Object Detection and Recognition Practical Course: Automated Bicycle

Mohamad Ayad
Faculty of Informatics
Technische Universität München
Boltzmannstr. 3, 85748 Garching bei München
Email: mohamad.ayad@tum.de

I. Introduction

What are the requirements for an automated bicycle? It should be able to drive by itself and steer. But that is not worth anything ifit has no knowledge of where it is driving. This report is about how to solve this problem. There are several solutions to this problem, many of them are a combination of many technologies. However, the solution proposed in this report will handle the use of cameras and image processing using common approaches such as image segmentation and machine learning. Cameras is not just good for taking photos and videos. They are relatively cheap instruments and are pretty good at giving information. With the combination of many cameras, the bicycle will be able to detect many objects from all sides. To make this possible, not just the image processing software has to work but also the mounting of the cameras is crucial. The report will be divided into two sections, hardware, and software, and will end up with a conclusion for the project as a whole.

II. Hardware

This section is about the hardware that is being used and how it is mounted onto the bicycle. The goal is to build a simple mount where cameras easily could be attached but also, offers great flexibility for modifications such as rotating the cameras and replace them. It is of utmost importance that the cameras are exposed for as few noisy obstacles as possible. Therefore, the mount has to move the cameras away from the bicycle but at the same time, they should catch as much information as possible.

A. Model proposals

1) Front model 1: In this model, the cameras are mounted on the handlebar. It consists of an aluminum rod, metallic rubber clamps, and plastic boxes. The clamps are mounted directly on the handlebar pointing upwards, the rod is connected to clamps using 8mm screws with nuts and bolts, and the boxes are connected to the rod using 5mm screws with nuts and bolts.

We build our model taking into consideration some future changes, based on that, the connection between the materials is very flexible. The height of the cameras with respect to the rod can be changed, the same for the height of the rod with respect to the clamps. Also, our cameras can be rotated and fixed again, this can be done just by releasing the screws and tight them again after changing the angle of the camera, also the camera position on the rod can be changed.

Fig. 1: First front model suggestion

By looking back at the description, it is clear that the main pro of this model is flexibility due to the ability to move the cameras and rotating them into different angles. Also, the whole scale can be changed without much effort. In addition, the model is very light and cheap. On the other hand, the boxes were built especially for those cameras, so the change of the cameras will lead to the change of the boxes.

2) Front model 2: This simple model uses a triangle-shaped wooden structure for mounting the cameras in the front. By choosing the triangle angles correct, the cameras will together have a maximum view and not too much overlap. To make the model even more flexible, the triangle is attached to a wooden piece. This wooden piece is attached to two modified stands which in turn is attached to the steering.

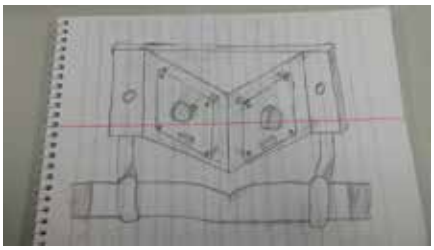


Fig. 2: Second front model suggestion

By using this approach, it is easy to remove and change model. It is not definite attached to the bicycle and it can

be moved to another location with ease. However, the model does not take into consideration of changing the angle of the cameras. It will be a built specific for these cameras. If the cameras will be changed at some occasion, the triangle structure must be rebuilt. An exception is if the new cameras are having the same field of view. To fix this problem, the triangle could be modified using a hinge. The hinge could be attached to the each side of the triangle and then allow the angle to become bigger or smaller. Such a construction could be built, but it also makes the construction more unstable due to a single point attachment.

3) Back model 1: The first model of where to put the camera in the back is simply to attach it beneath the saddle. By using a backlight fitting, the camera can be attached and removed with ease. The angle is also adjustable which makes it very flexible.

The strengths of this approach are the simplicity to remove and attach the camera. Its disadvantages are that it will be vulnerable for dirt and things hanging down from the saddle.

