

var beforeStart = '2019-02-16'

var beforeEnd = '2019-03-16'

var afterStart = '2019-03-17'

var afterEnd = '2019-03-25'

var ernakulam = admin2.filter(ee.Filter.eq('ADM2\_NAME', 'Gorgan'))

var geometry = ernakulam.geometry()

Map.addLayer(geometry, {color: 'grey'}, 'Gorgan District')

var collection= ee.ImageCollection('COPERNICUS/S1\_GRD')

.filter(ee.Filter.eq('instrumentMode','IW'))

.filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH'))

.filter(ee.Filter.eq('orbitProperties\_pass', 'DESCENDING'))

.filter(ee.Filter.eq('resolution\_meters',10))

.filterBounds(geometry)

.select('VH');

var beforeCollection = collection.filterDate(beforeStart, beforeEnd)

var afterCollection = collection.filterDate(afterStart,afterEnd)

var before = beforeCollection.mosaic().clip(geometry);

var after = afterCollection.mosaic().clip(geometry);

Map.addLayer(before, {min:-25,max:0}, 'Before Floods', false);

Map.addLayer(after, {min:-25,max:0}, 'After Floods', false);

var beforeFiltered = ee.Image(toDB(RefinedLee(toNatural(before))))

var afterFiltered = ee.Image(toDB(RefinedLee(toNatural(after))))

Map.addLayer(beforeFiltered, {min:-25,max:0}, 'Before Filtered', false);

Map.addLayer(afterFiltered, {min:-25,max:0}, 'After Filtered', false);

var difference = afterFiltered.divide(beforeFiltered);

// Define a threshold

var diffThreshold = 1.25;

// Initial estimate of flooded pixels

var flooded = difference.gt(diffThreshold).rename('water').selfMask();

Map.addLayer(flooded, {min:0, max:1, palette: ['orange']}, 'Initial Flood Area', false);

// Mask out area with permanent/semi-permanent water

var permanentWater = gsw.select('seasonality').gte(5).clip(geometry)

var flooded = flooded.where(permanentWater, 0).selfMask()

Map.addLayer(permanentWater.selfMask(), {min:0, max:1, palette: ['blue']}, 'Permanent Water')

// Mask out areas with more than 6 percent slope using the HydroSHEDS DEM

var slopeThreshold = 6;

var terrain = ee.Algorithms.Terrain(hydrosheds);

var slope = terrain.select('slope');

var flooded = flooded.updateMask(slope.lt(slopeThreshold));

Map.addLayer(slope.gte(slopeThreshold).selfMask(), {min:0, max:1, palette: ['cyan']}, 'Steep Areas', false)

// Remove isolated pixels

// connectedPixelCount is Zoom dependent, so visual result will vary

var connectedPixelThreshold = 8;

var connections = flooded.connectedPixelCount(25)

var flooded = flooded.updateMask(connections.gt(connectedPixelThreshold))

Map.addLayer(connections.lte(connectedPixelThreshold).selfMask(), {min:0, max:1, palette: ['yellow']}, 'Disconnected Areas', false)

Map.addLayer(flooded, {min:0, max:1, palette: ['red']}, 'Flooded Areas');

// Calculate Affected Area

print('Total District Area (Ha)', geometry.area().divide(10000))

var stats = flooded.multiply(ee.Image.pixelArea()).reduceRegion({

reducer: ee.Reducer.sum(),

geometry: geometry,

scale: 30,

maxPixels: 1e10,

tileScale: 16

})

print('Flooded Area (Ha)', ee.Number(stats.get('water')).divide(10000))

// If the above computation times out, you can export it

var flooded\_area = ee.Number(stats.get('water')).divide(10000);

var feature = ee.Feature(null, {'flooded\_area': flooded\_area})

var fc = ee.FeatureCollection([feature])

Export.table.toDrive({

collection: fc,

description: 'Flooded\_Area\_Export',

folder: 'earthengine',

fileNamePrefix: 'flooded\_area',

fileFormat: 'CSV'})

//############################

// Speckle Filtering Functions

//############################

// Function to convert from dB

function toNatural(img) {

return ee.Image(10.0).pow(img.select(0).divide(10.0));

}

//Function to convert to dB

function toDB(img) {

return ee.Image(img).log10().multiply(10.0);

}

//Apllying a Refined Lee Speckle filter as coded in the SNAP 3.0 S1TBX:

//https://github.com/senbox-org/s1tbx/blob/master/s1tbx-op-sar-processing/src/main/java/org/esa/s1tbx/sar/gpf/filtering/SpeckleFilters/RefinedLee.java

//Adapted by Guido Lemoine

// by Guido Lemoine

function RefinedLee(img) {

// img must be in natural units, i.e. not in dB!

