COG 260 Final Project Report: Color are universal

Mingyang Li 1004931425
Cheng Yu Chung 1005415953

Abstract

In this project, we are aiming to test the statement that focal colours are universal across languages. To do so, we will be reproducing the result from the study, Focal colors are universal after all, by Regier and Kay (2005). Specifically, there are two hypotheses that we will be testing, one is to show that 1) these focal colours will cluster around the prototypes for English white, black, red, yellow, green and blue. The other one is to show that 2) these focal colours will cluster more closely than the centers of category extensions, meaning the best examples reflect the universal structure around which color categories are formed. The first study utilizes the WCS data to validate Berlin and Kay's theory that these six English colour terms will show clustered foci, and we found the exact phenomenon as expected. Hence, even though the dataset Berlin and Kay used lacked variety, with more data from WCS, the pattern of universal foci is still evident after including more unwritten language. In particular, the best examples of the English terms appear to be the prototype as well because the figure displayed the foci chips in the middle of the cluster. The second study used CIEL*a*b colour space from WCS, which provides a psychologically meaningful distance metric as needed for this study. We started off by calculating the center of category extension of each color term in each language. Took the average of these centroids and coerced it to the chip most similar to it in the stimulus array. Next, we calculated the focus of each color term in each language as follows: we found chips which speakers have named for each color term in each language. Then, we selected the chip in the WCS array that received the maximum number of best-example choices for that term. Once we have two single-point representations for each color term: a centroid and a focus, we then restrict our attention to those terms for which we had both a centroid and a focus (filter out missing terms). Lastly, we calculated the centroid separation and focus separation, and performed statistical analysis (i.e., pairs t-test). In the results, the mean difference shows that focal colours ($M = 5593.89$) do cluster more closely than the centers of category extensions ($M = 5645.05$), yet the p-value from the paired t-test ($p > 0.05$) shows no evidence suggesting the difference between them. Study1 of this paper has successfully replicated Regier et al. 's (2005) results and its finding supports that focal colours are universal between industrial and non-industrial societies, which agrees to hypothesis 1). In terms of study 2, this paper could not reproduce an identical result as the Regier et al. 's (2005) paper. The discrepancy of the result between study2 of this paper and the original paper could be due to the difference in how the centroids were calculated, which Regier et al. 's (2005) might have calculated the most similar chip based on a different set of criteria. Thus, we expect follow-up researches can investigate using different sets of criteria to produce the result of study 2 and compare the differences across different methods.

Introduction

According to the Berlin and Kay theory, color naming patterns have the tendency to be universal, more specifically, language causes color categories to be centered strictly around certain focal points (Hardin, 2013). As a result of their study, they concluded that these six basic colors share universal foci across languages: white, black, red, yellow, green, and blue (Hardin 2013). However, their findings were questioned and challenged by others because they used a sample size of only 20 different languages in their study (Cook et al., 2005). To further examine their theory and complete the study with more data, the World Color Survey utilized similar methods to collect data from 110 unwritten language speakers. The WCS asked people to name colors in as few terms as possible, before showing them the Musell chips stimulus array that was used in Berlin and Kay's study, whereas the original research began with naming and best example picking without any prior procedures.

Subsequently, more recent studies suggest that language is not the constraint of color naming, and in fact color categorization is a derivative of one's cultural encounters (Roberson et al., 1999, 2000). They argue that it is even more unlikely for the category to be bounded by a prototype of a color, and the boundary of a color category would vary across languages. In other words, they think the term used for a color already has a defined boundary, based on the language background, and the best representation of that color is determined afterwards. Roberson et al's (2000) view on color categorization follows the prototype theory, that a category has already chosen what to be included first, then a prototype is calculated based on that cluster of items. New study indicated that it is possible that Berlin and Kay's results were confounded by one similarity of their tested languages, which are mostly from industrialized countries (Regier et al. 2005). Thus, to test this hypothesis, the WCS included a bigger sample size with industrialized countries' language to investigate the universal foci of colors. The study by Regier et al. (2005) made the following hypothesis to examine the suggested theory:

1) These focal colours will cluster around the prototypes for English white, black, red, green and blue (Regier et al. 2005).

2) These focal colours will cluster more closely than the centers of category extensions (Regier et al. 2005).

Methods

Study 1:

The first study uses the data from the primary WCS and Berlin & Kay (BK). The primary WCS contains the naming and foci data of languages from 110 non-industrialized countries. The foci data collects the best example(s) each speaker chose for the given color terms. The format of the foci data follows the stimulus array for 330 chips described by a hue and a lightness index. The BK data contains the foci data of speakers from 20 industrialized countries. The original data is cleaned using the provided helper functions. Then we record the number of hits on each color chip by creating an empty dictionary that corresponds to the stimulus array; 10 levels of lightness as the keys, and each key holds an array with 41 indices to represent 41 different hues. The value of a chip will go up by one each time it is picked to be the focal color of a color category. Subsequently, we transform the dictionary into a 2D-array and plot it as a density plot. Finally, we find the best examples of each English color term from the BK data, and label the corresponding coordinates on the same plot with the color of the terms.

Study 2:

The second study uses CIEL*a*b colour space from WCS, which provides a psychologically meaningful distance metric as needed for this study. This colour space data, which involves the CIEL*a*b* coordinates for each of the 330 stimulus chips is fetched from the World Color Survey Munsell Color Chip Mapping Table. This mapping table relate the WCS and the Munsell coordinates, as well as the CIEL*a*b* coordinates for each of the 330 stimulus chips. To test our hypothesis, we must calculate the center of category extensions for each term in each language of the naming data. For each speaker and the colour term t they named, we find the centroid in CIEL*a*b* of the chips that the speaker has named. Then we take the average of the centroids we found previously and match it to the most similar chip in the stimulus array. For each language, we calculate the focus of each colour term by selecting the chip that received the maximum number of choices from the speakers that indicate the best example for that colour term. In a case where there are more than one chip receiving the maximum number of choices, we break the tie by choosing randomly among these chips with the same number of choices. We now have two single-point representations for each color term: a centroid and a focus. We filtered out terms that are either missing a centroid or a focus, keeping terms that have both. Finally, for each language l in the WCS, we calculate the centroid separation (CSl) and focus separation (FSl) from the Berlin and Kay data set using the two formulae below, which are similar to the nearest neighbors algorithm.

$$CS_l = \sum_{t \in l} \sum_{l^* \in BK} \min_{t^* \in l^*} \text{distance}[c(t), c(t^*)].$$

$$FS_l = \sum_{t \in l} \sum_{l^* \in BK} \min_{t^* \in l^*} \text{distance}[f(t), f(t^*)].$$

Where t is a term in language l from WCS data and t* is a term in language l* from the Berlin and Kay data (BK). We then did further statistical analysis to test our second hypothesis. Applying a paired t test between centroid separation (CSI) and focus separation(FSI) to see the mean differences between these paired measurements. If the result yields that the best examples of colour categories cluster more tightly than the center of category extensions, it supports our second hypothesis that the best examples reflect the formation of colour categories but not derived by the colour categories.

