Automatic Color Segmentation Tool With ARCPro

Anthony K. Sonsteng

University of North Georgia

Abstract

Color segmentation is the division of an images pixels based on color. Automatic color segmentation has a number of applications across many different fields of work. from quality control to medical applications (Hance, 1996), traffic imaging (Dhanachandra, 2015), and Facial recognition . The application of color segmentation for this study is primarily land use. Being able to quickly and effectively isolate bodies of water before, during, and after a flood could help with rescue efforts. Being able to identify the areas affected by forest fires in a pinch can help curve further loss of green space. The ability to identify green space or impervious surfaces within a given area can help engineers make decisions on when, where, and what to construct.

Background

Easily accessible color segmentation is very handy. Having a tool such as this on hand should help a person to easily extract similar aspects within an area of interest such as green space or bodies of water. Being able to isolate these areas in a raster file makes processes such as extract by mask much more manageable and efficient. This is a tool I wish I had the means to manage and utilize several years ago in some of my lower level classes while in IESA.

Objectives

Isolate specific color ranges within a raster in order to make specific areas of interest more accessible for analysis

Materials & Methods

In order to produce the automatic color segmentation tool created here the following resources were used and utilized:

- ARCPro 2.1
- Notepad++

This is the python code implemented:

```
import numpy as np
import math
import statistics

class Toolbox(object):
    def __init__(self):
        """Define the toolbox (the name of the toolbox is the name of the
        .pyt file)."""
        self.label = "Toolbox"
        self.alias = ""

        # List of tool classes associated with this toolbox
        self.tools = [Automatic_greenspace_detector]


class Automatic_greenspace_detector(object):
    def __init__(self):
        """Define the tool (tool name is the name of the class)."""
        self.label = "Automatic Greenspace detector"
        self.description = ""
        self.canRunInBackground = False

    def getParameterInfo(self):
```

```
"""Define parameter definitions"""
input_raster = arcpy.Parameter(
    name="facc",
    displayName="input_raster",
    direction="Input",
    datatype="GPRasterLayer",
    parameterType="Required",
)
R_mean = arcpy.Parameter(
    name="R_mean",
    displayName="R_mean",
    direction="Input",
    datatype="GPDouble",
    parameterType="Required",
)
G_mean = arcpy.Parameter(
    name="G_mean",
    displayName="G_mean",
    direction="Input",
    datatype="GPDouble",
    parameterType="Required",
)
B_mean = arcpy.Parameter(
    name="B_mean",
    displayName="B_mean",
    direction="Input",
    datatype="GPDouble",
    parameterType="Required",
)
R_stdev = arcpy.Parameter(
    name="R_stdev",
    displayName="R_stdev",
    direction="Input",
    datatype="GPDouble",
    parameterType="Required",
)
G_stdev = arcpy.Parameter(
    name="G_stdev",
    displayName="G_stdev",
    direction="Input",
    datatype="GPDouble",
    parameterType="Required",
)
B_stdev = arcpy.Parameter(
    name="B_stdev",
    displayName="B_stdev",
            direction="Input",
            datatype="GPDouble",
            parameterType="Required",
        )
        K = arcpy.Parameter(
            name="K",
            displayName="K",
            direction="Input",
            datatype="GPDouble",
            parameterType="Required",
        )
        output_filepath_1 = arcpy.Parameter(
            name="output_filepath_1",
            displayName="Output_filepath_1",
            direction="Output",
            datatype="DERasterDataset",
            parameterType="Required",
        )
        output_filepath_2 = arcpy.Parameter(
            name="output_filepath_2",
            displayName="Output_filepath_2",
            direction="Output",
            datatype="DERasterDataset",
            parameterType="Required",
        )

        params = [input_raster, R_mean, G_mean, B_mean, R_stdev, G_stdev, B_stdev, K,
output_filepath_1, output_filepath_2]
        return params

    def isLicensed(self):
        """Set whether tool is licensed to execute."""
        return True

    def updateParameters(self, parameters):
        """Modify the values and properties of parameters before internal
        validation is performed.  This method is called whenever a parameter
        has been changed."""
        return

    def updateMessages(self, parameters):
        """Modify the messages created by internal validation for each tool
        parameter.  This method is called after internal validation."""
        return

    def execute(self, parameters, messages):
        """The source code of the tool."""

        input_ras = parameters[0].valueAsText
        R_mean = parameters[1].value
        G_mean = parameters[2].value
        B_mean = parameters[3].value
        R_stdev = parameters[4].value
        G_stdev = parameters[5].value
        B_stdev = parameters[6].value
        k = parameters[7].value
        output_filepath_1 = parameters[8].valueAsText
        output_filepath_2 = parameters[9].valueAsText

        ras = arcpy.Raster(input_ras)
        ras_a = arcpy.RasterToNumPyArray(ras)

        a = [R_mean, G_mean, B_mean]
        sd = [R_stdev, G_stdev, B_stdev]

        dist_a, ret_a = find_segments(ras_a, a, sd, k)
        dist = arcpy.NumPyArrayToRaster(dist_a, ras.extent.lowerLeft, ras.meanCellWidth,
ras.meanCellHeight)
        ret = arcpy.NumPyArrayToRaster(ret_a, ras.extent.lowerLeft, ras.meanCellWidth,
ras.meanCellHeight)

        dist.save(output_filepath_1)
        ret.save(output_filepath_2)
        return

    def find_segments(ras_a, a, sd, k):
        #get useful info from ras_a.shp
        nbands = 3
        nrows = ras_a.shape[1]
        ncols = ras_a.shape[2]

        #calculate radius of target color sphere
        radius = 0
        for b in range(nbands):
            radius += (k*sd[b])**2
        radius = math.sqrt(radius)

        #create a new zero distance array
        dist_a = np.zeros(ras_a.shape[1:3])

        #clone the original raster array
        ret_a = ras_a.copy()
```

```
#iterations
for r in range(nrows):
    for c in range(ncols):
        #calculate color distance for this call aat (r, c)
        for b in range(nbands):
            dist_a[r,c] +=(ras_a[b,r,c]-a[b])**2

        #sqrt to get the distance, notsum on squared distance
        dist_a[r,c] = math.sqrt(dist_a[r,c])

        #if this cell is outside the sphere
        if dist_a[r,c] > radius:
            #return a negative distance
            dist_a[r,c]= -dist_a[r,c]

            #return black
            for b in range(nbands):
                ret_a[b,r,c] = 0

return dist_a, ret_a
```