4) Back model 2 : The back model is easier to build due to that it only have a single camera. It could be built using an ordinary luggage rack and a plastic box.

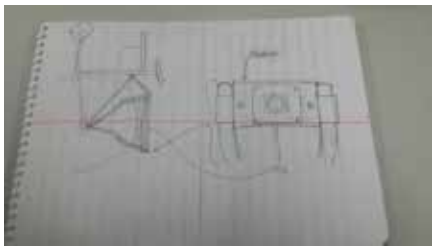


Fig. 3: Back mounting model

As seen in figure 3, the plastic box is only attached with two screws. The strengths of this model are its simplicity. It is easily attached and by placing the camera attachment directly on a luggage rack, the camera will be protected from dirt and such. What could be a problem is if the rack is unstable. Wobbling and a loose construction over time could make the overall construction unstable.

B. Comparisons

1) Front models : The difference between the models is the location of the cameras. It is also the availability to change the camera angle in different directions. As stated in the description, model one is able to change the angle of the cameras while model two is not. The cost of the models is similar.
2) Back models : As stated in the de- scription, model one is very easy to install and very flexible when changing the camera angle, compared to model two. It is also very cheap due to that it only needs a single attachment compared to model two, which needs a rack.

III. Conclusion -Hardware

The decision of model is based on the overall benefits from the model. Cost, stability, and flexibility are such benefits. The final impelementation of the mountings and its materials can be seen in appendix A.

A. Front model decision

Both models are strong candidates due to their simplicity. Although, the first model does have some more benefits. It is mainly the dynamic placement of the cameras and how they can be rotated in different angles, that makes this model the best. The rotation gives the possibility to catch different amount ofinformation. Due to the limitation of the cameras, limitations such that how much information that are catched by both cameras, this model gives an opportunity to adjust this. It also very stable and does not prevent the driver from using the steering.

B. Back model

The decision of the back model is based on the reduction of noise and impact from the rider. Model two reduces these factors which make it a great candidate for this mounting. It will not have the flexibility to move the camera in different angels. But since it is only one camera and the field of view is pointing backward all the time, this is not a real disadvantage.

IV. Future work - Hardware

Object detection can be improved by adding sensors. Cameras are not capable of deciding the distance for an object. By adding sensors, such as a lidar, distance measurement could be achieved. For objects close to the bicycle, such as in a situation where the bicycle is driving in the streets with a lot of cars, weaker sensors such as ultrasound sensors could be added.

V. Software

For the implementation of the software, the choice of approach is crucial to succeeding in detecting objects. The difficulties for object detection are to distinguish objects from a picture. What are objects and what are just noise?

A first idea is to find parts of the picture that differs over time, as objects. But since the bicycle are most of the time in a constant move, these parts does not always has to be objects but pure noise. The second idea is to think of threats and objects as something that moves either away or towards the bicycle. A detected region that is increasing could be seen as a threat and therfore something to avoid. Also if the region has a big size it should be considered as something to avoid.

But there are two different categories of objects, objects that actually move in reality and objects that is standing in one place all the time. Both types of objects will be seen as moving objects by the bicycle ifitself are on the move. So what is the difference? Lets say a motorcycle is approaching the bicycle, on the opposite side of the road. If the bicycle needs to make a turn, it needs to be aware of the motorcycle. In other words, the objects angle relative to the bicycle. The same logic goes with a big tree for example. The difference is

that the tree will not move. So if the bicycle always waits for a clear path, in other words waiting for the object to pass, it will never make the turn when it detects the tree. The problem is that trees and a motorcycle could look the same in the thinking of the regions. So the bicycle has to be able to distinguish these two objects. The first approach to solve these problems is to apply ordinary segmentation algorithms. Different algorithms returns regions either from sequential images or just a single image.