Results

Study 1:

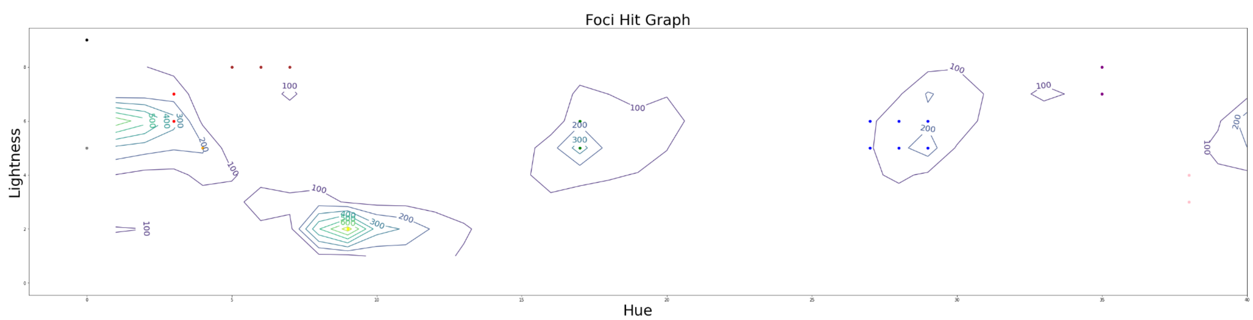


Figure 1. Lightness is on a scale from 0 to 9 corresponding to the WCS scale of A to J respectively.

The figure of the first study is identical to Reign and Kay's (2005) result. The red, yellow, green and blue dots, which represent the best examples of the English colour terms, all fall in the clusters of the WCS dataset. The other colors such as orange, brown, pink and purple did not have significant clusters form under them. Furthermore, the English foci for black and white falls exactly at the coordinate that represents A0 and J0 on the achromatic chips. We know the speakers in the WCS data also chose this to be the best examples of black and white, because the dictionary recorded 2359 and 2422 hits on those chips, much higher than any of the center hits of the clusters. Although English had B0 as a foci chip for white, it is one chip away from A0.

Study 2

In [177]:
```
total_dist = []
for lang_wcs, terms_wcs in centroids["wcs"].items():
    dist = 0
    for term_wcs, coord_wcs in terms_wcs.items():
        for lang_bk, terms_bk in centroids["bk"].items():
            bks = []
            for term_bk, coord_bk in terms_bk.items():
                bks.append(coord_bk)
            tree = spatial.KDTree(bks)
            dist += tree.query(coord_wcs)[0]
    total_dist.append(dist)
print(total_dist[:10])
print("CS mean:", np.mean(total_dist))
```

```
[3386.479947851556, 11444.810169594253, 3859.500961845106, 7923.350943780291, 3305.648181071123, 13460.768969964978,
8645.602810005103, 4589.890292532931, 2615.2899078791997, 6693.804958537411]
CS mean: 5645.046205688967
```

In [178]:
```
total_dist_focus = []
for lang_wcs_focus, terms_wcs_focus in focus["wcs"].items():
    dist_bk_focus = 0
    for term_wcs_focus, coord_wcs_focus in terms_wcs_focus.items():
        for lang_bk_focus, terms_bk_focus in focus["bk"].items():
            bk_focus = []
            for term_bk_focus, coord_bk_focus in terms_bk_focus.items():
                bk_focus.append(coord_bk_focus)
            tree_focus = spatial.KDTree(bk_focus)
            dist_bk_focus += tree_focus.query(coord_wcs_focus)[0]
    total_dist_focus.append(dist_bk_focus)
print(total_dist_focus[:10])
print("FS mean:", np.mean(total_dist_focus))
```

```
[3738.8918637147012, 13647.580600201483, 3131.4790349303385, 8427.61021390781, 2691.5706689371705, 13051.01310268482
3, 8698.171198794149, 4358.784274172861, 2358.447302478546, 7199.790096907786]
FS mean: 5593.888682198148
```

In [179]:
```
print("p-value:", stats.ttest_rel(total_dist,  total_dist_focus)[1])
```

```
p-value: 0.3918045229384892
```

Looking at the figure above, the mean of our centroid separation across all languages in WCS is 5645.05, which is significantly different from the result found in Reign and Kay's study (2005) with M = 6391.78. On the other hand, the mean of our focus separation across all languages in WCS is 5593.89, which is almost identical to Reign and Kay's result where M = 5,596.98. The p-value from our paired t-test ($p = 0.39$) shows that there is no evidence suggesting the difference between the centroid separation and the focus separation. For both the mean of the centroid separation and the p-value from the paired t-test, we obtained different results than Reign and Kay's study (2005). The possible factors that led to the difference between our results and Reign and Kay's result will be discussed in the next section. Here, I would like to highlight the findings based on our results. If we solely consider the mean difference, the FSI is indeed smaller than CSI, which suggests that the best examples of colour categories cluster more tightly than the center of category extensions. The mean difference supports our second hypothesis that the best examples reflect the universal structure around which color categories are formed. Nonetheless, the paired t-test says otherwise, which there is no evidence to suggest their difference and refutes our hypothesis.

Conclusion

To conclude, study 1 of this paper has successfully replicated Regier et al.'s (2005) results and our finding supports that focal colours are universal between industrial and non-industrial societies, in which the results from our study 1 successfully showed that foci in WCS will cluster around the prototypes for English white, black, red, yellow, green and blue. This further shows

Berlin and Kay's theory that color naming is universal across languages, because we produced the exact results of which they find to be dominated by the theory. In terms of study 2, in a sense, we were also able to replicate a similar result as the one from Reign and Kay's study, that is the FSI is indeed smaller than CSI. Solely looking at the mean difference between CSI and FSI, this indeed agrees with Reign and Kay's findings that "the best examples reflect universal foci against a background of cross-linguistically varying category extensions". However, our paired t-test refutes this hypothesis since it showed no evidence suggesting the differences between CSI and FSI. After several times of code debugging processes and inspections, our p-value from the t-test and the CSI value yield the same results, thus, it is less likely that an unintentional mistake has been made in our codes. Given that there is a significant discrepancy between the CSI value from Reign and Kay's study and our paper, one possibility could be the difference in how the centroids were calculated. Specifically, since the original study did not explicitly mention how the most similar chip was determined, they have probably used a different method or a set of criterias to find the most similar chips from the stimulus array. In our case, it was rather straightforward and obvious to find the most similar chips based on its location in the 3D CIEL*a*b colour space from WCS. Some possible extensions follow up in this study could be testing different sets of criterias to find the most similar chip. For instance, the future researchers can try to determine the most similar chip solely based on lightness in the stimulus array, perhaps, some other combinations of Hue, chroma and lightness.