This code, once implemented into ARCPro, should produce the following interface:

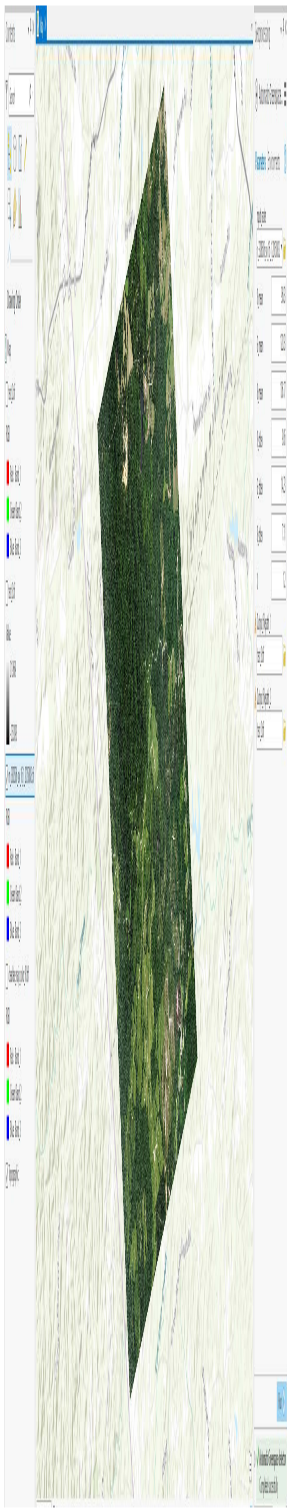

Figure 1. (Tool interface)

Here we have a total of ten parameters we must define. The first is a drop down menu in which we select the desired raster for analysis. Before filling out the rest of the parameters, you should take a number of samples of your area of interest. The more samples taken, the more accurate your results should be. The following three parameters, labeled R_mean, G_mean, and B_mean respectively, are for your average red, green, and blue bands across the samples taken. The next three parameters, labeled R_stdev, G_stdev, and B_stdev respectively, are for the standard deviation of your red, green, and blue samples taken. The parameter labeled “K” is essentially a modifier to increase or decrease the range of colors allowed inside your target area. It is recommended you start with a K value of 1 and increase or decrease this input as needed. The last two parameters are outputs for the two raster files this tool will create for you. The first being a color distance raster determining the distance of each pixel from your target range, the second being a strictly black or white raster with pixels with your range in while and pixels outside of your range being displayed as black. If parts of the desired area of interest are displayed in black the input for K should be increased, and if parts outside of your area of interest are being displayed in white, your K value should be decreased.