A. Research

1) Segmentation algorithms : A common way to calculate differences between images is background subtraction. It calculates the element-wise distance between two images called the foreground mask. The foreground mask will contain areas of where the two images differ. To then actually define an area ofinterest in the scenario an edge detector, such as the cany edge detector, is used followed by a contour approximation.[1][2]

Fig. 4: Backgroundsubtraction

With information provided by previous groups research in this course, the optical flow algorithm with edge detection seemed to be a good approach. The algorithm computes the motion in an image between two consecutive frames. The motion is caused by an object moving in front of a camera or that the camera itself moves. It is a 2D vector field where each vector is a displacement vector showing the movement of points from one frame to another. The assumptions that the optical flow algorithm does, is that the pixel intensities of an object, do not change between two consecutive frames. It also assumes that neighboring pixels have similar motions. [3][4]

Another way to segment the image is to use efficient graph based segmentation. Instead of representing an image as a grid ofjust pixels, it is now represented as a graph. Each pixel is defined as a vertex and has edges to all surrounding neighbors. The similarity or dissimilarity is then defined as the weights on these edges. By then partition the graph according to a criterion designed to model clusters, each such partition is considered as a segment. By using this approach, the algorithm does take into account that an object might have invariance in intensity.[10]

Fig. 5: Graph based segmentation

A fourth way is to use the Watershed algorithm with a distance transform. It starts by calculating the distance transform of a binary image. Then it applies the algorithm using the original image as a mask. OpenCV has a mark-based implementation, which means that it uses predefined valley points. The valley points are then defined as points to be merged and points that are not. Regions that are certain to be foreground or an object with one color and regions that are certain to be background or a non-object with another colour are labeled. The remaining regions are unknown. Regions of interest, regions that are known to be objects or non-objects, will then be found by the algorithm. [5]

Fig. 6: Watershed

Last is the corner detection. This method is good if motion in an image should be detected. There are several algorithms but the one that was taken into consideration here was the Sci-Tomasi algorithm. It finds the strongest edges out of N corners. A dissimilarity between two images defines a strong edge. The quality of an edge is measured between 0 and 1. If anything is measured below this, it is not considered and it will be filtered out. The rest are sorted in descending order.[6]

After some tests with each algorithm, the conclusion was that they did not provide enough information to achieve the task. They gave good results in terms of dividing the image into regions, but they did not give enough information regarding what object there were. To be able to classify the images, there has to be a prior knowledge about the found regions. So instead of object detection, a better approach would be object recognition.

2) Image processing with classification :
With the results from the segmentation algorithms, the conclusion was that an approach using machine learning together with segmentation could be a better solution. Especially if an angle should be given to each found object.

So how is an image classified, so that all objects ofinterest are classified separately? CNN or Convolutional Neural Networks are good for classifying images[7]. It splits the image into subcomponents and then decides using a predefined model what there is in the picture. With this approach, all interesting areas, both sub images and the image as a whole can be checked for objects ofinterest. However, it is slow to classify images when executed on an ordinary CPU. Different predefined models render different result and with different accuracy[8].

As most CNN libraries are heavily depending on GPU:s[9] it puts certain requirements on the hardware. In this project, the hardware is limited which makes it important to optimize the software. This means that the classifier should only classify regions which are ofinterest and not regions that most likely not containing any vehicles. A simple approach for such region proposer would be to implement an exhaustive search algorithm such as the sliding window algorithm. It takes rectangular snapshots of the image and then proposes these sub-images to the classifier. This are however, a quite heavy task due to that many of these images would be classified as nothing or at least nothing ofinterest. To improve this, the region proposer could instead propose smarter by looking at regions that may contain similar colors. This can be done by graph-based image segmentation[10], such as the one described in the segmentation algorithms. The proposed algorithm is called selective search algorithm [11]. The selective search algorithm finds interesting regions using a segmentation algorithm called Felzenwalb. It then returns the regions in the picture which might be objects. The Felzenwalb segmentation consists of three parameters$scale$, $sigma$, and min_size . The scale parameter limits the size of the clusters of colors that the Felzenwalb segmentation can detect. The sigma defines the length of the gaussian kernel that is used in the segmentation for smoothing the image as well as removing certain artifacts. The min_size defines the minimum component.

