A Proposed System for Efficient Edge Detection and 3D Planar Information Extraction from a 2D Image

Pratham Gandhi (pratham_gandhi@horacemann.org),
Samuel Schuur (samuel_schuur@horacemann.org)

Horace Mann School, Bronx, New York

Abstract

In this paper, the development and application of a proposed system to computationally inexpensively determine a three dimensional model of a space from only a single image is presented, using applied machine learning and simple hardware components. Using feature classification of the relevant objects in the original scene, in collection with a physical grid of laser dots refracted across the scene, the system is able to quickly judge the nature of the object, its position in relation to other objects around it, and distance of those objects from the camera, traits which are commonly compromised on comparable approaches. The system is x times more efficient on average than current industry leading approaches, and has potential applications in various fields requiring rapidly updating spatial awareness and environment generation, including autonomous vehicles, defense, fire-safety and architecture.

August 2018

Contents

1. Introduction

The current most popular approach to creating a three dimensional informational estimate from a two dimensional image is triangulation, or binocular, stereo vision approach. This approach take several images of the same scene from different locations and angles in order to generate a spatial awareness. Computer stereo vision, however, is difficult to implement in practical applications in our increasingly evolving digital world, due to the limiting factor that they require multiple images, the acquisition of which is sometimes impractical. Additionally, they can sometimes be quite slow, in that they need to first identify common features in the different photos, find out the positional relation of those features, and then only begin to start seeing the greater picture of the full space. The purpose of Constellation was to solve both of these problems and create a flexible system which can adapt and continue to be functional in many environments.

We chose to approach this problem first through the somewhat traditional technique of using artificial neural networks, mathematical functions modeled after nature which specialize in taking in large amounts of input to produce outputs, to identify features in a two dimensional image. Neural networks are are perfect for navigating the fuzzy problems of feature extraction from images. Where our system differs is in its use of a grid of refracted laser points into the scene to judge the distances and orientations of various objects in relation to the camera. This allows superior environment generation in a shorter amount of time, due to the omission of the multiple-image stereo aspect. This approach and its implemented system was named Constellation for the star-like appearance of the laser dot grid it relies on.

2. Background

2.1. Binocular Computer Stereo Vision

In order to perceive depth the human eye uses many visual cues, both monocular and binocular. Of these visual cues, stereopsis, the perception of depth resulting from the brain comparing the differences in images produced by both eyes, is in many ways one of the easiest depth perceptions methods to mimic through software, as it is the least reliant on external sources of stimuli and human experience. This is what Stereo Computer Vision attempts to do; building depth maps by comparing scenes from multiple vantage points. More specifically Binocular Computer Stereo Vision does this by comparing the images from two cameras and by extracting and comparing common features from both images it builds a depth map of the scene in question.

2.1.1. Removing Distortion from Images

Modern cameras introduce two major kinds of distortion to their images: barrel distortion and tangential distortion. Barrel distortion, a form of radial distortion, is introduced due to the curvature of lenses and, as such, is more prominent in wider angle lenses. Barrel distortion causes objects toward the edges of the frame to appear smaller than objects in the middle of the frame, therefore creating a situation where distances in relation to the size of an object are nonlinear. Tangential distortion on the other hand is introduced due to the lens of a given camera not being properly aligned with the imaging sensor, causing some

areas of the image to look larger than others, once again causing the relationship between size and distance to be nonlinear. However, since binocular computer vision requires that the distance-size relationship must remain consistent throughout an image to be able to accurately compare images from different positions, these distortions must be corrected for, in order to effectively allow our camera to mimic the perfect pinhole camera.

In order to correct for radial and tangential distortion, many models and formulas have been proposed and developed, but for the purpose of this paper we will be using formulas derived from the Brown-Conrady model. The formulas for correcting radial distortion read as follows, where r is the distance from the distortion center to a distorted point:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

So a pixel (x, y) in the distorted image will be remapped to ($x_{corrected}$, $y_{corrected}$). The same is true in the following formulas for correcting tangential distortion:

$$x_{corrected} = x + (2p_1 xy + p_2(r^2 + 1x^2))$$
$$y_{corrected} = y + (p_1(r^2 + 2y^2) + 2p_2 xy)$$

In above formulas are five distortion parameters that must be seperately experimentally determined in order to correct the image: $k_1, k_2, p_1, p_2,$ and k_3. In addition to these, a given camera's "camera matrix" must be determined, which is a set of intrinsic camera parameters such as focal length and optical centers. This information is determined experimentally, on a camera to camera basis, by taking multiple images of a grid of objects of known size, shape, and relation to each other and tuning these parameters until the corrected grid matches the actual grid.