Results

The tools is displayed correctly within the ACRPro user interface. Perhaps some of the instructions for input could be more clear. As far as the actual application of the tool to manipulate your desired raster is works rather well, displaying a clear contrast between areas within and outside your desired color range. The example below is an example of tree cover identification in an area approximately four miles southwest of Cedartown, Georgia(Fig. 2 & 3). The one major and yet unresolved issue with the tool is the georeferenceing
of the resulting outputs.

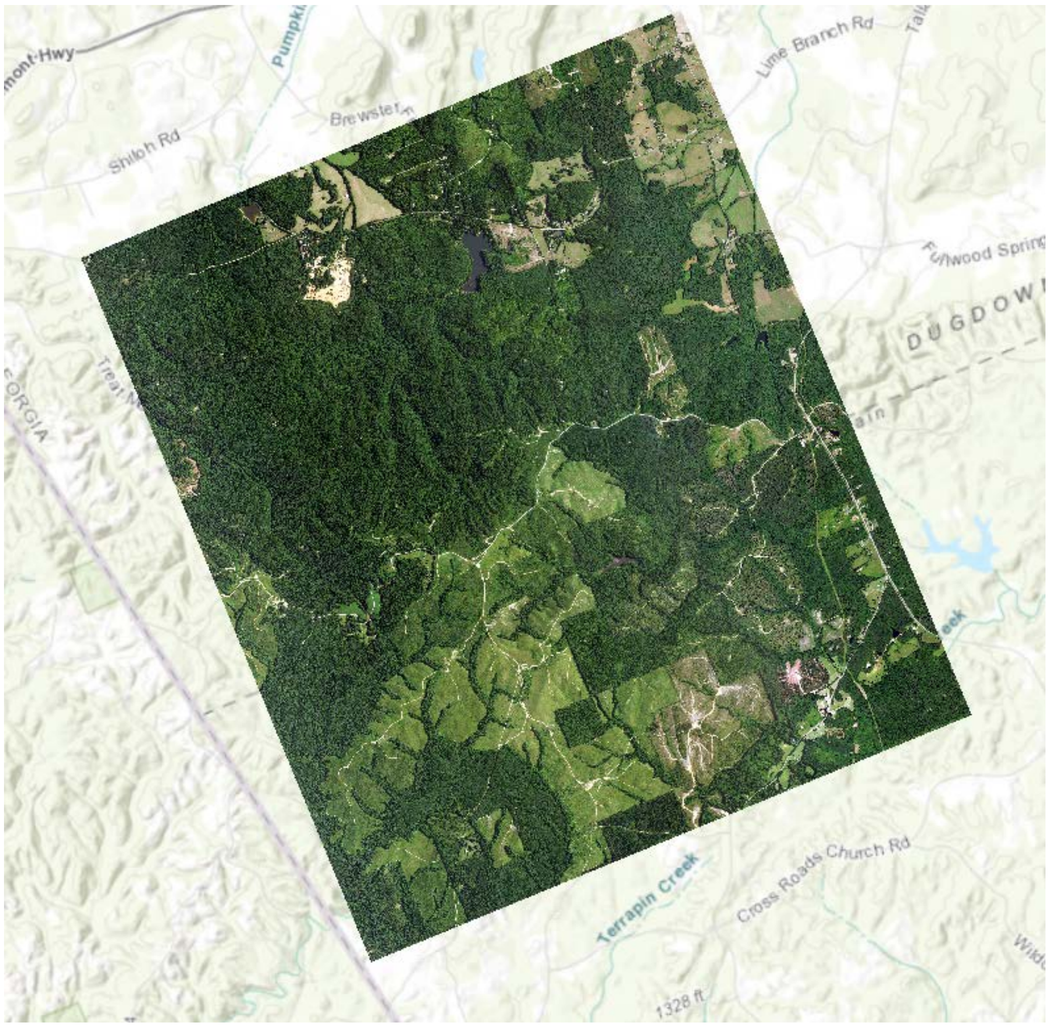

Figure 2 (Original Raster)

