Project Illumination

By Jeremy and Songting Wu

**Main Idea**: Detecting if a light bulb is broken.

**Abstract**:

Large buildings such as museums and office buildings require lots of maintenance and upkeep. Within a single floor alone there can be thousands of ceiling lights so the use of autonomous agents will greatly increase the time efficiency of maintenance workers looking for broken lights. With our project, we try to prove through using the Kinect that there is an accurate method for evaluating if there are any broken ceiling lights within a given area.

**Setup**:

Our project makes use of two lamps and a room with the ceiling lights turned off. This is to provide a more controlled environment that cannot be ideally simulated through our current resources. We also used a turtlebot for its ease of handling as well as its low lying profile in relation to the ground. This is because the ground will be a better reflection of the lighting conditions in a particular area. The testing took place in room 3.414C near the double doors so that there was a mix of high natural lighting conditions as well as somewhat dark conditions when ceiling lights are off.

**Overall Structure of the Program:**

The main.cpp subscribes to the kinect color\_image topic, and publishes on the velocity topic. The image is processed in the main.cpp into a vector and sent by service to the writeUnit.py node. The writeUnit.py node will either write the data after compressing it into a list of 4 mean values, the degree it was taken at and the target class, or it will train a neural network and support vector machine then outputting them to file. The classificationUnit.py is a node that also has a service which takes in the realtime data from an actual run of the program, then classifies the input image and sending back the output to the main.cpp node. The main.cpp then decides overall out of all the images what action to take.

**Initial Design and Evolution:**

Data collection is done through the kinect. Multiple images are taken and then each pixel is converted to a luma value using the formula:

*Luma = 0.2126 R + 0.7152 G + 0.0722 B*

This places the luma value between 0-255 on the luma scale. The initial design for data processing was to do binary classification by histograms. However using the frequency of values did not seem it would be a good measure of overall luminance. We intended to implement our own Naive Bayes classifier, however we concluded that it would be more time efficient to use sci-kit and pybrains python libraries for our classifiers.We also considered using the HSI color space conversion to compare with the luma conversion, however this would something to be considered after verifying that the kinect is capable of solving the problem.

Our initial design for data gathering was to take continuous pictures 360 degrees around, however we were unable to successfully implement this. So now the agent will gather data by turning and taking a color image every 45 degrees. Also keeping track of its rotation is done by the number of images taken not by using the odometer’s yaw reading which also was an initial attempt. To streamline the writing to a text file of all the data we made another python node. Also the number of features from 307,200 per image was decided to be condensed to 4 mean values as a way to increase accuracy of the classifiers. We at first just had the agent rotate around, however this did not get the data we wanted. So we had to manually pan the agent in front of zero, one and two lamps in many different directions within the room.

Initial testing using a feed forward neural network (pybrains) with data that considered only if the ceiling lights in a room were on or off showed to be promising. However when applied to try and classify if a lamp light is on or off did not show any viable results. Therefore we decided to go with the support vector machine (sci-kit) which showed promising results from the initial testing.After we were sure that a binary classification of lamp lights on or off was accurately predicted by the SVM, we tried a multiclass classification. The intermediary case is 1 lamp on, with 2 lamps and 0 lamps being on the extremes. This is to detect more accurately the possible lighting conditions that can happen with a possible broken ceiling light. Sci-kit’s SVM also supports multi classification. With the current SVM which uses a rbf kernel, the results did not match up to when there was only binary classification. One reason for this could be the kernel and also the fact that this SVM is a one to one classifier. We tried different kernels such as polynomial,linear, and sigmoid, the most promising was the polynomial kernel. However we also tried a Linear SVM of one to many classification. The Linear SVM proved to be more accurate than the one-to-one polynomial SVM.

**Final Design Summary (what we managed to implement):**

The agent turns every 45 degrees and takes an image for a total of 8 images. It will stop rotating after 8 images. The classifier used is a one-to-many Linear Support Vector Machine from the sci-kit python library. The SVM model is persisted as an xml file format. There is 4 features which derive from splitting the image vector into 4 lists of 76,800 values and taking the 4 means. There are 3 classes where 0 denotes there were 0 lamps, 1 denotes 1 lamp and 2 denotes 2 lamps were in the image. There are 3 nodes, one does the image gathering and processing as well as movement/navigation, another does the training and writing of data, and the last purely does classification.