VI. Classified images situations

Simple images, images that not contain a lot of noise, are easy and fast to classify. The region proposer can easy distinguish the different areas without much pain. However, in a real life situation, such pictures rarely exist and the proposer needs to consider a lot of different regions.

A. One vehicle alone

The most simple scenario is on a single vehicle alone without any noise. The figures below shows a car in profile non-classified(7a), classified with parameters$scale = 900, sigma = 0.8, min_size = 30$(7b) and classified with parameters $scale = 1000, sigma = 0.8$ and $min\_size = 90$(7c).

If the parameter$scale$ is chosen to be a high value, the region proposer will end up giving a proposal such as the one in figure 7b or figure 7c. However, if the$scale$ is adjusted to a much lower value, in this case,$scale = 400$, the classifier will classify the whole picture and therefore the whole car as seen in figure 7d.

(a) Single car, non-classified

(b) Single car, classified with $scale = 900$ and $min\_size = 30$

(c) Single car, classified with $scale = 1000$ and $min\_size = 90$

(d) Single car, classified with $scale = 400$ and $min\_size = 30$

Fig. 7: A single car 7a Ref.[12]. All classified images uses $sigma = 0.8$

B. Several vehicles in a row

This is another scenario, where several vehicles appear in the front of the bicycle. The optimal solution would be to find all vehicles and report them. But it could also be sufficient to just report the closest one. Below is two classified images with the same choice of parameters as in the previous example.

(a) Car queue, non-classified

(b) Car queue, classified with $scale = 900$ and $min\_size = 30$

(c) Car queue, classified with $scale = 1000$ and $min\_size = 90$

Fig. 8: Car queue 8a, Ref. [13]. All classified images uses $sigma = 0.8$

As can be seen in the classified images, the result differs a bit from the result with the single car. By using the lower value on themin_size, $min\_size = 30$ and $scale = 900$, only the white truck will be classified(figure 8b). The closest car will not even be considered as an interesting region. But in the third

figure 8c, where the $scale = 1000$ and the $min\_size = 90$, it will be a much better result. This is because it will both classify the closest car as a vehicle and the truck in the front.

C. Scattered vehicles in front of the bicycle

Next scenario is when vehicles are scattered in front of the bicycle. It is important that all vehicles are found and classified in this image.

(a) Scattered vehicles on highway, non-classified

(b) Scattered cars on highway, classified with $scale = 900$ and $min\_size = 30$

(c) Scattered cars on highway, classified with $scale = 1000$ and $min\_size = 90$

Fig. 9: Scattered vehicles in front of the bicycle 9a, Ref.[14]. All classified images uses $sigma = 0.8$

As seen in the previous scenario a smaller value on min_size will result in that only some of the vehicles are being classified. In figure 9b, the closest car will be found and classified. But if the min_size is raised, the classifier will also find the cars a bit further away as seen in figure 9c.

D. Vehicles approaching from behind

This scenario is occurring when the bicycle is having many vehicles approaching from the behind. Just as in the previous scenario with the scattered vehicles, every vehicle must be found.

(a) Cars approaching from behind, non-classified

(b) Classified cars approaching from behind, $scale = 900$ and $min\_size = 30$

(c) Classified cars approaching from behind, $scale = 1000$ and $min\_size = 90$

Fig. 10: Vehicles approaching from behind 10a, Ref.[15]. All images uses $sigma = 0.8$

In this scenario, with the same configurations as before, both configurations is sufficient as seen in figure 10b and figure 10c.

VII. Conclusion - Software

The conclusion is that the main challenge is not to find objects anymore but to do this in real time and to distinguish objects from each other. As mentioned in section VI, it is not enough to just find areas ofinterest. The segmentation could result in an area which does not propose a threat to the bicycle, but it could be ofimportance due to what it is and in which situation.

Fig. 11: Classified cars approaching from behind, with inaccurate classification but good segmentation.