2.1.2. Finding the epiline of an Image

2.1.3. Feature detecting and matching

2.1.4. Creating a depth map

2.2. Structure-From-Motion Pipelines

Structure-from-motion (SFM) pipelines operate on a simple premise; finding common anchor points among several images of the same subject, and using triangulation to estimate where those points lie in relation to the camera.

2.2.1. Shortcomings

One of the major issues in using SFM pipelines in practical applications where rapidly updating three dimensional estimations are required is that the models produced by SFM pipelines are lacking a scale factor, that is, there is no sense accurate of scale or distance in the model. This is one of the issues Constellation aims to address.

2.3. Neural Networks

An Artificial Neural Network (ANN) is a computational model inspired by biological networks in the human brain which process large amounts of information.

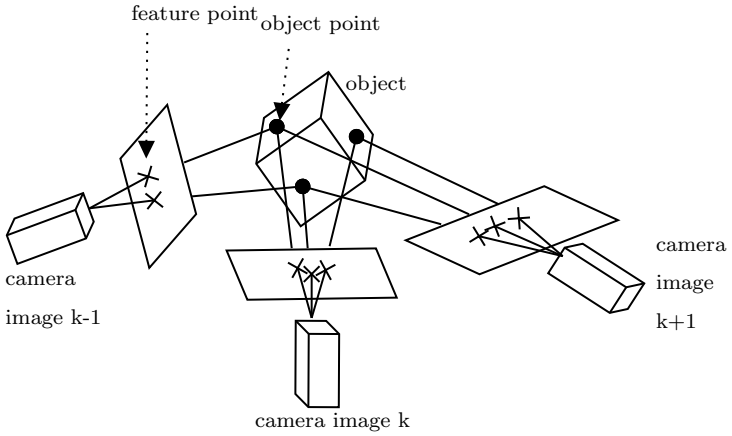

Figure 1: SFM Camera Setup Diagram

2.3.1. A Single Neuron

The basic unit if a neural network is a neuron. A neuron is essentially a node in the graph which represents the neural network. Each neuron in an ANN, similar to a biological neuron, receives input from other neurons. Each input has an associated weight W, which is adjusted based on the importance of its corresponding input in the large picture of all the inputs to a neuron. Once it has acquired its inputs, a neuron will apply an activation function f, as shown below, to the weighted sum of its inputs to produce an output.

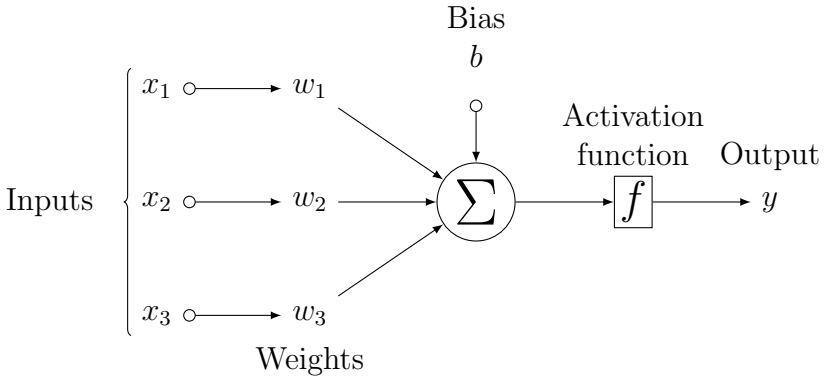

Figure 2: Artificial Neuron Diagram

The function f is a non-linear activation function. The role of the activation function is to create non-linearity in the output of a neuron, which is key because most real-world data in non linear and the purpose of neurons is to learn how to represent that data. Every activation function takes a single input and performs a mathematical operation on it in order to produce the final output. Common activation functions include:

- Sigmoid: takes a value and squashes it to range 0-1. $\sigma(x) = \frac{1}{1+e^{-x}}$
- tahn: takes a value and squashes it to range [-1,1]. $\tanh(x) = 2\sigma(2x) - 1$

- ReLU: stands for Rectified Linear Unit. Takes a value and replaces all negative values with 0. $f(x) = \max(0, x)$

Constellation chooses to use sigmoid as the primary activation function due to its gradient nature, fixed range, and the fact that it is easier to differentiate values closer to one asympote or the other.

$$\text{input to activation} = (\sum_{i=1}^{n}(x_i \times w_i)) + b$$

$$\text{activation } f(x) = \frac{1}{1 + e^{-x}}$$

$$\text{output} = \frac{1}{1 + e^{-((\sum_{i=1}^{n}(x_i \times w_i)) + b))}}$$

2.3.2. Feedforward Networks

The feedforward neural network is the simplest and most widely applied type of ANN devised. Feedforward nets contain several neurons sorted into layers. Nodes from adjacent layers have connections, represented as edges on the graph, between them. Every connection, as discussed earlier, has a weight associated with it.