**Problems/Sources of Error:**

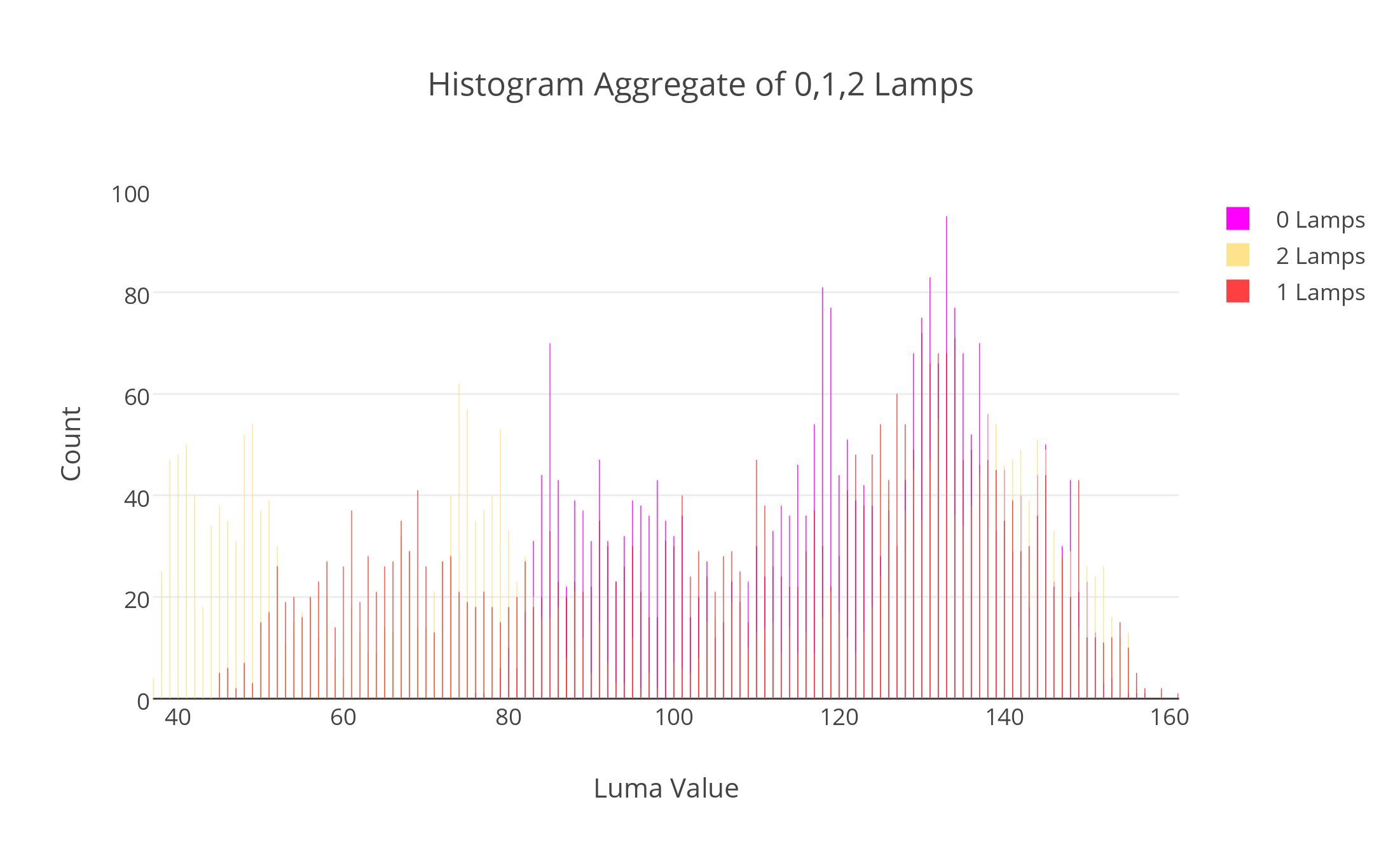
Varying lighting conditions played a significant factor in our testing. In the particular room we gathered data in, there was a lot of natural light that was coming in from the windows. This could have greatly skewed the luma values. Especially with when there was two lamps, the background light may have “merged” the light to make it seem like there was “one” lamp in the image. In addition the reflection of the light from the floor could have skewed our data. The weather also plays an important part in the varying light conditions. As we took data on different days in different times, the variability could have cause the classifier problems in regards to training the right values.