References

Cook, R. S., Kay, P., & Regier, T. (2005). The world colour survey database. *Handbook of Categorization in Cognitive Science*, 223-241. https://doi.org/10.1016/B978-008044612-7/50064-0

Davidoff, J., Davies, I., & Roberson, D. (1999). Colour categories in a stone-age tribe. *Nature*, *398*(6724), 203-204. https://doi.org/10.1038/18335

Hardin, C.l. (2013). Berlin and kay theory. *Encyclopedia of Color Science and Technology*, 1-4. https://doi.org/10.1007/978-3-642-27851-8_62-2

Regier, T., Kay, P., & Cook, R. S. (2005). Focal colors are universal after all. *Proceedings of the National Academy of Sciences*, *102*(23), 8386-8391. https://doi.org/10.1073/pnas.0503281102

Roberson, D., Davies, I., & Davidoff, J. (2000). Color categories are not universal: Replications and new evidence from a stone-age culture. *Journal of Experimental Psychology: General*, *129*(3), 369-398. https://doi.org/10.1037/0096-3445.129.3.369

Appendix

In the following pages of this pdf

Untitled

December 13, 2021

[8]:
```
from wcs_helper_functions import *
import numpy as np
from scipy import stats
from random import random
%matplotlib inline
```

[9]:
```
munsellInfo = readChipData('./WCS_data_core/chip.txt');
indexCoord = munsellInfo[1]
coordIndex = munsellInfo[0]
fociData = readFociData('./WCS_data_core/foci-exp.txt');
namingData = readNamingData('./WCS_data_core/term.txt')
BKDict =  readBKDictData('./BK-dict.txt')

BKFoci = readBKFociData("./BK-foci.txt")
```

[10]:
```
#create empty dict
fociHitDict = {}
for item in coordIndex:
    key = item[0]
    if (key not in fociHitDict):
        fociHitDict[key] = [0]*41

print(fociHitDict)
```

```
{'E': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'C': [0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0], 'F': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'I':
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'H': [0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0], 'G': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'B': [0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'D': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0], 'J': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'A': [0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0]}
```

[11]:
```
#count foci chosen by different language speakers
for lang in fociData:
    for speaker in fociData[lang]:
        for foci in fociData[lang][speaker]:
            for coord in fociData[lang][speaker][foci]:
                coord = coord.split(":")
                row = coord[0]
                col = int(coord[1])
                #fitler out column 0 and row A and J which are achromatic chips
                #if row not in ['A', 'J']:
                fociHitDict[row][col] += 1
print(fociHitDict)

fociHitLst = []
for key in sorted(fociHitDict.keys()):
    fociHitLst.append(fociHitDict[key][1:])
#fociHitLS.sort()
print(fociHitLst)
```

```
{'E': [139, 98, 69, 68, 131, 116, 37, 41, 37, 34, 31, 31, 28, 28, 26, 76, 155,
117, 110, 95, 81, 52, 49, 41, 48, 58, 69, 85, 118, 83, 44, 27, 27, 32, 28, 28,
28, 35, 66, 78, 81], 'C': [66, 102, 100, 88, 81, 85, 67, 65, 492, 752, 351, 286,
181, 116, 57, 37, 29, 30, 32, 35, 35, 26, 27, 22, 24, 27, 34, 29, 27, 33, 28,
21, 27, 25, 24, 39, 36, 61, 74, 64, 80], 'F': [160, 276, 240, 210, 216, 37, 33,
46, 44, 50, 32, 24, 27, 28, 25, 53, 157, 351, 156, 151, 102, 61, 47, 39, 40, 49,
65, 90, 174, 253, 88, 30, 42, 65, 44, 31, 36, 41, 59, 131, 198], 'I': [113, 102,
102, 78, 73, 85, 86, 83, 75, 66, 79, 59, 58, 55, 56, 46, 48, 48, 54, 54, 55, 47,
45, 46, 45, 41, 48, 51, 60, 76, 95, 93, 66, 73, 69, 59, 63, 59, 83, 75, 96],
'H': [91, 126, 140, 143, 58, 37, 81, 107, 79, 51, 39, 22, 25, 15, 14, 24, 23,
125, 99, 79, 94, 47, 23, 20, 22, 24, 31, 43, 88, 210, 159, 95, 85, 112, 95, 60,
73, 51, 61, 58, 69], 'G': [144, 665, 541, 342, 77, 45, 65, 77, 69, 55, 33, 26,
21, 24, 18, 24, 39, 171, 185, 119, 147, 70, 45, 32, 26, 30, 57, 75, 184, 178,
190, 63, 55, 66, 59, 40, 44, 39, 69, 83, 260], 'B': [109, 86, 81, 83, 85, 82,
81, 74, 73, 71, 117, 139, 132, 87, 61, 64, 63, 68, 71, 74, 79, 80, 77, 75, 73,
72, 66, 66, 71, 75, 71, 70, 72, 70, 74, 71, 70, 78, 80, 77, 87], 'D': [77, 75,
74, 56, 51, 46, 174, 130, 153, 84, 66, 69, 52, 40, 36, 57, 71, 75, 58, 54, 52,
44, 34, 29, 36, 41, 40, 42, 60, 50, 33, 16, 18, 23, 15, 24, 29, 39, 54, 51, 71],
'J': [2359, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'A': [2422, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0]}
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [86, 81, 83, 85, 82, 81, 74, 73, 71,
117, 139, 132, 87, 61, 64, 63, 68, 71, 74, 79, 80, 77, 75, 73, 72, 66, 66, 71,
```

```
75, 71, 70, 72, 70, 74, 71, 70, 78, 80, 77, 87], [102, 100, 88, 81, 85, 67, 65,
492, 752, 351, 286, 181, 116, 57, 37, 29, 30, 32, 35, 35, 26, 27, 22, 24, 27,
34, 29, 27, 33, 28, 21, 27, 25, 24, 39, 36, 61, 74, 64, 80], [75, 74, 56, 51,
46, 174, 130, 153, 84, 66, 69, 52, 40, 36, 57, 71, 75, 58, 54, 52, 44, 34, 29,
36, 41, 40, 42, 60, 50, 33, 16, 18, 23, 15, 24, 29, 39, 54, 51, 71], [98, 69,
68, 131, 116, 37, 41, 37, 34, 31, 31, 28, 28, 26, 76, 155, 117, 110, 95, 81, 52,
49, 41, 48, 58, 69, 85, 118, 83, 44, 27, 27, 32, 28, 28, 28, 35, 66, 78, 81],
[276, 240, 210, 216, 37, 33, 46, 44, 50, 32, 24, 27, 28, 25, 53, 157, 351, 156,
151, 102, 61, 47, 39, 40, 49, 65, 90, 174, 253, 88, 30, 42, 65, 44, 31, 36, 41,
59, 131, 198], [665, 541, 342, 77, 45, 65, 77, 69, 55, 33, 26, 21, 24, 18, 24,
39, 171, 185, 119, 147, 70, 45, 32, 26, 30, 57, 75, 184, 178, 190, 63, 55, 66,
59, 40, 44, 39, 69, 83, 260], [126, 140, 143, 58, 37, 81, 107, 79, 51, 39, 22,
25, 15, 14, 24, 23, 125, 99, 79, 94, 47, 23, 20, 22, 24, 31, 43, 88, 210, 159,
95, 85, 112, 95, 60, 73, 51, 61, 58, 69], [102, 102, 78, 73, 85, 86, 83, 75, 66,
79, 59, 58, 55, 56, 46, 48, 48, 54, 54, 55, 47, 45, 46, 45, 41, 48, 51, 60, 76,
95, 93, 66, 73, 69, 59, 63, 59, 83, 75, 96], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0]]
```

