over the existing project and click on the link.

App Details

✓ users/francescadragut01/beta-version

App Name ⓘ
Beta Version

URL: <https://francescadragut01.users.earthengine.app/view/beta-version>

Google Cloud Project ⓘ
ee-francescadragut01 [CHANGE](#)

Source Code ⓘ
☐ Existing app source code
☒ Current contents of editor
☐ Repository script path

☒ Feature this app in your [Public Apps Gallery](#)

When the app is published, **it's public and anyone can view it**. The published source code will be publicly readable. All assets must also be shared publicly or with the app to display properly. See <https://developers.google.com/earth-engine/apps> for more information about publishing apps.

[CANCEL](#) [SAVE](#)

Figure 11. Editing existing app. Make sure to check the box „Current contents of editor” (in red box), in order to update the app with the script that is in the code editor.