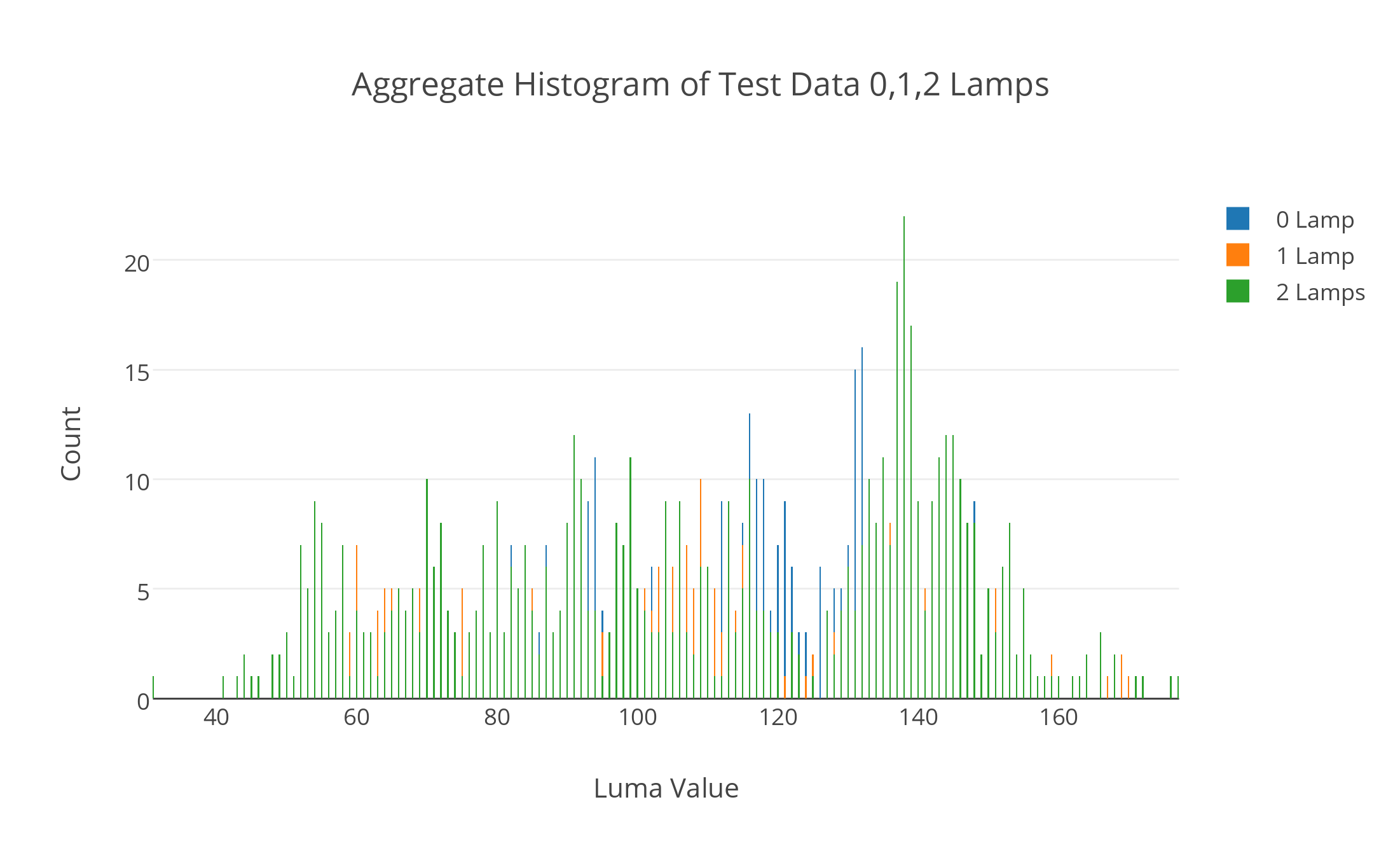
Some other problems we encountered was figuring out how much data we wanted and how we were going to obtain it. We could not get the robot to continuously take a 360 degree stream of photos which would have enhanced our data accuracy. In addition we had problems in automating the data so that when the data was written to the file it had the the right target associated with it. The turtlebot also has a part of the top black board where the laptop sits on, in view of the kinect camera, which can skew the data. People in the same room wearing dark were also taken with the image.