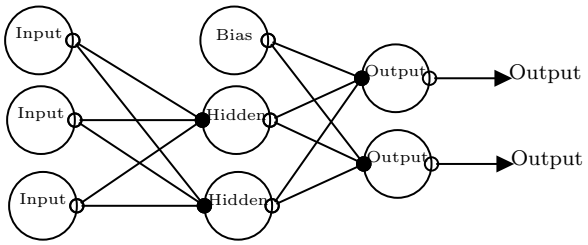

Figure 3: An example of a feedforward neural network

(a) Note than an empty circle on an edge denotes a source and a filled circle on an edge denotes a sink.

A feedforward network can consist of three types of nodes:

1. Input Nodes - Provide information from the outside world to the network, the set of which is called the "Input Layer". None of these nodes perform any computation.
2. Hidden Nodes - Have no direct connection with outside world and only perform computations and transfer information from the input nodes to the output nodes. A set of these nodes in called a "Hidden Layer". Feedforward nets can have any number of hidden layers.
3. Output Nodes - Collectively called the "Output Layer", these nodes perform computations and also transfer information from the network to the outside world

In a feedforward network, information only moves forward, through the input nodes, hidden nodes, and then finally on to output nodes. There is no cycling or looping back through the network. Below is an abridged psuedocode version of the feedforward algorithm implemented in Constellation:

2.3.3. Training Using Back-Propagation

It is possible to train a Multi-Layer Perceptron (MLP) Neural Network, a network which has at least one hidden layer, using a technique called back-propagation. Back-Propagation is a type of training approach which is called supervised learning, which means that the network is learning from labeled training data, where there is data is fed in, accompanied by classifications for each set of data which the network is told are correct. Each connection between nodes has a weight, which, as mentioned earlier, dictates the relevance of a specific input in the greater scope of all the inputs to a given neuron. The goal of learning is to assign correct weights to these edges in order to provide the most accurate classifications.

All weights are randomly assigned to begin with. Then, one by one, every training set is passed through the neural network, and, depending on how vast the difference desired and actual ouput of each layer, the error is propagated back to the previous layer. The error is noted and each weight is adjusted accordingly. This process is repeated using the provided data set until the network is outputting acceptably close classifications to the labeled classifications. Once this level is reached, the network can be passed previously unseen data and can use its trained weights to produce a classification.

Below is an abridged psuedocode version of the backpropagation algorithm implemented in Constellation:

3. Approach

3.1. Edge Detection

3.2. Edge Formulation

3.2.1. Determining Lines of Edges

Before the discussion of how to determine which points a line consists of, it is important to discuss an efficient model for formulating a representation of a line which is represented by a set points. Consider constants α (alpha) and β (beta) such that a following line exits:

$$y_i = \beta x_i + \alpha + \epsilon_i$$

where y_i is the y value of point i, x_i is the x value of i, and ϵ_i is the optimally small error term representing the imperfectness of the model to include that point. The problem now presents a textbook optimization problem; finding values of α and β which produce the minimum ϵ. Immediately, gradient descent comes to mind, however, a different approach to this optimization problem was taken, for reasons explained later. In order to find the alpha and beta values, the error term must be first defined. The error should increase as points are further away from the line which is created. This logic would lend itself well to a representation as simply the difference between the y_i value of the point i and the y_l value of the line l at x_i. If this approach is taken, however, when the error term ϵ is calculated, if one x_1 is very positive and another x_2 is very negative, meaning that (x_1, y_1) lies far above the line and (x_2, y_2) lies far below the line, then their errors might cancel out. So, instead, the squared error ϵ of each point is considered, providing a total error E, represented as:

$$\epsilon(y_i, y_l) = (y_i - y_l)^2$$

$$E(m,b) = \sum_{i=1}^{n} \epsilon(y_i, mx_i + b)$$

Explicitly writing out this function composition we get the following, which can be expanded as shown:

$$E(m,b) = \sum_{i=1}^{n} (y_i - mx_i - b)^2$$

$$E(m,b) = \sum_{i=1}^{n} (y_i^2 + m^2x_i^2 + b^2 - 2mx_iy_i - 2by_i + 2bmx_i)$$

$$E(m,b) = \sum_{i=1}^{n} y_i^2 + \sum_{i=1}^{n} m^2x_i^2 + \sum_{i=1}^{n} b^2 - \sum_{i=1}^{n} 2mx_iy_i - \sum_{i=1}^{n} 2by_i + \sum_{i=1}^{n} 2bmx_i$$

