Cyclist Collision Detection in Urban Environments
— Background Report —

Peter Hedley
ph817@doc.ic.ac.uk

Supervisor: Dr. Stefan Leutenegger
Course: CO541, Imperial College London

5th June, 2018

Abstract

This report investigates the use of existing state of the art image detection, segmentation and SLAM algorithms for use by cyclists in urban environments. The challenges facing cyclists in a daily commute include monitoring pedestrian movement, vehicle movement and obeying the rules of the road. The aim of the project is to provide a detection system to enable safer cycling on dangerous city-roads.

Contents

1 Introduction

There were c. 18,500 cycling accidents and 3,500 deaths on British roads in 2016 [1]. Of these accidents 80% occur during the day and driver/rider error accounts for 71% of the total collisions [2]. The most common place for a cycling incident is at a road junction with 75% of collisions occurring there [3]. With cycling becoming a more accepted form of transport, the need to protect and modify bikes with a comprehensive safety system is important.

This report will examine state of the art technologies that are used for autonomous vehicles to investigate their applicability to a cycling environment. The three main topics involve image detection/segmentation, SLAM algorithms and collision/path prediction.

2 Background, Computer Vision

2.1 Deep Learning Preliminaries

Computer vision and its applications has been revolutionised recently by Deep Learning frameworks. These have led to super-human accuracy in image recognition, for example Inception v3 [4], release in 2016, reached a 3.46% error rate on the ImageNet [5] dataset. This is known of as an image classification task; the photo contains one object in the dataset and that object generally occupies the majority of the frame. Similar networks are used with additional features for image detection and segmentation which are integral to detecting moving objects such as pedestrians and vehicles in an urban environment.

The current state-of-the-art methods in computer vision use a form of neural network. These networks are characterised by a series of non-linear nodes that are organised in layers. At each layer the network learns progressively abstract and higher-level features of the image to be detected until the final (usually dense layer) at the end which determines the predicted class. Therefore the 'depth' of a neural network often refers to the number of layers and hence the inherent complexity of features that a network can learn. This is, however, only a basic representation of the problem and some clever and shallower architectures are able to outperform deeper networks. It is important to note that the depth and width, more specifically the number of parameters in the model generally reduces speed of prediction and increases the memory required to store/load the model into RAM. This is of little consequence in most applications, however for the limited computational power able to be carried on a bicycle it will be of concern in this report.

2.2 Convolutional Networks

A CNN generally contains convolutional, pooling, fully connected and normalisation layers. This form of network became particularly popular in computer vision after Geoffrey Hinton's 2012 paper [6], with an ImageNet classification error of c.16%, significantly lower than the state-of-the-art at the time 26%. Convolutional networks use the assumption that each input is an image and therefore certain properties can be encoded into the architecture of the network. This allows a significant reduction in the number of network parameters to tune.

A regular neural network, as mentioned earlier, contains a number of fully connected layers i.e. each neuron in a layer is connected to every neuron in the previous and next layers respectively. However convolutional neural networks are arranged in three dimensions: width, height, depth (Figure 1). In this architecture the neurons in a layer are only connected to some of the layer before unlike in the fully connected case, this significantly reduces the number of parameters [7].

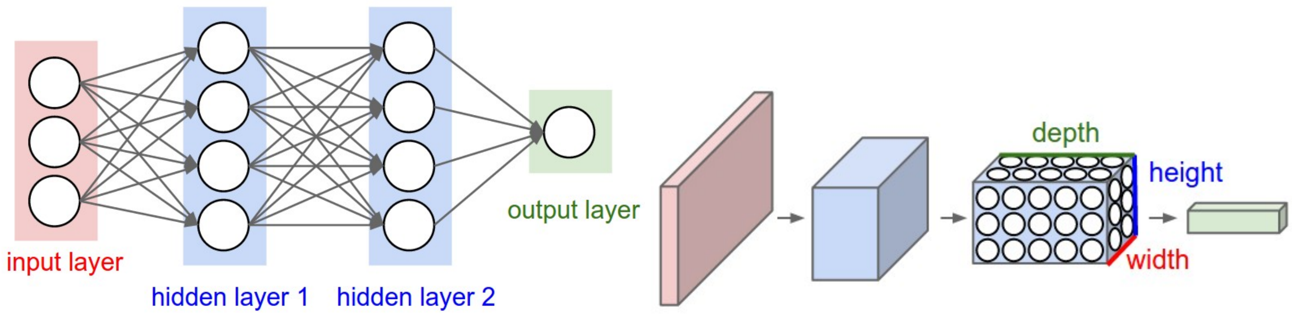


Figure 1: Regular Neural Network (Left) [7] , CNN (right) [7].