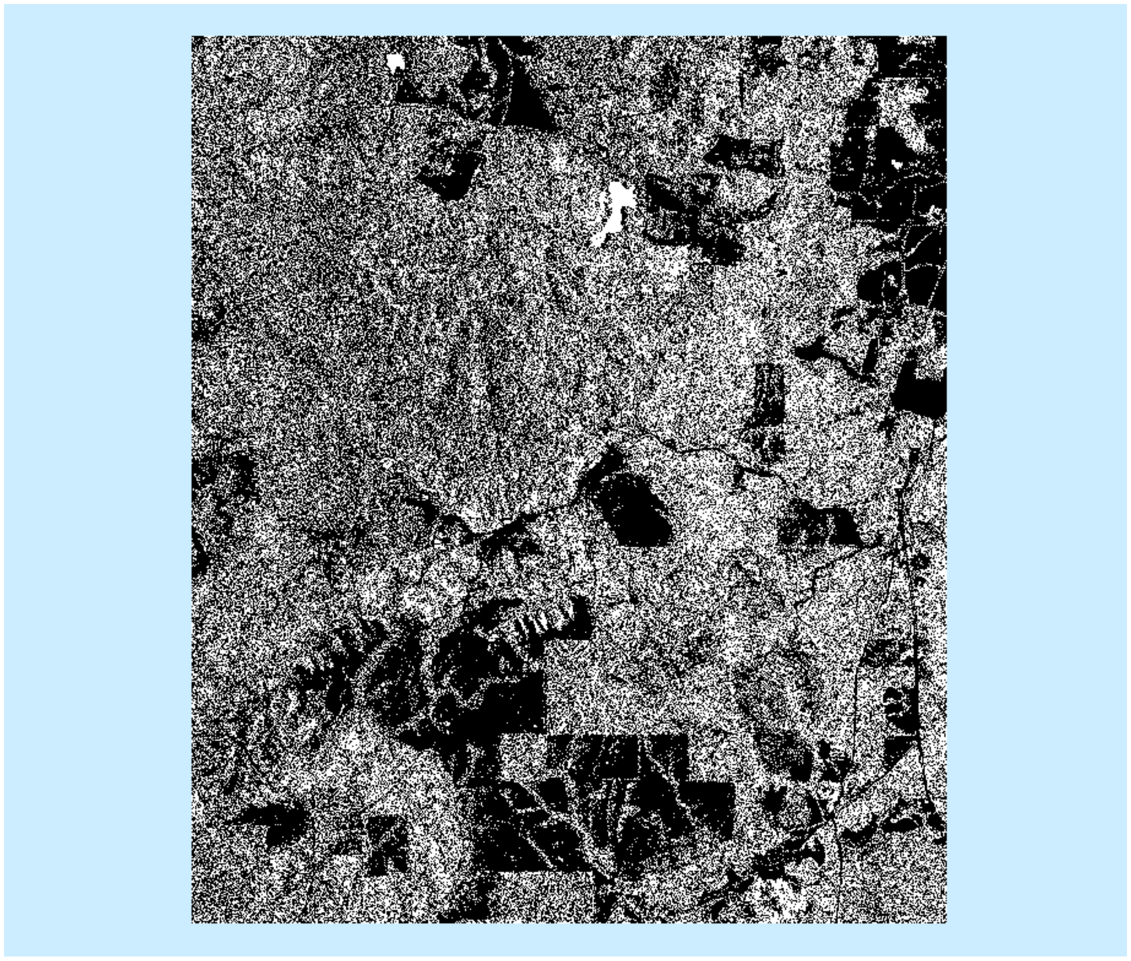

Figure 3. (Resulting Output Raster)

References

Hance, G. A., Umbaugh, S. E., Moss, R. H., & Stoecker, W. V. (1996). Unsupervised color image segmentation: With application to skin tumor borders. *IEEE Engineering in Medicine and Biology Magazine*, *15*(1), 104-111.

Dhanachandra, N., Manglem, K., & Chanu, Y. J. (2015). Image Segmentation Using K - means Clustering Algorithm and Subtractive Clustering Algorithm. *Procedia Computer Science, 54*, 764-771

J. Fritsch, S. Lang, A. Kleinehagenbrock, G. A. Fink and G. Sagerer, "Improving adaptive skin color segmentation by incorporating results from face detection," *Proceedings. 11th IEEE International Workshop on Robot and Human Interactive Communication*, Berlin, Germany, 2002, pp. 337-343