// Set up 3x3 kernels

var weights3 = ee.List.repeat(ee.List.repeat(1,3),3);

var kernel3 = ee.Kernel.fixed(3,3, weights3, 1, 1, false);

var mean3 = img.reduceNeighborhood(ee.Reducer.mean(), kernel3);

var variance3 = img.reduceNeighborhood(ee.Reducer.variance(), kernel3);

// Use a sample of the 3x3 windows inside a 7x7 windows to determine gradients and directions

var sample\_weights = ee.List([[0,0,0,0,0,0,0], [0,1,0,1,0,1,0],[0,0,0,0,0,0,0], [0,1,0,1,0,1,0], [0,0,0,0,0,0,0], [0,1,0,1,0,1,0],[0,0,0,0,0,0,0]]);

var sample\_kernel = ee.Kernel.fixed(7,7, sample\_weights, 3,3, false);

// Calculate mean and variance for the sampled windows and store as 9 bands

var sample\_mean = mean3.neighborhoodToBands(sample\_kernel);

var sample\_var = variance3.neighborhoodToBands(sample\_kernel);

// Determine the 4 gradients for the sampled windows

var gradients = sample\_mean.select(1).subtract(sample\_mean.select(7)).abs();

gradients = gradients.addBands(sample\_mean.select(6).subtract(sample\_mean.select(2)).abs());

gradients = gradients.addBands(sample\_mean.select(3).subtract(sample\_mean.select(5)).abs());

gradients = gradients.addBands(sample\_mean.select(0).subtract(sample\_mean.select(8)).abs());

// And find the maximum gradient amongst gradient bands

var max\_gradient = gradients.reduce(ee.Reducer.max());

// Create a mask for band pixels that are the maximum gradient

var gradmask = gradients.eq(max\_gradient);

// duplicate gradmask bands: each gradient represents 2 directions

gradmask = gradmask.addBands(gradmask);

// Determine the 8 directions

var directions = sample\_mean.select(1).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(7))).multiply(1);

directions = directions.addBands(sample\_mean.select(6).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(2))).multiply(2));

directions = directions.addBands(sample\_mean.select(3).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(5))).multiply(3));

directions = directions.addBands(sample\_mean.select(0).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(8))).multiply(4));

// The next 4 are the not() of the previous 4

directions = directions.addBands(directions.select(0).not().multiply(5));

directions = directions.addBands(directions.select(1).not().multiply(6));

directions = directions.addBands(directions.select(2).not().multiply(7));

directions = directions.addBands(directions.select(3).not().multiply(8));

// Mask all values that are not 1-8

directions = directions.updateMask(gradmask);

// "collapse" the stack into a singe band image (due to masking, each pixel has just one value (1-8) in it's directional band, and is otherwise masked)

directions = directions.reduce(ee.Reducer.sum());

//var pal = ['ffffff','ff0000','ffff00', '00ff00', '00ffff', '0000ff', 'ff00ff', '000000'];

//Map.addLayer(directions.reduce(ee.Reducer.sum()), {min:1, max:8, palette: pal}, 'Directions', false);

var sample\_stats = sample\_var.divide(sample\_mean.multiply(sample\_mean));

// Calculate localNoiseVariance

var sigmaV = sample\_stats.toArray().arraySort().arraySlice(0,0,5).arrayReduce(ee.Reducer.mean(), [0]);

// Set up the 7\*7 kernels for directional statistics

var rect\_weights = ee.List.repeat(ee.List.repeat(0,7),3).cat(ee.List.repeat(ee.List.repeat(1,7),4));

var diag\_weights = ee.List([[1,0,0,0,0,0,0], [1,1,0,0,0,0,0], [1,1,1,0,0,0,0],

[1,1,1,1,0,0,0], [1,1,1,1,1,0,0], [1,1,1,1,1,1,0], [1,1,1,1,1,1,1]]);

var rect\_kernel = ee.Kernel.fixed(7,7, rect\_weights, 3, 3, false);

var diag\_kernel = ee.Kernel.fixed(7,7, diag\_weights, 3, 3, false);

// Create stacks for mean and variance using the original kernels. Mask with relevant direction.

var dir\_mean = img.reduceNeighborhood(ee.Reducer.mean(), rect\_kernel).updateMask(directions.eq(1));

var dir\_var = img.reduceNeighborhood(ee.Reducer.variance(), rect\_kernel).updateMask(directions.eq(1));

dir\_mean = dir\_mean.addBands(img.reduceNeighborhood(ee.Reducer.mean(), diag\_kernel).updateMask(directions.eq(2)));

dir\_var = dir\_var.addBands(img.reduceNeighborhood(ee.Reducer.variance(), diag\_kernel).updateMask(directions.eq(2)));

// and add the bands for rotated kernels

for (var i=1; i<4; i++) {

dir\_mean = dir\_mean.addBands(img.reduceNeighborhood(ee.Reducer.mean(), rect\_kernel.rotate(i)).updateMask(directions.eq(2\*i+1)));

dir\_var = dir\_var.addBands(img.reduceNeighborhood(ee.Reducer.variance(), rect\_kernel.rotate(i)).updateMask(directions.eq(2\*i+1)));

dir\_mean = dir\_mean.addBands(img.reduceNeighborhood(ee.Reducer.mean(), diag\_kernel.rotate(i)).updateMask(directions.eq(2\*i+2)));

dir\_var = dir\_var.addBands(img.reduceNeighborhood(ee.Reducer.variance(), diag\_kernel.rotate(i)).updateMask(directions.eq(2\*i+2)));

}

// "collapse" the stack into a single band image (due to masking, each pixel has just one value in it's directional band, and is otherwise masked)

dir\_mean = dir\_mean.reduce(ee.Reducer.sum());

dir\_var = dir\_var.reduce(ee.Reducer.sum());

// A finally generate the filtered value

var varX = dir\_var.subtract(dir\_mean.multiply(dir\_mean).multiply(sigmaV)).divide(sigmaV.add(1.0));

var b = varX.divide(dir\_var);

var result = dir\_mean.add(b.multiply(img.subtract(dir\_mean)));

return(result.arrayFlatten([['sum']]));

}