Another source of problems came from the large amount of parameters that we could have have picked such as classifiers, data location, etc. There were many other classifiers that we could have tried and that could have been more effective in solving our problem. Within these classifiers there are even more parameters that can be modified to change the effectiveness of the classifier. The data source is one of the most important factors. In a more controlled environment our training set could have possibly been more accurate. In addition the images had to be taken manually with rearrangement of the lamps in different areas which accosted a lot of time. With more data, it is possible to reduce the variability and increase the robustness of the SVM.

**Data**:

Each image is split into 4 parts and the mean is taken of each part. The list that is written to the file for each image is as follows:

 *[mean1,mean2,mean3,mean4,degree,target\_class]*

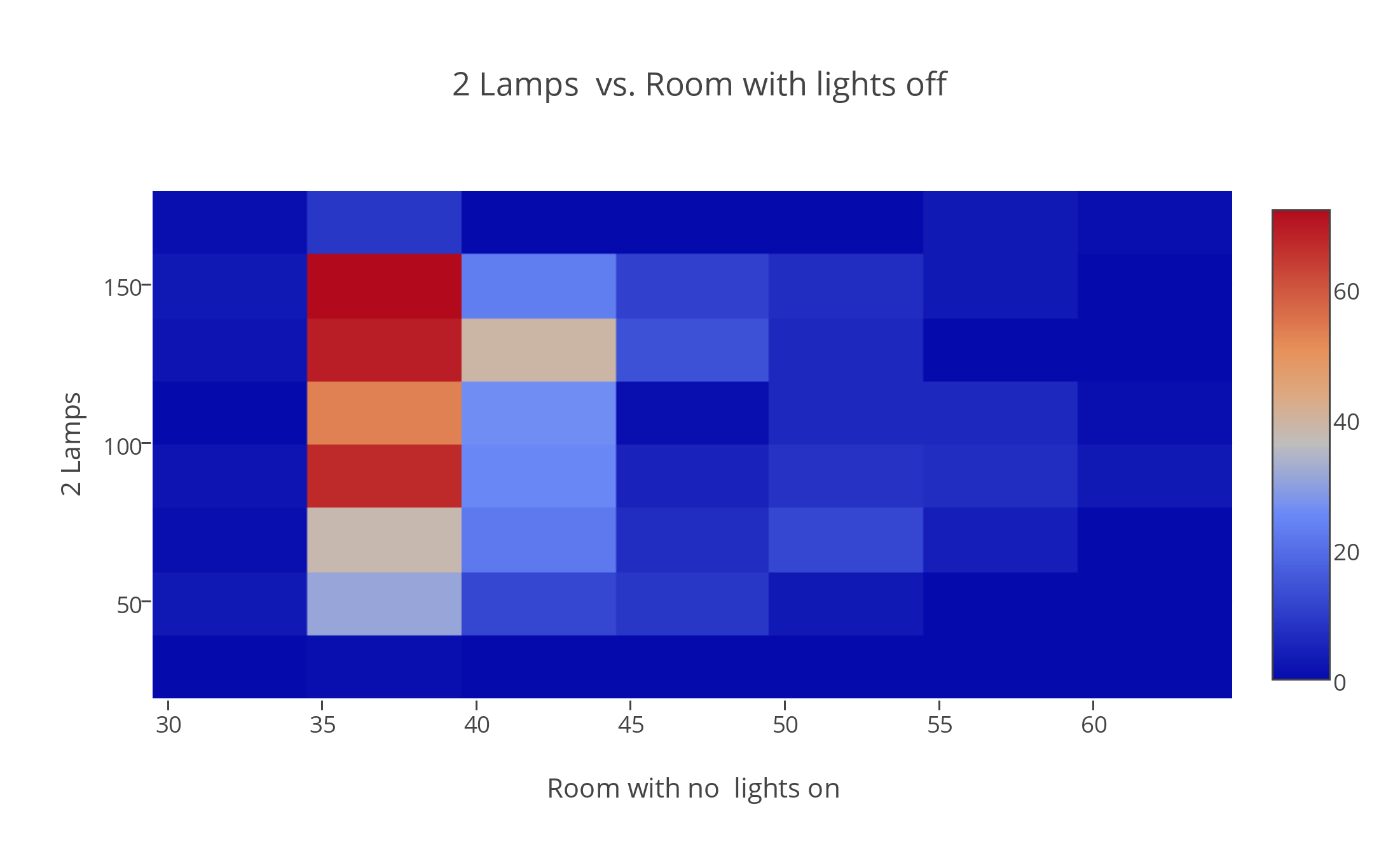
The degree does not contribute to the algorithm or decision as of this version. Data processing and gathering is explained above in the evolution of the overall design and the final design sections. The disparity between the two lamps luma value counts in the histogram stems from our data gathering methods which may not have been even in terms of matching lighting conditions from the initial training set to the test data.