[12]:
```
print(BKDict[6])
```

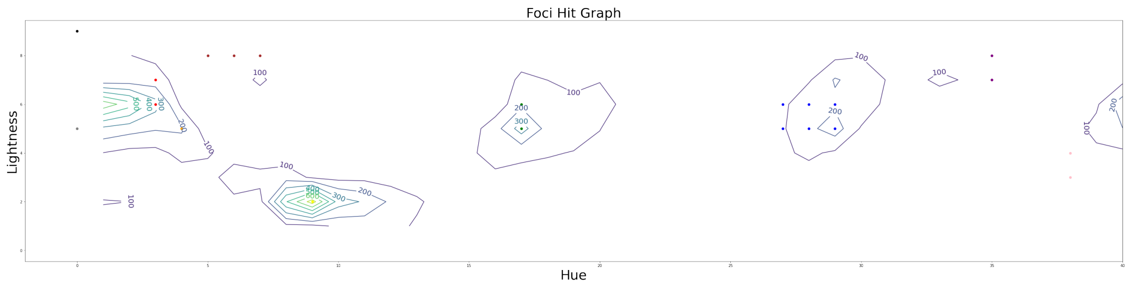
```
{6: {'BU': 'blue'}, 4: {'GN': 'green'}, 8: {'PU': 'purple'}, 9: {'PI': 'pink'},
10: {'OR': 'orange'}, 7: {'BR': 'brown'}, 3: {'RE': 'red'}, 11: {'GY': 'grey'},
5: {'YE': 'yellow'}, 1: {'BA': 'black'}, 2: {'WH': 'white'}}
```

[13]:
```
print(BKFoci[6])
```

```
{1: {1: ['J0'], 2: ['A0B0'], 3: ['G3H3'], 4: ['F17G17'], 5: ['C9'], 6:
['F27..29G27..29'], 7: ['I5..7'], 8: ['H35I35'], 9: ['D38E38'], 10: ['F4'], 11:
['F0']}}
```

[21]:
```
x, y = np.meshgrid(range(1, 41), range(1, 9))
fig = plt.figure(figsize=(41, 9))
axis = fig.add_axes([1, 1, 1, 1])
cont = axis.contour(x, y, fociHitLst[1:9])
axis.clabel(cont, fontsize=20)
axis.set_title('Foci Hit Graph', fontsize=36)
axis.set_xlabel('Hue', fontsize=36)
axis.set_ylabel('Lightness', fontsize=36)

# match the coordinate with the BK foci data
fociDict = {'black': [[9, 0]], 'white': [[0, 0], [1, 0]], 'red': [[6, 3], [7,␣
↪3]], 'green': [[5, 17], [6, 17]],
            'yellow': [[2, 9]], 'blue': [[5, 27], [5, 28], [5, 29], [6, 27],␣
↪[6, 28], [6, 29]],
            'brown': [[8, 5], [8, 6], [8, 7]], 'purple': [[7, 35], [8, 35]],␣
↪'pink': [[3, 38], [4, 38]],
```

```
            'orange': [[5, 4]], 'grey': [[5, 0]]}

for color in fociDict:
        for val in fociDict[color]:
            plt.plot(val[1], val[0], 'ro', color=color)
fig.show()
```



[ ]:

study2

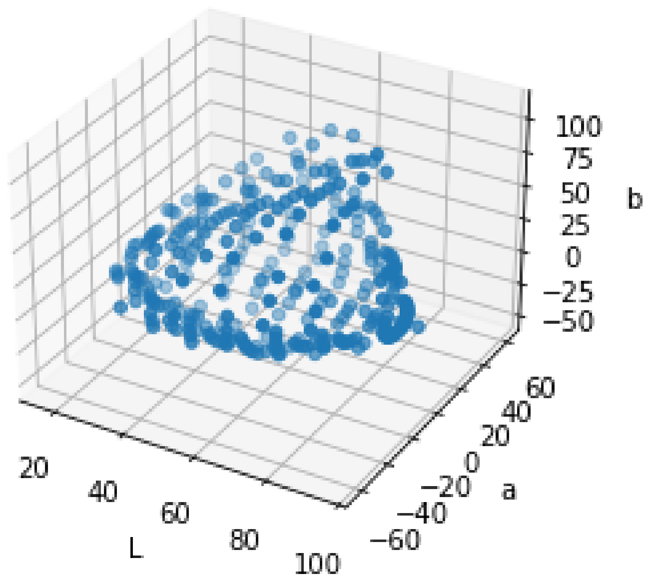
December 14, 2021

[167]:
```
from wcs_helper_functions import *
import numpy as np
from scipy import stats, spatial
```