For ease of readability the six terms are somewhat replaced [by $X = \sum x_i, Y = \sum y_i, A = \sum x_i^2, B = \sum x_iy_i, C = \sum y_i^2, N =$ number of points] to produce:

$$E(m,b) = C + m^2A + b^2N - 2mB - 2bY + 2bmX$$

In order to find the values of m and b which minimize the error E, the relationships between those constants and E must be found. This is can simply be done by taking the derivates of E with respect to m and b, as shown below. And since it is known that if there is an absolute minimum value of E, then it will occur at $\frac{\partial E}{\partial m} = 0$ and $\frac{\partial E}{\partial b} = 0$, values which make those relationships as close to 0 as possible are to be found.

$$\frac{\partial E}{\partial m} = 2mA - 2B + 2bX = 0 \quad (1)$$

$$\frac{\partial E}{\partial b} = 2bN - 2Y + 2mX = 0 \quad (2)$$

Now these equations can be solved by isolating for b in equation (2), and substituting that expression for all b into equation (1) to solve for m, producing (note - $\bar{x}$ means the average of all the x's):

$$b = \frac{Y - mX}{N} = \frac{\sum y_i - m\sum x_i}{N}$$

$$m = \frac{NB - XY}{NA - X^2} = \frac{N\sum x_iy_i - \sum x_i \sum y_i}{N\sum x_i^2 - (\sum x_i)^2} = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x - \bar{x})^2}$$

The reason this approach was taken as opposed to a gradient descent one is due to maximum likelihood estimation. A sample distribution of data $v_1, \ldots, v_n$ which depends on an unknown parameter θ can be expressed as $p(v_1, \ldots, v_n \mid \theta)$. If theta is unkown, this could conversely be represented as the likelihood of θ being true given the data; $L(\theta \mid v_1, \ldots v_n)$. Given this, it can be seen that the most likely to be true theta is the θ which maximizes the likelihood function. As seen ealier, a linear regression model with the least E should have a mean ϵ of 0, because it best distributes points above and below the line of best fit equally, with most points within a standard deviation of σ away from the line. If this is the case,

then it can be seen that the likelihood of α and β being the best based on seeing a point (x_i, y_i) is:

$$L(\alpha, \beta \mid x_i, y_i, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(y_i - \alpha - \beta x_i)^2 / 2\sigma^2}$$

While gradient descent is more efficient than this approach, gradient descent provides α and β values which do not maximize the likelihood function, for gradient descent only moves opposite the gradient till a given convergence threshold which must be greater than 0, thus never actually finding the absolute minimum, unlike this approach which does computationally find the absolute minimum ϵ by finding exact values of α and β.

As discussed later, Constellation's edge detection algorithm operates based on the angle of change between the line of best fit of a set $S = \{(x_1, y_1), \ldots, (x_i, y_i)\}$ and set $S' = \{(x_1, y_1), \ldots, (x_{i+1}, y_{i+1})\}$. The angle of change, however, can only be measured between two lines which share an endpoint, so the current model must be adapted to represent only a line which begins at a given point. Logically, it makes sense to choose the first point (x_1, y_1) considered when building the line to be the common anchor point, as it will always be the first point in any line built on S or any S'. From there, the dataset is transformed to form $S^T = \{(u_i, v_i) = (x_i, y_i) - (x_1, y_1)\}$ $\forall$ points i. So now the model to be fit is no longer $y_i = \beta x_i + \alpha + \epsilon_i$, but instead $y_i - y_1 = \beta(x_i - x_0) + \epsilon_i$

3.2.2. Determining Which Points Lie in Line

3.3. Distance Estimation

Constellation is unique from other systems in its approach of judging the distance of various objects within its captured scene from the camera. Most other approaches might use computationally intensive math and thus have very slow running times. Constellation, instead, uses efficient neural network driven image classification, in combination with simple statistical analysis to perform the same task. The underlying principle which allows this approach to function is that, when refracted through a particularly shaped crystal, the light from a single laser is scattered in such a manner that, as it moves further in distance from its source, it strays further in one direction or the other from its origin. Thus, when it is intercepted by another object at a certain distance, the amount the light at that distance has moved away from the position it would be at if it was intercepted directly in front of the source should be somewhat representative of the numerical distance travelled by that light, and additionally, the distance between individual points of refracted light should also be indicative of the distance from the camera, as each refracted beam of light would stray further from its neighbors.