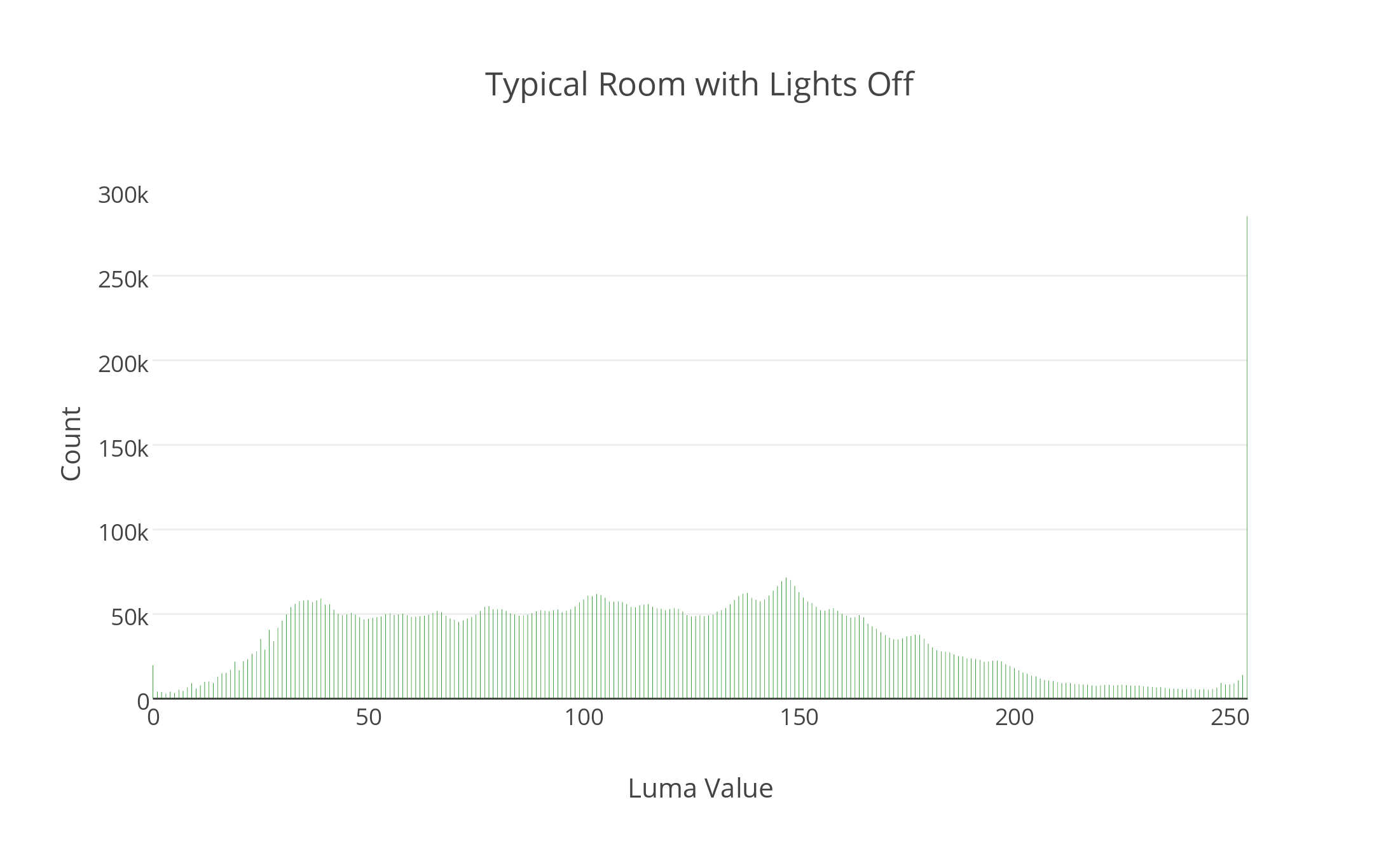