Take a look at picture 11. In the middle, two white lines are classified as a limousine which is clearly wrong. In the main context of the picture, these lines do not propose a real threat to the bicycle. But the bicycle must be able to make a switch to a line and then it should not see the lines as something uncrossable. However, if the bicycle is alone on a highway, these lines are very important. The bicycle should not drive on the wrong side of the road and must, therefore, keep track of these lines. Without classification, these lines are seen as objects and therefore proposed as potential threats. And that means every situation!

The classification is a tedious process especially when runned on an ordinary CPU. However, the Caffe library has

some tricks that could be applied to fasten up the classification. Caffe defines a network with the class Net. The Net, Layers, and Solvers use another class called Blob, which represents a holder of memory with which the other uses as a computational unit. If this computational unit is optimized, in this case just reshaped to fit better dimensions and channels, the classification will increase rapidly.

An important notice for the captured images. The selective search will work on RGB images but the classification will not. Since a captured image from OpenCV always is in the BGR format, there have to be two conversions. One from BGR to RGB and one from BGR to grayscale.

What also slows down the process is the region proposer. Different values ofits parameters will make it more or less sensitive. As seen in the classified images situations section, the same values do not always give the best result. The right front tire of the single car, for example is classified as something interesting but the calculated angle will point to the tire and not to the center of the car. If the car would be long enough it would mean that the bike could make the decision of driving right into the back.

VIII. Code and Software architecture

The software is written in python and uses the libraries Caffe, Numpy, OpenCV and Scikit-image to process the images. A simple API is provided with which a simple client could be built.

1) Classes and parameters: The software consists of five classes:

1. ThreadedDetector - This class is the core class which defines all detectors. It is created by calling the constructor with all necessary arguments such as a list with id:s for each camera connected.
2. ClassificationThread - The class that is responsible for classifying images from one camera. Each camera connected will have an associated ClassificationThread.
3. CaptureThread - Each camera that is connected will have a CaptureThread. Its only responsibility is to grab pictures from a camera and provide them to the ClassificationThread.
4. WordDetector - This is a single instance class to which all classifiers are asking if what they have found is relevant. The WordDetector will parse a file with words or classes that should be seen as relevant by the classifiers.
5. InformationBase - Also a single instance class. Its responsibility is to provide and store information passed by the detectors. A client that is built around the ThreadedDetector should call this class to get information such as newly captured images from the camera, classified images and angles to found objects.

So how is the software built? As already mentioned, the ThreadedDetector is the core class that should be called. It takes six arguments at minimum but thirteen at most. The six arguments that must be passed are:

1. detector_ids - An array of camera ids. This is exactly how OpenCV defines different cameras, by integers numbered from 0 and up.
2. vehicle_words - The path to the file containing all the relevant classes and names that the WordDetector class is parsing.
3. mean - This is a file used by Caffe for calculating the mean in pictures. When installing Caffe, such file could be found in the<path-to-caffe> /caffe/python/caffe/imagenet/ directory.
4. pretrained _model - The pretrained model, in this case <path-to-caffe>/caffe/models/bvlc_ reference_caffenet/bvlc_reference_ caffenet.caffemodel . This file contains the weights for the CNN.
5. model_def - This is the design file for the CNN network. Each such file can be found with its corresponding .caffemodel-file, <path-to-caffe>/caffe/models/bvlc_ reference_caffenet/deploy.prototxt
6. synset_file - All classes that could be classified is mapped with their names in this file. The simplest solution, without training the network is just to use the <path-to-caffe>/caffe/data/ ilsvrc12/synset_words.txt which was used when the bvlc_reference_caffenet model was trained.