2.2.1 Convolutional Layers

A convolutional layer learns a set of filters, each filter is compact in size, say $(7 \times 7 \times 3)$. This filter slides (convolves) over the image, then a dot product is taken between the filter values and the inputs to that filter (Figure 2. The filter slides across with a respective stride, which is a hyperparameter, a stride of one equates to moving one pixel across at a time. A convolutional layer is therefore able to pick up features such as a corner, or higher level features such as the texture of fur. This process creates a 2D activation map containing the response of the filter at each position on the input volume. A number of filters are passed over and a number of activation maps created, relating to the depth which is a tune-able hyperparameter.

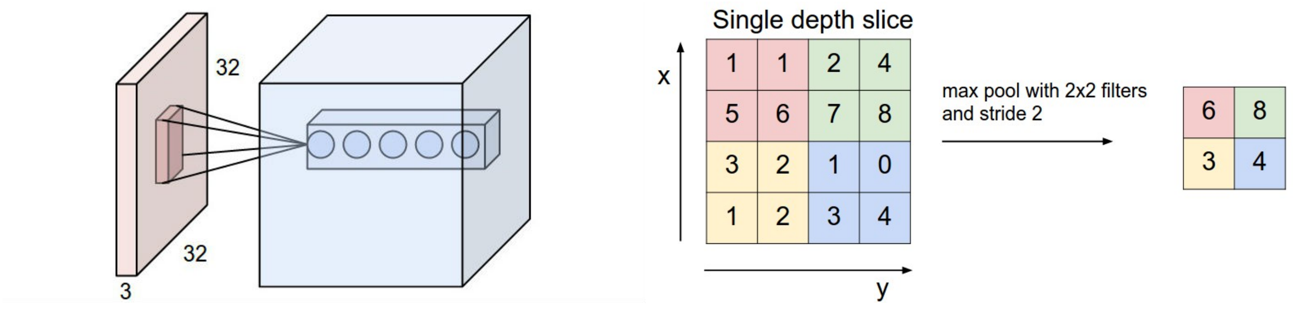


Figure 2: CNN Filter (left) [8], Max Pooling [9]

2.2.2 Pooling Layers

Pooling layers are used to reduce the number of parameters in a network. This is a much debated topic, Geoffrey Hinton's [10] dicusses the problems of convolutional networks in his 2017 paper and especially focuses on the disadvantages of pooling layers. It is however a current, although fading, method of parameter reduction. For the purposes of optimising network architecture for smaller hardware such as is portable on a bike this is still a recognised and useful method.

A pooling layer such as that depicted in (Figure 2) passes a kernel over the input (much like in a convolutional layer) and at each spacial location it calculates a metric. For example, max pooling takes the largest value contained within the kernel grid at each consecutive spacial location.

2.3 Image Detection

Image detection is an area of research that uses the CNN networks described in Section 2.2. This is, however, a more difficult task as there can be multiple objects/classes of interest within one image. A naive approach to this problem would be to slide a window across an image and classify at each location. When the network classifies, say a cat at a particular window location it is likely that a cat is located in that area.

Although the aforementioned approach is valid it would be very slow, not only would it be needed to slide the window over the whole image, the size of the window would also need to be varied. The network cannot know the area of the photo that the object occupies and therefore what size to make the window. There are a number of differing approaches to this problem R-CNN [11] and other later iterations vs YOLO (you only look once) [12]. The R-CNN approach is very similar to the vanilla, naive solution, discussed above (commonly known as exhaustive search) and is used by later techniques for instance segmentation.

2.3.1 R-CNN

R-CNN, proposed by Ross Girshick et al. [11] in 2014, incorporates a region proposal method. This process produces 2000 proposals which are then classified by a large CNN classifies each region using class-specific linear SVMs. This process equates to a 53.7% mean average precision (mAP) compared to 35.1% by the current state-of-the-art at the time. In addition to this performance gain the system doesn't rely on far more complicated ensemble methods.