[168]:
```
munsellInfo = readChipData('./WCS_data_core/chip.txt')
indexCoord = munsellInfo[1]
coordIndex = munsellInfo[0]
cielabCoord = readClabData('./WCS_data_core/cnum-vhcm-lab-new.txt')
```

[169]:
```
Ls = []
A = []
B = []
for index, coords in cielabCoord.items():
    L, a, b = np.array(coords).astype(np.float)
    Ls.append(L)
    A.append(a)
    B.append(b)
ax = plt.axes(projection='3d')
ax.scatter(Ls, A, B)
ax.set_xlabel("L")
ax.set_ylabel("a")
ax.set_zlabel("b")
```

[169]:
```
Text(0.5, 0, 'b')
```




```
[170]: def centroid(data):
           res = {}
           for lang, speakers in data.items():
               terms = {}
               for speaker, chips in speakers.items():
                   for chip, term in chips.items():
                       if term not in terms:
                           terms[term] = []
                       terms[term].append(cielabCoord[chip])
               for t, coords in terms.items():
                   terms[t] = np.mean(np.array(coords)[:, -3:].astype(np.float),␣
        ↪axis=0)
               res[lang] = terms
           return res
       centroid(namingData)[1]
```

```
[170]: {'LB': array([ 37.3253854 , -13.25516734,   2.23640974]),
        'LE': array([52.90576327, 41.20450612, 23.82702857]),
        'WK': array([59.15118717, 10.78465241, 27.25052406]),
        'LF': array([86.17827119, -4.04503955,  0.3179096 ]),
        'F': array([70.85285519, -4.31862022, 14.50961066]),
        'G': array([ 47.94192615, -13.62643713, -17.09810379]),
        'S': array([53.23678899,  2.80614679, 15.98816514]),
        'GB': array([ 42.95411765, -10.67647059,  17.17411765]),
        'FU': array([68.36385417, -7.22854167, 18.0975    ])}
```

```
[171]: def cloest_chips(centroid):
    res = {}
    coords = []
    chips = []
    for chip_index, coord in cielabCoord.items():
        chips.append(chip_index)
        coords.append(np.array(coord).astype(np.float))
    tree = spatial.KDTree(coords)
    for lang, terms in centroid.items():
        cloest_chip = {}
        for term, centroid in terms.items():
            cloest_chip[term] = coords[int(tree.query(centroid)[1])]
        res[lang] = cloest_chip
    return res
cloest_chips(centroid(namingData))[1]
```

```
[171]: {'LB': array([ 4.122e+01, -3.000e-02,  3.000e-02]),
 'LE': array([61.7 , 48.53, 25.92]),
 'WK': array([41.22,  9.31, 37.8 ]),
 'LF': array([ 8.135e+01, -5.000e-02,  6.000e-02]),
 'F': array([ 7.16e+01, -4.00e-02,  5.00e-02]),
 'G': array([ 41.22, -20.46, -17.64]),
 'S': array([ 5.157e+01, -3.000e-02,  4.000e-02]),
 'GB': array([ 30.77, -13.04,  21.97]),
 'FU': array([ 7.16e+01, -4.00e-02,  5.00e-02])}
```

```
[172]: def chip_select(data):
    total_chips = {}
    for lang, speakers in data.items():
        total_chips_term = {}
        term_lst = []
        for speaker, terms in speakers.items():
            for term, chips in terms.items():
                if term not in term_lst:
                    term_lst.append(term)
                    total_chips_term[term] = []
                total_chips_term[term].extend(chips)
        total_chips[lang] = total_chips_term
    return total_chips
chip_select(fociData)[1]["WK"]
```

```
[172]: ['D:9',
 'D:10',
 'D:11',
 'D:12',
 'E:1',
 'E:3',
```

```
'E:5',
'C:10',
'D:2',
'D:3',
'D:4',
'D:5',
'B:12',
'E:2',
'C:11',
'D:38',
'C:1',
'C:2',
'C:3',
'C:4',
'C:5',
'D:6',
'D:7',
'C:4',
'C:5',
'C:6',
'E:38',
'E:39',
'E:40',
'B:16',
'D:8',
'D:5',
'F:6',
'F:35',
'F:7',
'F:8',
'E:1',
'C:1',
'C:6',
'C:7',
'C:3',
'C:4']
```

[173]:
```
def foci(selected_chips):
    foci_res = {}
    for lang, terms in selected_chips.items():
        foci_term = {}
        for term, chips in terms.items():
            chips_dict = dict((chip, chips.count(chip)) for chip in set(chips))
            foci_term[term] = np.array(cielabCoord[coordIndex[max(
                chips_dict, key=chips_dict.get).replace(":", "")]]).astype(np.
↪float)
        foci_res[lang] = foci_term
```

```
    return foci_res
foci(chip_select(fociData))[1]
```

[173]:
```
{'LF': array([ 9.6e+01, -6.0e-02,  6.0e-02]),
 'WK': array([81.35, 21.06, 22.4 ]),
 'F': array([91.08,  5.21,  7.67]),
 'LB': array([15.6 , -0.02,  0.02]),
 'G': array([ 41.22,   7.17, -48.94]),
 'LE': array([41.22, 61.4 , 17.92]),
 'S': array([61.7 , 49.15, 56.82]),
 'GB': array([ 71.6 , -24.87,  85.15]),
 'FU': array([ 71.6 , -12.05, -29.46])}
```

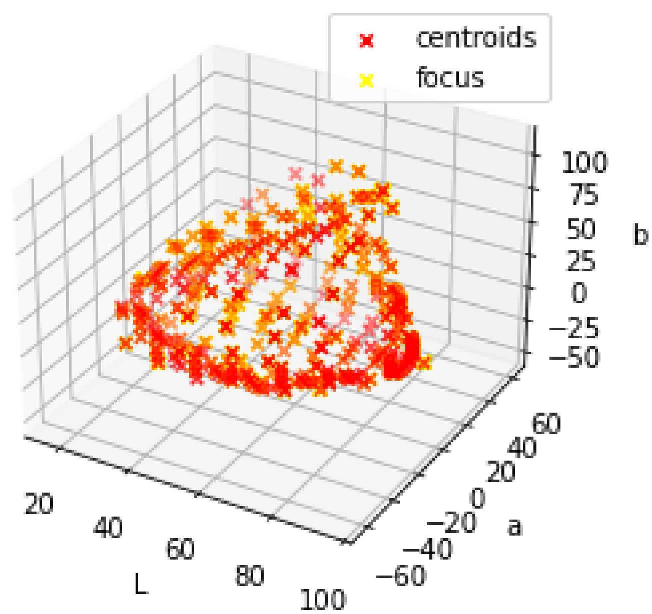
[174]:
```
namingData = readNamingData('./WCS_data_core/term.txt')
fociData = readFociData('./WCS_data_core/foci-exp.txt')

namingDataBK = readNamingData('./bk_data/BK-term.txt')
fociDataBK = readFociDataBK('./bk_data/BK-foci.txt')
```