The seven optional arguments that can be passed are:

1. camera_resolution - This option enables to change the camera resolution of the cameras. It is defined as an array with four elements, [CV_CAP_PROP_FRAME_WIDTH, "Desired width of the picture", CV_CAP_PROP_FRAME_HEIGHT, "Desired height of the picture"] where the CV_* is two constants provided by the OpenCV library.
2. raw_scale - Single float used by the Caffe transformer. Default value is 255.0.
3. channel_swap - Tuple containing three integers which are used by the Caffe transformer. The default value is (2,1,0) .
4. mode_gpu - This is boolean which indicates if Caffe should classify using a GPU or a CPU. If this is set to True , Caffe must be compiled with CUDA support and there must be a CUDA compatible graphics card available. Default value is False .
5. debug - Just for showing some output if the detector is debugged.
6. selective_search_params - Specifies a dictionary with the following structure: { 'scale': 150, 'sigma': 0.8,

'min_size': 30 }.
This is by default set to:
scale=400, sigma=0.8, min_size=10 . As mentioned before, these parameters control the behaviour of the selective search algorithm.

7. use_partly _angle_check - This enables a pre-check by the classifier. The idea is that if the classifier do not find anything ofinterest in an image, there is no need to look for angles. By default, this parameter is set to false which means that the detector will always look for objects, classify them and then try to calculate the angles.

The program could be executed as a standalone program. However, this is only preferred ifit should process a single picture and not a video. To be able to process a single image, the code needs some small modifications. The lines that need to be out commented can be found in the code.

2) Building a simple client: A client is simply built by creating an instance of the ThreadedDetector and then follow the same style as when building a video capture software using OpenCV. See Appendix D.

IX. Future work - Software

There are a lot that can be improved with the software, such as the speed of the classification. It can be improved by using different models. The CNN that was used in this project, was the BVLC CaffeNet model[16]. But models such as the GoogLeNet_cars model[17] and the Cityscapes dataset[18] can improve the classification by adding more objects to the classification. However, the speed also heavily depends on how many layers a network have. A relatively small network will be fast but may lack the accuracy a larger network provides.

The selective search algorithm is depending on the Felzenwalb algorithm but can be tested more extensively with other segmentation algorithms such as quickshift clustering or k-means clustering. Both algorithms is implemented in the scikit-image library and can be called by skimage.segmentation.quickshift respectively skimage.segmentation.slic .

X. Overall project conclusion

As can be seen in the house of quality, the first approach was to mount both sensors and cameras on the bicycle. However, since the result is not based on using sensors anymore, the house of quality does not fit completely with the work that has been done. But most demanded qualities are fulfilled and their respective quality characteristics are either maximized or minimized. See appendix B. The bicycle has the mount for the cameras which have been built with the consideration of noise, the field of view and accuracy. The software is also built with these parameters in mind. But as already mentioned, the hardware limits it and there have been some difficulties to make it very accurate.

The project timeline has not entirely been followed. This is mainly because of the dead-end with the segmentation algorithms. See appendix C. But otherwise, the amount of time spent on each part has been estimated well.

XI. Overall project future work

As said in the previous sections, IV and IX, to succeed in preventing the bicycle to drive into obstacles an interaction between hardware sensors and the current image processing should be implemented. Also by adding an external graphics card to the bicycle, the classification could be improved greatly.

References

[1] Elgammal, A., Harwood, D. and Davis, L., 2000, June, Non-parametric model for background subtraction, In European conference on computer vision, (pp. 751-767), Springer Berlin Heidelberg.

[2] Javed, O., Shafique, K. and Shah, M., 2002, December, A hierarchical approach to robust background subtraction using color and gradient information. In Motion and Video Computing, 2002. Proceedings. Workshop on (pp. 22-27), IEEE.

[3] Horn B.K. and Schunck B.G., 1981, Determining optical flow, Artificial intelligence, 17(1-3), pp.185-203.

[4] Optical Flow, 8 January 2017., Optical Flow Wikipedia, Retrieved from https:en.wikipedia.orgwikiOptical_flow

[5] Beucher, S., 1992, The watershed transformation applied to image segmentation, Scanning Microscopy-supplement, pp.299-299.

[6] Shi, J., 1994, June, Good features to track, Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference, (pp. 593-600).

[7] Hijazi S., Kumar R. and Rowen C., 2015., Using Convolutional Neural Networks for Image Recognition., IP Group, Cadence..

