ConstellationCV: Obtaining 3D Information from 2D Images Efficiently using Feature Extraction and Laser Dot Projection

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

Bronx, New York

Abstract

In this paper, the development and application of a 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, and architecture.

August 2018

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

2.2. Structure-From-Motion Pipelines

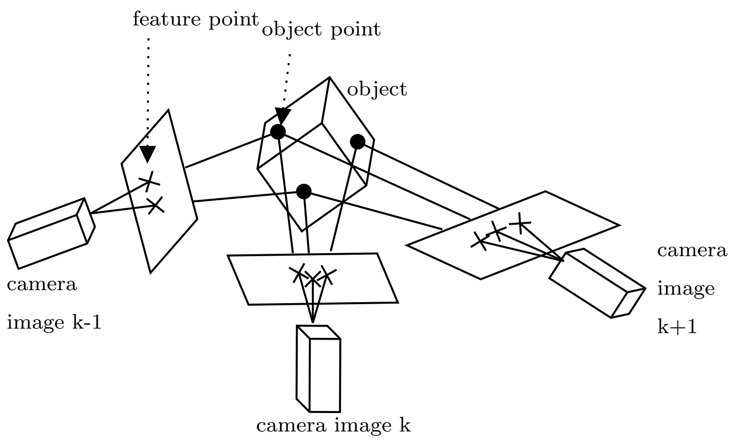

Figure 1: SFM Camera Setup Diagram

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.

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.

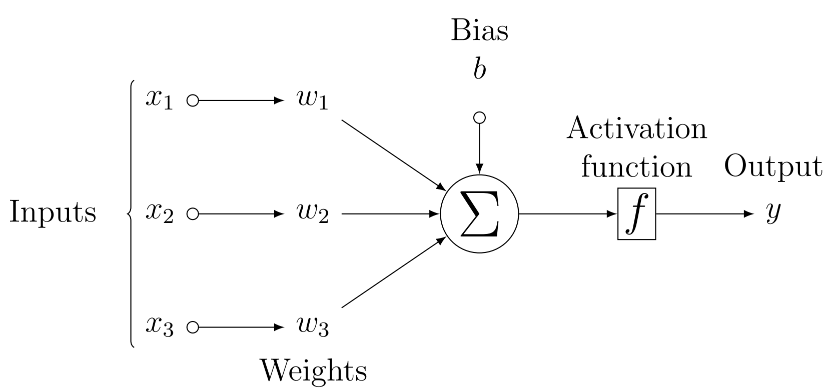

Figure 2: Artificial Neuron Diagram

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

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

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

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.

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.

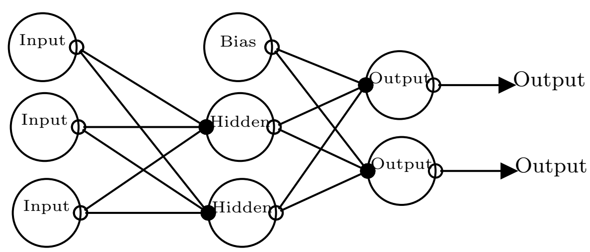

Figure 3: An example of a feedforward neural network

(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:

Algorithm 1 Feed-Forward Algorithm

procedure FEEDFORWARD:
    create and empty list of all layers' output
    loop for each layer in network:
        create empty list of this layer's outputs
        biased input $\leftarrow \sum_{i=1}^{n}(x_i \times w_i) + b$
        loop for each neuron in layer :
            neuron output $\leftarrow \sigma$(biased input)
            layer's outputs $\leftarrow$ layer's outputs + neuron output
        all outputs $\leftarrow$ all outputs + layer's output
    return all outputs

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:

Algorithm 2 Backpropagation Algorithm

procedure BACKPROPAGATE(NETWORK, INPUT VECTOR, TARGETS):
    hidden outputs, outputs $\leftarrow$ feed_forward(network, input vector)
    loop for each output:
        delta output $\leftarrow \dot{\sigma}$(output) $*$ (output $-$ target)
    loop for each output:
        adjust weights of the ith neuron
    loop for each hidden output: // hOutput = hidden output
        hidden delta $\leftarrow \dot{\sigma}$(hOutput) $*$ (hOutput $-$ next adjusted input)
    loop for each hidden output:
        adjust weights of the ith hidden neuron