[175]:
```
centroids = {
    "bk": cloest_chips(centroid(namingDataBK)),
    "wcs": cloest_chips(centroid(namingData))
}

focus = {
    "bk": {lang: {term: colour for term, colour in terms.items() if term in␣
 ↪centroids["bk"][lang]} for lang, terms in foci(chip_select(fociDataBK)).
 ↪items()},
    "wcs": {lang: {term: colour for term, colour in terms.items() if term in␣
 ↪centroids["wcs"][lang]} for lang, terms in foci(chip_select(fociData)).
 ↪items()}
}

centroids = {
    "bk": {lang: {term: colour for term, colour in terms.items() if term in␣
 ↪focus["bk"][lang]} for lang, terms in centroids["bk"].items()},
    "wcs": {lang: {term: colour for term, colour in terms.items() if term in␣
 ↪focus["wcs"][lang]} for lang, terms in centroids["wcs"].items()}
}
```

[176]:
```
Lsc = []
Ac = []
Bc = []
for i,j  in centroids["wcs"].items():
    for k, l  in j.items():
        L, a, b = np.array(l).astype(np.float)
        Lsc.append(L)
```

```
        Ac.append(a)
        Bc.append(b)
Lsf = []
Af = []
Bf = []
for i,j  in focus["wcs"].items():
    for k, l  in j.items():
        L, a, b = np.array(l).astype(np.float)
        Lsf.append(L)
        Af.append(a)
        Bf.append(b)


ax = plt.axes(projection='3d')
ax.scatter(Ls, A, B, c="red", marker="x", label="centroids")
ax.scatter(Lsf,Af, Bf, c="yellow", marker="x", label="focus")
ax.set_xlabel("L")
ax.set_ylabel("a")
ax.set_zlabel("b")
ax.legend()
```

[176]: <matplotlib.legend.Legend at 0x7fcc0dcb9e80>



[177]:
```
total_dist = []
for lang_wcs, terms_wcs in centroids["wcs"].items():
    dist = 0
```

```
    for term_wcs, coord_wcs in terms_wcs.items():
        for lang_bk, terms_bk in centroids["bk"].items():
            bks = []
            for term_bk, coord_bk in terms_bk.items():
                bks.append(coord_bk)
            tree = spatial.KDTree(bks)
            dist += tree.query(coord_wcs)[0]
    total_dist.append(dist)
print(total_dist[:10])
print("CS mean:", np.mean(total_dist))
```

```
[3386.479947851556, 11444.810169594253, 3859.500961845106, 7923.350943780291,
3305.648181071123, 13460.768969964978, 8645.602810005103, 4589.890292532931,
2615.2899078791997, 6693.804958537411]
CS mean: 5645.046205688967
```

[178]:

```
total_dist_focus = []
for lang_wcs_focus, terms_wcs_focus in focus["wcs"].items():
    dist_bk_focus = 0
    for term_wcs_focus, coord_wcs_focus in terms_wcs_focus.items():
        for lang_bk_focus, terms_bk_focus in focus["bk"].items():
            bk_focus = []
            for term_bk_focus, coord_bk_focus in terms_bk_focus.items():
                bk_focus.append(coord_bk_focus)
            tree_focus = spatial.KDTree(bk_focus)
            dist_bk_focus += tree_focus.query(coord_wcs_focus)[0]
    total_dist_focus.append(dist_bk_focus)
print(total_dist_focus[:10])
print("FS mean:", np.mean(total_dist_focus))
```

```
[3738.8918637147012, 13647.580600201483, 3131.4790349303385, 8427.61021390781,
2691.5706689371705, 13051.013102684823, 8698.171198794149, 4358.784274172861,
2358.447302478546, 7199.790096907786]
FS mean: 5593.888682198148
```

[179]:

```
print("p-value:", stats.ttest_rel(total_dist,  total_dist_focus)[1])
```

```
p-value: 0.3918045229384892
```

wcs_helper_functions

December 13, 2021

Code developed and shared by: Vasilis Oikonomou Joshua Abbott Jessie Salas

[ ]:
```
import numpy as np
from matplotlib.colors import LinearSegmentedColormap
import matplotlib.pyplot as plt
import re
from random import random
import numpy as np
from matplotlib import gridspec
import warnings
import string
warnings.filterwarnings('ignore')
```

[ ]:
```
def readNamingData(namingDataFilePath):
    """
    Read all of namingDataFilePath into a dictionary, and return it.  Assumes␣
 ↪data file follows WCS format:
    language number\tspeaker number\tchip number\tlanguage term for chip\n

    Parameters
    ----------
    namingDataFilePath : string
        The path (and filename, with the extension) to read the WCS-formatted␣
 ↪color naming data from.


    Returns
    -------
    namingData : dictionary
            A hierarchical dictionary:␣
 ↪namingData[languageNumber][speakerNumber][chipNumber] = languageTerm


    Example Usage:
    --------------
    import wcsFunctions as wcs
    namingDictionary = wcs.readNamingData('./WCS-Data/term.txt')

```

```
    """
    namingData = {}  # empty dict
    fileHandler = open(namingDataFilePath,'r')

    for line in fileHandler:                                      # for each␣
↪line in the file
        lineElements = line.split()                               # lineElements␣
↪are denoted by white space

        # WCS format for naming data from term.txt:
        # language number\tspeaker number\tchip number\tlanguage term for chip

        languageNumber = int(lineElements[0])              # 1st element is␣
↪language number, make it an int
        speakerNumber = int(lineElements[1])                    # 2nd is speaker␣
↪number, make int
        chipNumber = int(lineElements[2])                       # 3rd is chip␣
↪number, make int
        languageTerm = lineElements[3]                     # 4th is␣
↪languageTermegory assignment (term), keep string

        if not (languageNumber in namingData.keys()):    ␣
↪                                                    # if this language isn't a␣
↪key in the namingData dict
            namingData[languageNumber] = {}              ␣
↪                                                            # then make it one,␣
↪with its value an empty list
        if not (speakerNumber in namingData[languageNumber].keys()):  ␣
↪                              # if this speaker isn't a key in the languageNumber␣
↪dict
            namingData[languageNumber][speakerNumber] = {}            ␣
↪                                    # then make it one, with its value an empty␣
↪list

        namingData[languageNumber][speakerNumber][chipNumber] = languageTerm ␣
↪         # fill in these empty lists to make a GIANT namingData dictionary
                                          ␣
↪                                                                          #␣
↪where each entry looks like this: {1: {1: {1: 'LB'}}
                                          ␣
↪                                                                          #␣
↪and thus namingData[1][1][1] returns string 'LB'

    fileHandler.close()                                # close file after␣
↪reading it in, for neatness
    return namingData                                  # return the dict
```