[8] Mital P. K., 4 January 2015., Real-time object detection using ofxcaffe, Retrieved from http://pkmital.com/home/tag/object-recognition/

[9] Jia, Y., Shelhamer E., Donahue J., Karayev S., Long J., Girshick R., Guadarrama S. and Darrell T., 2014., Caffe: Convolutional Architecture for Fast Feature Embedding., arXiv preprint arXiv:1408.5093.

[10] Felzenszwalb P. F., Huttenlocher Daniel P., 2003, Efficient Graph-Based Image Segmentation, International journal of computer vision, 59(2), 167181

[11] Uijlings, J. R., Van De Sande, K. E., Gevers, T. and Smeulders, A. W., (2013)., Selective search for object recognition., International journal of computer vision, 104(2), 154-171.

[12] Mercedes-Benz 450 SEL, Available at:https://commons.wikimedia.org/wiki/File:Mercedes-Benz_450_SEL _(V116) _in_profile.jpg, [Accessed 16 August 2015]

[13] Brandom R., (2012), Self-driving cars can navigate the road, but can they navigate the law?, Available at:http://www.theverge.com/2012/12/14/3766218/self-driving-cars-google-volvo-law, [Accessed 7 March. 2017]

[14] Pixabay, Available at:https://pixabay.com/sv/trafikskola-k%C3%B6r-bil-gator-motorv%C3%A4g-397293/, [Accessed 20 July 2014]

[15] Pexels, Available at:https://www.pexels.com/photo/traffic-cars-street-traffic-jam-7674/, [Accessed 7 March. 2017]

[16] Donahue J., BVLC CaffeNet Model, 2015, Github repository:https://github.com/BVLC/caffe/tree/master/models/bvlc_reference_caffenet, Commit: 2cc3844cb2a4a72de10d321781dc8f994bef95c7

[17] Yang L., Luo P., Loy C. C. and Tang X., 2015, A Large-Scale Car Dataset for Fine-Grained Categorization and Verification, arXiv:1506.08959

[18] The Cityscapes Dataset, 2017, Retrieved from https://www.cityscapes-dataset.com/

APPENDIX A
MATERIALS FOR CAMERA MOUNTING

1) Materials for the front model

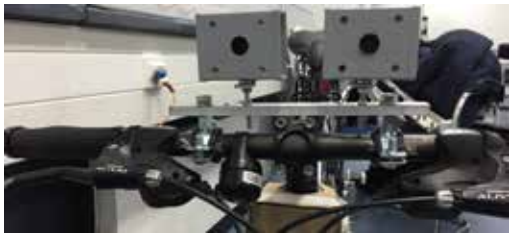
(a) Metallic rubber clamps

(b) 7cm x 5cm x 3cm plastic boxes for cameras

(a) 8mm and 5mm screws bolts and nuts

,(b) 22cm aluminum rectangular rod

(a) 22cm aluminum rectangular rod



(b) Front model attached

2) Materials for the back model

(a) Mounted back rack.

(b) 7cm x 5cm 3cm plastic box attached to back rack.

APPENDIX B

SIMPLE CLIENT CODE

```
import threaded_detector as td
import cv2

threaded_detector = td.ThreadedDetector(detectorids=[0],
                                        vehicle_words='./gnetmodels/carrelated_words.txt',
                                        mean='./gnetmodels/ilsvrc2012_mean.npy',
                                        pretrained_model='./gnetmodels/referencecaffenet/
                                        bvlc_reference_caffenet.caffemodel',
                                        model_def='./gnetmodels/referencecaffenet/deploy.prototxt',
                                        synset_file='./gnetmodels/synset_words.txt')

threaded_detector._start()

while True:
    image = threaded_detector.information_base.get_latest_camera_image_for(0)
    processed_image = threaded_detector.information_base.get_latest_image_for(0)

    if image is not None:
        cv2.imshow("Camera", image)

    if processed_image is not None:
        cv2.imshow("Processed", processed_image)

    if cv2.waitKey(1) & 0xFF ==  ord('q'):
        threaded_detector._stop()
        break
```