Constellation's object detection and classification subsystem provides the image coordinates of the center of the object it detects. This is the foundation point used for judging the distance within the image between the laser dots refracted across the figure. In order to find the distance of the object from the camera, the distance between the dots projected onto the object has to be found. Constellation does this by obtaining a sorted list of all the points in the image, a process which in the worst case is $O(n \log n)$, due to the choice to employ a modified Timsort algorithm, but due to the nature of Constellation's object detection implementation, trends nearer to $O(n)$. Following processing and sorting the list of all the dots found in the image, Constellation proceeds to conduct a modified binary search

through the list in an effort to find the laser dot closest to the center of the object, taking $O(\log n)$ in both the worst and average cases. Once that has been acquired, the system takes an iterative approach to finding as many possible non-distorted dots on the surface of the object, calculating the euclidean distance between the dot closest to the center of the object and the 4 nearest dots, and checking if they are within the shape of the object determined during the process of training Constellation to recognize the object. If all 4 of the closest dots are within the shape of the object, Constellation proceeds to check the next 4 closest dots, and so on. The entirety of these processes assembles an array of all the representative dots on the visible surface of the object, and that array is used to find the average euclidean distance between each all the points on the surface of the object.

As mentioned earlier, the distance between the refracted dots projected onto an object is theoretically representative of the distance of that object from the camera which captured the image. Several approaches were considered when determining the most efficient and accurate method to extract information about an object's distance from the camera, including finding an as-accurate-as-possible multiple-degree polynomial modeling of the distance between dots vs. the distance from the camera, linear modeling, training a neural network to predict the distance, and a k-nearest neighbors (KNN) approach. Firstly, we considered multiple degree polynomials to mathematically model the correspondence between average distance between dots on the surface of an object and the distance of that object, however, polynomials over a degree of 2 often result in over-fitting, or creating a model which too closely or too accurately represents the training set. This might seem like an advantage of this approach, however, over-fitted models never generalize well to datasets beyond the training set, and thus, as shown in figure 5, actually lose accuracy. So, the rational next step after multiple degree polynomials is linear modeling. A linear function representing the relationship would probably be fairly accurate and abstract well enough to general datasets, however, linear functions exhibit another phenomenon called "underfitting", which means that the model is neither very accurate with the training set nor with general sets. Having ruled out functional modeling all together, the focus was shifted towards models grounded in statistical science and machine learning. Having already heavily implemented them in other parts of Constellation, neural networks were considered for the tasks, but it was determined that they were uneccesary due to their relatively extensive training data necessity and comparatively slow classification speed. Another machine learned based model, SVMs, were considered, but SVMs can only really produce one decision boundary in a dataset well, and therefore cannot be abstracted to classify more than 2 possibilities or labels easily (Weston, Jason, and Chris Watkins, 1998). All of these reasonings led to the choice of a KNN model, a model which finds the k nearest data points to the one which is to be classified, and essentially lets them "vote" on the classification of the data point based on their own classifications, therefore choosing the majority classification of the points near the target point. The graph in figure 5 graphically illustrates the advantages and disadvantages of each of the viable considered models in this particular task on a sample dataset.

A KNN model's accuracy grows near-exponentially with the addition of more labeled training data, as its training data is essentially the entire model. Constellation currently operates on limited range, due in part to the limited visibility of the laser dots as they move farther from the source, but also due to the limitation of available training data. As of now, a relatively large training set with over x data points has been gathered, and is effectively used