[ ]:
```
def readFociData(fociDataFilePath):
    """
    Read all of fociDataFilePath into a dictionary, and return it. Assumes data␣
↪file follows WCS format:
    language number\tspeaker number\tterm number\tterm abbreviation\tWCS chip␣
↪grid coordinate

    Paramaters
    ----------
    fociDataFilePath : string
        The path (and filename, with the extension) to read the WCS-formatted␣
↪color foci data from.

    Returns
    -------
    fociData : dictionary
            A hierarchical dictionary:␣
↪fociData[languageNumber][speakerNumber][languageTerm].append(modGridCoord)


    Example Usage:
    --------------
    import wcsFunctions as wcs
    fociDictionary = wcs.readFociData('./WCS-Data/foci-exp.txt')

    """

    fociData = {} # empty dict
    fileHandler = open(fociDataFilePath,'r')
    for line in fileHandler:                                   # for each line in␣
↪the file
        lineElements = line.split()                  # elements are denoted by␣
↪white space

        # WCS format for naming data from foci-exp.txt:
        # language number\tspeaker number\tfoci number in term\tlanguage term␣
↪for chip\tWCS grid coordinate

        languageNumber = int(lineElements[0])        # 1st element is language␣
↪number, make it an int
        speakerNumber = int(lineElements[1])        # 2nd is speaker number,␣
↪make int
        termNumber = int(lineElements[2])                # 3rd is term number␣
↪for which foci goes to which term
        languageTerm = lineElements[3]                         # 4th is term␣
↪abbreviation
```

```
        gridcoord = lineElements[4]                                    # 5th is␣
↪chip grid coord

        # fix WCS bad entries - collapse A1-40 to A0 and J1-40 to J0
        if (gridcoord[0] == 'A'):
            if (int(gridcoord[1:]) > 0):
                gridcoord = "A0"

        if (gridcoord[0] == 'J'):
            if (int(gridcoord[1:]) > 0):
                gridcoord = "J0"

        modGridCoord = gridcoord[0] + ":" + gridcoord[1:]        # make it␣
↪nicer for parsing purposes later


        if not (languageNumber in fociData.keys()):        # if this language␣
↪isn't a key in the fociData dict
            fociData[languageNumber] = {}                          # then make it␣
↪one, with its value an empty list
        if not (speakerNumber in fociData[languageNumber].keys()):        # if␣
↪this speaker isn't a key in the languageNumber dict
            fociData[languageNumber][speakerNumber] =␣
↪{}                         # then make it one, with its value an empty list
        if not (languageTerm in fociData[languageNumber][speakerNumber].keys()):
↪         # if this term isn't a key in the speakerNumber dict
            fociData[languageNumber][speakerNumber][languageTerm] =␣
↪[]                         # then make it one, with its value an empty list

        if not(modGridCoord in␣
↪fociData[languageNumber][speakerNumber][languageTerm]): # if they provided␣
↪multiple A0 or J0 hits, only record 1
            fociData[languageNumber][speakerNumber][languageTerm].
↪append(modGridCoord)

    fileHandler.close()
    return fociData
```

[ ]:
```
def readChipData(chipDataFilePath):
    """
    Read all of chipDataFilePath into two dictionaries, one maps from row/
↪column code to WCS chip number,
        the other maps in the other direction.  Assumes data file follows WCS␣
↪format:
    chip number\tWCS grid row letter\tWCS grid column number\tWCS grid rowcol␣
↪abbreviation\n
```

```
    Parameters
    ----------
    chipDataFilePath : string
        The path (and filename, with the extension) to read the WCS-formatted␣
↪chip data from.


    Returns
    -------
    cnum : dictionary
            cnum[row/column abbreviation] = WCS chipnumber, thus cnum maps from␣
↪row/col designation to chip number

    cname : a dictionary
            cname[WCS chipnumber] = row letter, column number (a tuple), thus␣
↪cname maps from chip number to row/col designation


    Example Usage:
    --------------
    import wcsFunctions as wcs
    cnumDictionary, cnameDictionary = wcs.readChipData('./WCS-Data/chip.txt')

    """

    cnum = {}    # empty dict to look up number given row/column designation
    cname = {}   # empty dict to look up row/column designation given number
    fileHandler = open(chipDataFilePath, 'r')    # open file for reading
    for line in fileHandler:                     # for each line in the file
        lineElements = line.split()      # elements are denoted by white space
        chipnum = int(lineElements[0])   # 1st element is chip number, make it␣
↪an int
        RC = lineElements[3]             # 4th is row/column designation, leave␣
↪str (NB dictionaries don't exactly reverse each other)
        letter = lineElements[1]         # 2nd is the letter (row) designation
        number = str(lineElements[2])    # 3rd is the number (column)␣
↪designation, make string so we can combine it with letter as a tuple in␣
↪cname dict

        # cnum[rowcol] maps from row/column designation to chip number
        cnum[RC] = chipnum
        # cname[chipnum] maps from chip number to row/column designation (a␣
↪tuple)
        cname[chipnum] = letter,number
```

```
    fileHandler.close()

    return cnum,cname            # return both dicts
```