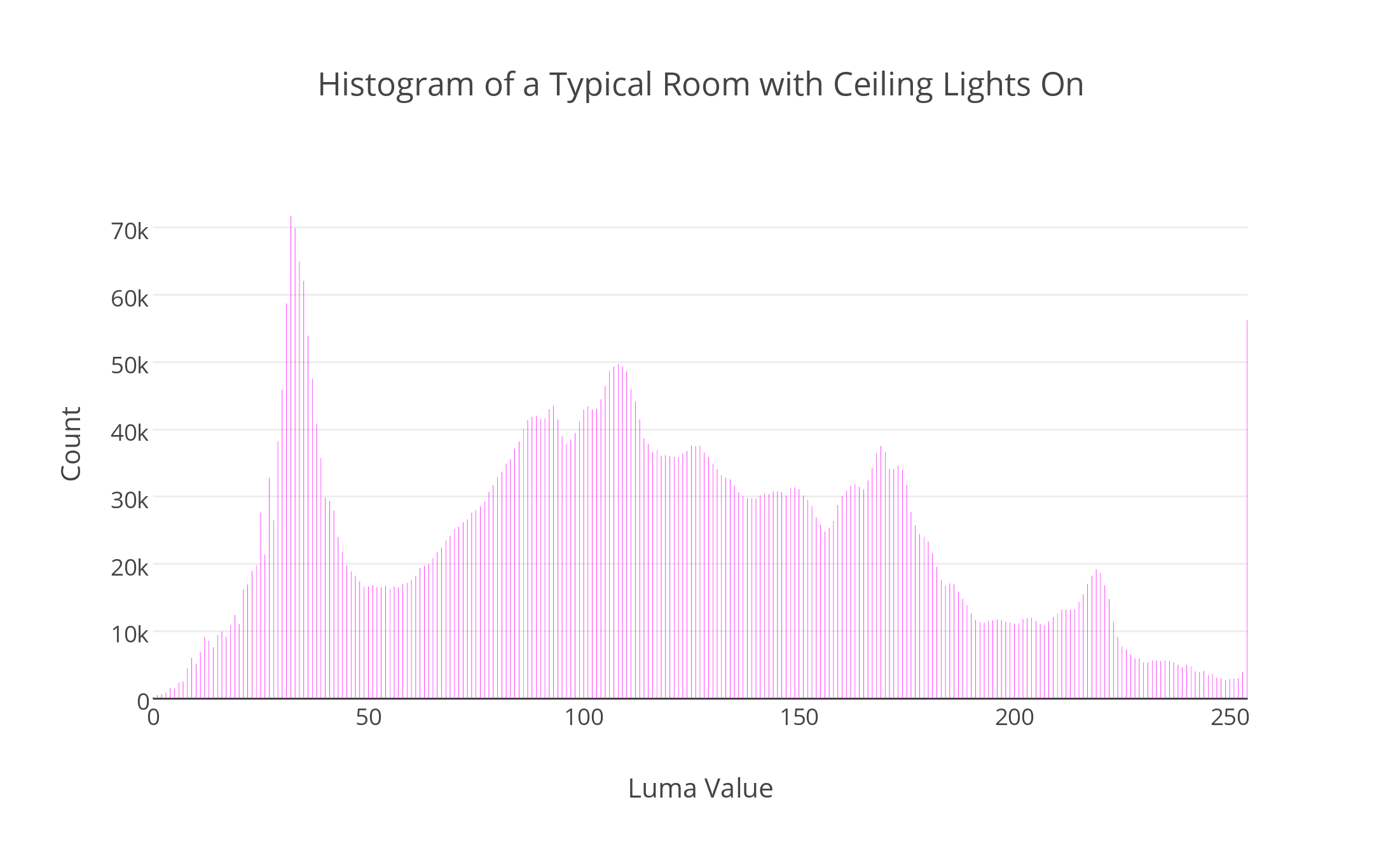
**Training Data Set:**