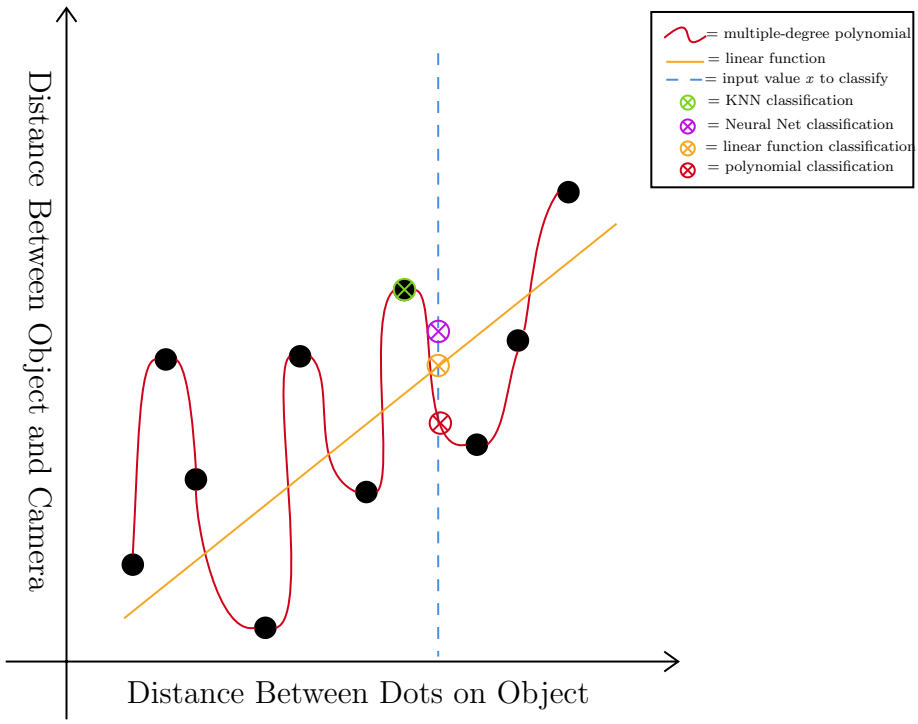

Figure 5: Different Classification Approaches on Sample Set

to predict distance within a range of x. As described earlier, the model predicts the distance of an object depending on the spread of the laser dots on the object's surface by finding the values of the 6 nearest data points and letting them vote on an appropriate distance for the object. Assume there is a defined function `majority_vote(labels)` which returns the majority vote of k labels and, in the case of ties, recursively reduces k until there are no more ties. Given that, the following is an abridged psuedocode segment of Constellation's implementation of a KNN model illustrating the logic behind the model:

Algorithm 1 KNN Classification Algorithm

```
1: procedure KNN_CLASSIFY(K, LABELED POINTS, NEW POINT):
2:     by distance ← sorted labeled points by distance from new point
3:     create empty list of k nearest
4:     loop for first k points in by distance:
5:         k nearest ← k nearest + point
6:     classification ← majority_vote(k nearest)
7:     return classification
```

Appendix A. Open Source Code and Library

The ConstellationCV project was created and built as a research project to explore new approaches to an old problem. As such, most of the code and documents produced in the research project have been open-sourced and are accessible by anyone. All of the production code has been formatted into an open library structure to allow for easy use of it, as well as to encourage the modification and adaptation of the principles for specific uses. All relevant pieces of code have been well documented and well commented, allowing for increased readability and understandibility. Additionally, all directories contain a `.txt` file which contains information about that directory for easy use. The authors of this paper welcome any contributions to or thoughts about the project, and would additionally encourage those interested to fork the repository on GitHub and attempt improve it in any way possible. All of the code can be found at:

https://github.com/ConstellationCV/Constellation

Additionally, for those interested, all documents produced in the process of developing Constellation, including research logs and all the source files for this paper, have been open sourced and can be found at:

https://github.com/ConstellationCV/Documents

Appendix B. Refrences

[1] M. Xu, X. Liu, Y. Liu, F. X. Lin, Accelerating convolutional neural networks for continuous mobile vision via cache reuse, arXiv preprint arXiv:1712.01670 (2017).

[2] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, R. Koch, Visual modeling with a hand-held camera, International Journal of Computer Vision 59 (2004) 207–232.

[3] J. Li, M.-T. Luong, D. Jurafsky, E. Hovy, When are tree structures necessary for deep learning of representations?, arXiv preprint arXiv:1503.00185 (2015).

[4] I. Tsochantaridis, T. Joachims, T. Hofmann, Y. Altun, Large margin methods for structured and interdependent output variables, Journal of machine learning research 6 (2005) 1453–1484.

[5] J. Weston, C. Watkins, Multi-class support vector machines, Technical Report, Citeseer, 1998.

[6] K. Crammer, Y. Singer, On the algorithmic implementation of multiclass kernel-based vector machines, Journal of machine learning research 2 (2001) 265–292.

[7] M. Potmesil, Generating octree models of 3d objects from their silhouettes in a sequence of images, Computer Vision, Graphics, and Image Processing 40 (1987) 1–29.

Appendix C. Aknowledgments