3. Approach

3.1. Feature Detection and Classification

The central problem of detecting various features within an image is one of classification. There exist several different approaches or algorithms for classification, including decision trees, naive, bayes, and KNN. However, only two classification models stood out for our particular application of object recognition; support vector machines (SVMs), and neural networks. SVMs operate by determining an optimal classification line, which seperates training data of two different types into two distinct sections, and performs classification by determining which side of that line a set of input data falls. While SVMs have a higher accuracy in general, and are more tollerant to redundant and irrelevant attributes, they require on average three times more training samples to accurately classify features than neural networks, and still perform evenly with neural networks with regards to the speed of classification, speed of learning, and tolerance to highly interdependent attributes. This system's applications require rapidly updating environments, which thus require rapid computations and rapid training. These reasons make SVMs impractical for use in Constellation, which means the best choice for Constellation's object detection and classification approach was neural networks, which could easily be pretrained and then adjusted during operation.

- Detecting and Differentiating Objects and Laser Dots Using Efficient Universal Algorithms

3.2. 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, as shown in figure 5, thus 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 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.

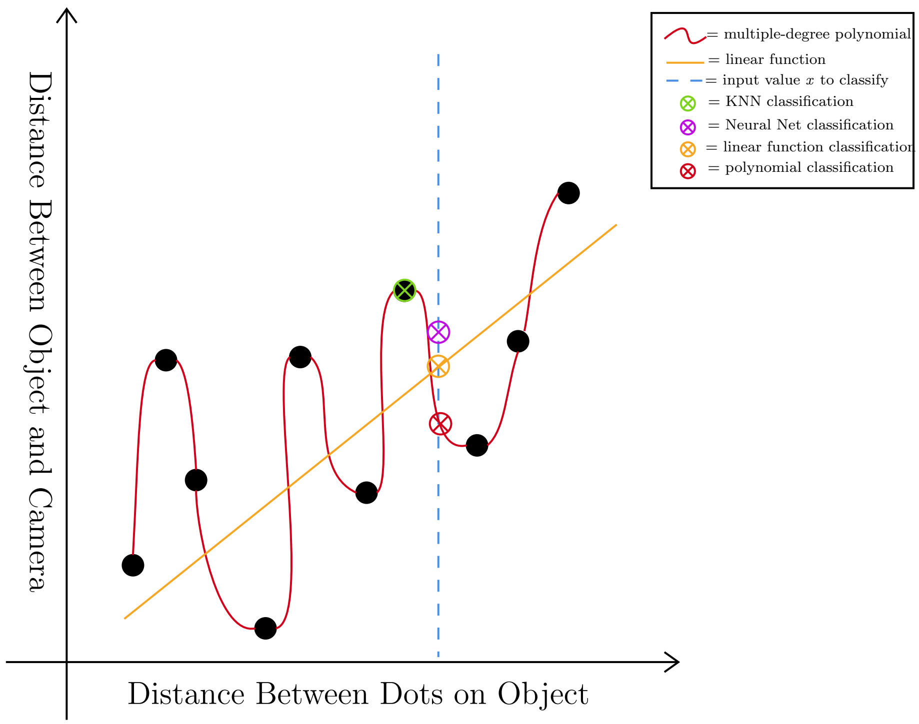

Figure 5: Different Classification Approaches on Sample Set

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 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 3 KNN Classification Algorithm

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

4. Implementation

4.1. Hardware

4.1.1. Refracting Grid of Laser Points

4.1.2. Computers

4.2. Software

4.2.1. Object Oriented vs. Procedural Neural Network Design

4.2.2. Feature Extraction

In constellation's implementation, there are three main parts of extracting features from a two dimensional image; the first is training our system to recognize the object

4.2.3. Distance Estimation

4.2.4. Environment Model Generation

4.2.5. User-Facing Application

5. Results

5.1. Algorithmic Analysis

Constellation's increased speed is due in part to its superior algorithmic efficiency. First consider its core object detection system, a neural network. The network is pre-trained, so in the analysis of its efficiency, the training steps, including the backpropagation algorithm used to fine tune its weights,