*Zero lamps has many high luma values because of external lighting influences. One lamps has the majority of its luma values higher than 100. Two lamps have significant amounts of luma values less than 100 possibly because of the higher contrast between light and dark with two lamps on and external light conditions.*

***Test Data:***

*The shape of the test data follows the shape of the training set histogram. Here the two lamps has a lot more high luma values. The one lamp is evenly distributed with the most in the middle luma values. The irregularity of the zero lamp is from the external lighting conditions.*





The first graph above is a 2d histogram that compares the images when there is 2 lamps against for when there is a room with no lamps and lights on. Here there is a clear spot of high luma values. This implies that our classifier should be able to detect the extreme changes in luma values. The next two histogram graphs are to illustrate a contrast between when a room has all lights on and no lights on. This is an important difference to our problem even though what we are working with are lamps. We can see that a typical room with all lights on and all lights off, has a fairly even distribution of luma values with very little abnormal spikes in the data.

**Results**:

Depending on the lighting condition, the classifier has at worst about 40% accuracy. The linear support vector machine has the most inaccuracy with the two lamp images. There is almost a 100% accuracy with no lamps, with some outliers due to external lighting conditions. For a single lamp, it can mostly classify this type of image accurately.With external lighting conditions, it seems to merge the luma values together when there are two lamps, so it incorrectly classifies the image as an image of 1 lamp. With more data and accountability of the variable external lighting conditions, i think the result mispredictions can be greatly reduced.

**My Contribution:**

Programmed rotating agent, classification, training models, luma algorithm,data processing and gathering for training and testing. Testing classification model and processed data. Designing the final program structure. Research into which classifier to use.

**Unimplemented/Future Features:**

We could have possibly tested more classifiers to enhance results such as Naive Bayes, decision trees and more advanced neural networks. We were not able to implement any navigation of any sort. A future feature could be navigation and a recording of the location on the map of where the possible broken light is. Using a one class SVM to detect anomalies(possible broken light) as the agent is on the patrol route, and then spinning in a circle to confirm the possibility. Another possibility is to first using a neural network for initial classification, then using an SVM on the subsequent vector of outputs. Gathering more data in more locations at different light variations could also be put in as an on going “learning” type of feature on a patrol route. A feature that accounts for variable light conditions based on the current weather, month and UV index.

**Links to some resources used:**

First Histogram: https://plot.ly/~phalax4/66/histogram-aggregate-of-012-lamps/

Second Histogram: https://plot.ly/~phalax4/66/aggregate-histogram-of-test-data-012-lamps/

Sci-kit SVM: <http://scikit-learn.org/stable/modules/svm.html>

Pybrain neural network: <http://pybrain.org/docs/tutorial/fnn.html>

\*Group only had 2 people, so no report on colleagues was included.

\*Histograms made through plotly.