[ ]:
```
def readSpeakerData(speakerFilePath):
    #lANUGUAGE[SPEAKER NUMBER]
    """
    Parameters
    ----------
    speakerFilePath : string
        The path (and filename, with the extension) to read the WCS-formatted␣
↪speaker data from.

    Returns
    ------
    speakers : dictionary
               The keys are ints corresponding to the language IDs and the␣
↪values are lists of tuples, where
               each element of the list contains (AGE,GENDER) corresponding to␣
↪the speakers recorded for each language

    Example Usage:
    -------------
    >>> from pprint import pprint
    >>> speakerDictionary = readSpeakerData('./WCS_data_partial/spkr-lsas.txt')
    >>>  pprint(speakerDictionary)
    """
    speakers = {}                          #Initialize the dictionary
    f = open(speakerFilePath, 'r')         #Open the file containing the input data
    for line in f:                         #Iterate through each line
        content = line.split()             #split input data by whitespace
        language_ID = int(content[0])      #ID is the first element of row, cast as␣
↪int
        speaker_ID = int(content[1])
        speaker_age = content[2]           #Age is the third element of row
        speaker_gender = content[3]        #Gender is the fourth element of row
        if not (language_ID in speakers.keys()):
            speakers[language_ID] = {}
        if not (speaker_ID in speakers[language_ID].keys()):
            speakers[language_ID][speaker_ID] = []
        if not((speaker_age, speaker_gender) in␣
↪speakers[language_ID][speaker_ID]):
            speakers[language_ID][speaker_ID].append((speaker_age,␣
↪speaker_gender))
    return speakers
```

```
[ ]: def readClabData(clabFilePath):
         """
         Parameters
         ----------
         clabFilePath : string
             The path (and filename, with the extension) to read the WCS-formatted␣
     ↪clab data from.

         Returns
         -------
         clab : dictionary
                The keys are ints and the values are tuples (n1,n2,n3), representing␣
     ↪the clab coordinates

         Example Usage:
         -------------
         >>> clabDictionary = readClabData('./WCS_data_core/cnum-vhcm-lab-new.txt')
         >>> print(clabDictionary[141])
         (96.00, -.06,.06)

         """
         clab = {}                       #Initialize the dictionary
         f = open(clabFilePath,'r') #Open the file containing the input data
         for line in f:                  #Iterate through each line
             content = line.split() #split input data by whitespace
             ID = int(content[0])          #ID is the first element of row, cast as int
             n1,n2,n3 = content[-3],content[-2],content[-1] #coords are the last␣
     ↪three elements of row
             clab[ID] = (n1,n2,n3)  #Add ID, coordinate pairs to the dictionary
         return clab
```

```
[ ]: def plotValues(values, figx = 80, figy = 40):
         """Takes a numpy array or matrix and produces a color map that shows␣
     ↪variation in the values of the array/matrix."""
         """values: array or matrix of numbers
            figx: length of plot on the x axis, defaults to 10
            figy: length of plot on the y axis, defaults to 10"""
         #read in important information for reordering
         plt.rc(['ytick', 'xtick'], labelsize=50)
         cnumDictionary, cnameDictionary = readChipData('./WCS_data_core/chip.txt')
         #reorder the given values
         lst = [values[cnumDictionary['A0']-1], values[cnumDictionary['B0']-1],
            values[cnumDictionary['C0']-1], values[cnumDictionary['D0']-1],␣
     ↪values[cnumDictionary['E0']-1],
           values[cnumDictionary['F0']-1], values[cnumDictionary['G0']-1],␣
     ↪values[cnumDictionary['H0']-1],
           values[cnumDictionary['I0']-1], values[cnumDictionary['J0']-1]]
```

```
    for letter in list(string.ascii_uppercase[1:9]):
        for num in range(1, 41):
            lst.append(values[cnumDictionary[letter+str(num)]-1])
    values = np.array(lst)
    #plot
    ha = 'center'
    fig = plt.figure(figsize=(figx,figy))
    fig.suptitle('WCS chart', fontsize=80)
    gs = gridspec.GridSpec(2, 2, width_ratios=[1, 8], height_ratios=[1,1])
    ax1 = plt.subplot(gs[1])
    ax2 = plt.subplot(gs[0])
    core = values[10:].reshape((8, 40))
    ax1.imshow(core, extent = [0, len(core[0]),len(core), 0],␣
↪interpolation='none')
    labels = ["B", "C", "D", "E", "F", "G", "H", "I"]
    ax1.set_yticklabels(labels)
    ax2.imshow([[i] for i in values[:10]], extent = [0, 0.5, 0, 10],␣
↪interpolation='none')
    ax1.yaxis.set(ticks=np.arange(0.5, len(labels)), ticklabels=labels)
    ax2.yaxis.set(ticks=np.arange(0.5, len(["A"]+labels+["J"])),␣
↪ticklabels=(["A"]+labels+["J"])[::-1])
    ax1.xaxis.set(ticks=np.arange(0.5, 40), ticklabels=np.arange(1, 41))
```

[ ]:
```
def generate_random_values(ar):
    """Takes in an array of terms and returns a dictionary that maps terms to␣
↪random values between 0 and 1"""
    d = {}
    for term in ar:
        d[term] = random()
    return d
```

[ ]:
```
def map_array_to(ar, d):
    """Maps an array of terms into an array of random values given the␣
↪dictionary created by the above function"""
    return [d[i] for i in ar]
```

[ ]:
```
def readBKFociData(path):

    fociData = {}
    fileHandler = open(path,'r')

    for line in fileHandler:
        lineElements = line.split()

        languageNumber = int(lineElements[0])
        speakerNumber = int(lineElements[1])
```

```
        termNumber = int(lineElements[2])
        gridcoord = lineElements[3]

        if not (languageNumber in fociData.keys()):
            fociData[languageNumber] = {}
        if not (speakerNumber in fociData[languageNumber].keys()):
            fociData[languageNumber][speakerNumber] = {}
        if not (termNumber in fociData[languageNumber][speakerNumber].keys()):
            fociData[languageNumber][speakerNumber][termNumber] = []

        if not(gridcoord in␣
↪fociData[languageNumber][speakerNumber][termNumber]):
            fociData[languageNumber][speakerNumber][termNumber].
↪append(gridcoord)

    fileHandler.close()
    return fociData
```

[ ]:
```
def readBKDictData(path):

    dictData = {}
    fileHandler = open(path, 'r')

    for line in fileHandler:
        lineElements = line.split()

        try:
            languageNumber = int(lineElements[0])
            termNumber = int(lineElements[2])
        except:
            continue

        languageNumber = int(lineElements[0])
        abbrev = lineElements[1]
        termNumber = int(lineElements[2])
        termName = lineElements[3]

        if not (languageNumber in dictData.keys()):
            dictData[languageNumber] = {}
        if not (termNumber in dictData[languageNumber].keys()):
            dictData[languageNumber][termNumber] = {}
        if not (termName in dictData[languageNumber][termNumber].keys()):
            dictData[languageNumber][termNumber][abbrev] = {}

        if not(termName in dictData[languageNumber][termNumber][abbrev]):
            dictData[languageNumber][termNumber][abbrev] = termName
```

```
    fileHandler.close()
    return dictData
```

[ ]:

```

```