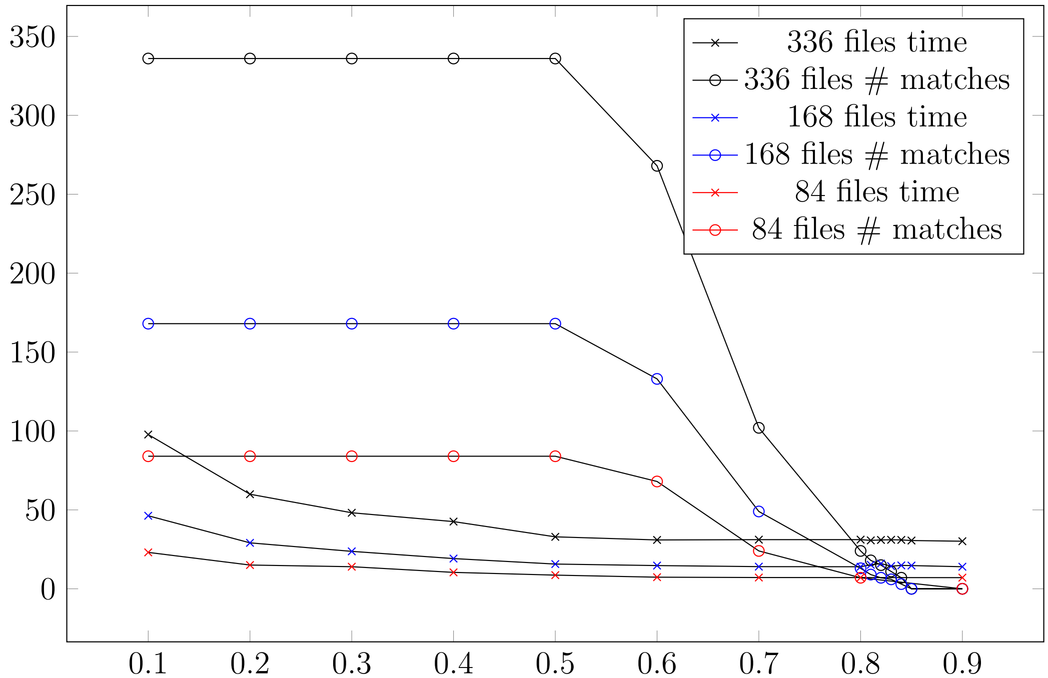

Figure 6: Number of Matches Found and Speed of Matching (seconds) vs. Confidence Level For Varying Amounts of Training Images

will be ignored, as at run-time, they will not play any role in the time and number of calculations it takes to determine the nature of an object. Thus, the complexity of identifying an object is only the complexity of feeding the data from the image through the neural network and producing an output. The derivations and declarations of variables used to figure the complexity of the feed-forward algorithm are:

$$\#\text{ layers} = \text{constant } c \;\; (\text{in our implementation } c = 5)$$

$$\frac{\#\text{ neurons}}{\text{layer}} = \sqrt{n} \text{ for } n \text{ elements in an input feature vector}$$

$$\frac{\#\text{ calculations}}{\text{neuron}} = n + 5$$

$$\#\text{ times object detection called} = (w - \sqrt{n})(h - \sqrt{n})$$

$$(\text{for image width } w, \text{ and image height } h;$$

$$\text{in our implementation } w = 1080,\ h = 720)$$

Thus, it can be determined that the number of calculations required to determine the classification of a set of n inputs is $5\sqrt{n}n + 25\sqrt{n}$, and further, the complexity of classifying all objects in an image is $O(n^2)$.

To perform similar tasks, a fairly naive implementation of a Structure-From-Motion pipeline would require $15n^3 + c$ calculations, and a stereo vision system implementation would require $find\ this$ calculations, both far more than Constellation's mere $find\ this$ needed calculations for fairly standard object image size of $n = 100$ pixels.

5.2. Implemented Execution Time

5.3. Environment Accuracy

6. Conclusion

6.1. Advantages Over Similar Systems

6.2. Shortcomings and Future Improvements for Constellation

- need to pre-train the network on objects before Constellation can put it in the environment accurately
- non-visual dot projection (infared or something else)
- increase time efficiency by concurrently carrying out object detection and dot detection operations (weight the pros and cons of distributed computations at that scale and their intensiveness on cpu, also weave in gpu acceleration)
- range, increase by stronger lasers and more training data for KNN model