The region proposal method used is called selective search [13] , this uses a fast segmentation algorithm by Felzenszwalb [14] followed by a greedy algorithm to iteratively group regions together using a similarity metrics. The four metrics proposed in the paper are s_{colour}, $s_{texture}$, s_{size}, s_{fill} the aim is to use a number of metrics to pick up the differences between regions when growing and also limit one region's growth at the expense of others (hence the size metric).

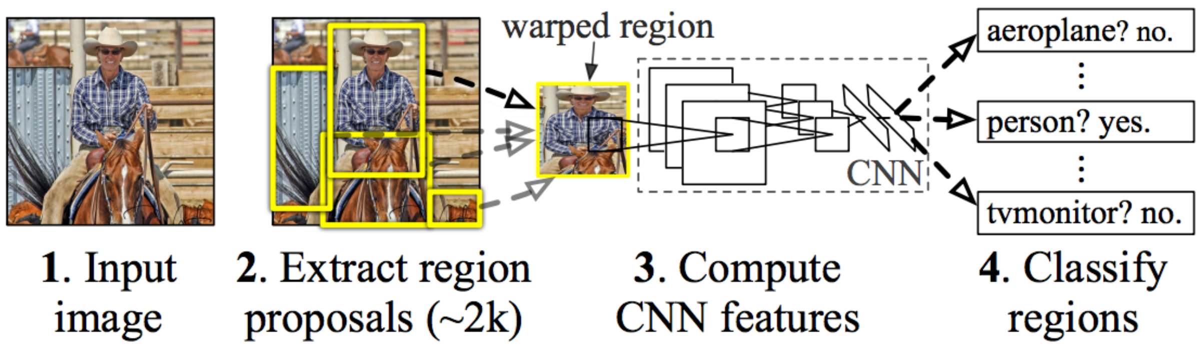


Figure 3: R-CNN Region Proposal & Architecture [11]

The result of the region proposal (Figure **??**) are a series of windows that are likely to contain an object. These are then simply warped to 227×227 pixels to fit the input dimensions CNN network that contains five convolutional layers and two fully connected layers. The output of the CNN is then fed as a 4096 dimensional feature vector to a set of trained SVMs, one for each class, that are used to score the likelihood for that class. This approach is slow at 13s/image on a GPU and 53s/image on a CPU, the separation of a CNN and SVM is peculiar and adds to this timescale. A later iteration of this method Fast-RCNN is discussed in Section 2.3.2.

2.3.2 Fast-RCNN

R-CNN performs a CNN forward pass for each region proposal and does not share computation [15]. Spacial pyramid pooling networks (SPPnet) [16] are designed to mitigate this problem by creating a feature map (which is always the same shape) from an input image and then classifies each object proposal from a vector extracted from the shared feature map. This paper still uses a multiple-stage classification pipeline but is a marked improvement on R-CNN with regards to speed.

Fast-RCNN uses the SPPnet concept but also incorporates a single-stage pipeline. The process inputs an image and multiple RoIs into a fully convolutional network (Figure 4). Each RoI is then pooled into a fixed-size feature map as in SPPnet and then mapped to a feature vector by fully connected layers. As can be seen in Figure 4 the network then branches into 2 separate outputs, one 4-point regression output of the bounding box location and a K-size classification output, where K is the number of object classes. The model therefore must contain a multi-task loss function that accounts for both bounding box and softmax loss (Equation 5).

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v) \tag{1}$$

in which $L_{cls}(p, u) = -log(p_u)$ is the log-loss for true class u, and L_{loc} is the bounding box regression loss defined as:

$$L_{loc} = \sum_{i \in \{x,y,w,h\}} smooth_{L_1}(t_i^u - v_i), \tag{2}$$

in which

$$smooth_{L_1} = \begin{cases} 0.5x^2 \text{if} x < 1 \\ |x| - 0.5 \text{otherwise} \end{cases} \tag{3}$$

is a L_1 loss. This architecture allows for fast training with the forward pass of the network and therefore processes images 146× faster than R-CNN.

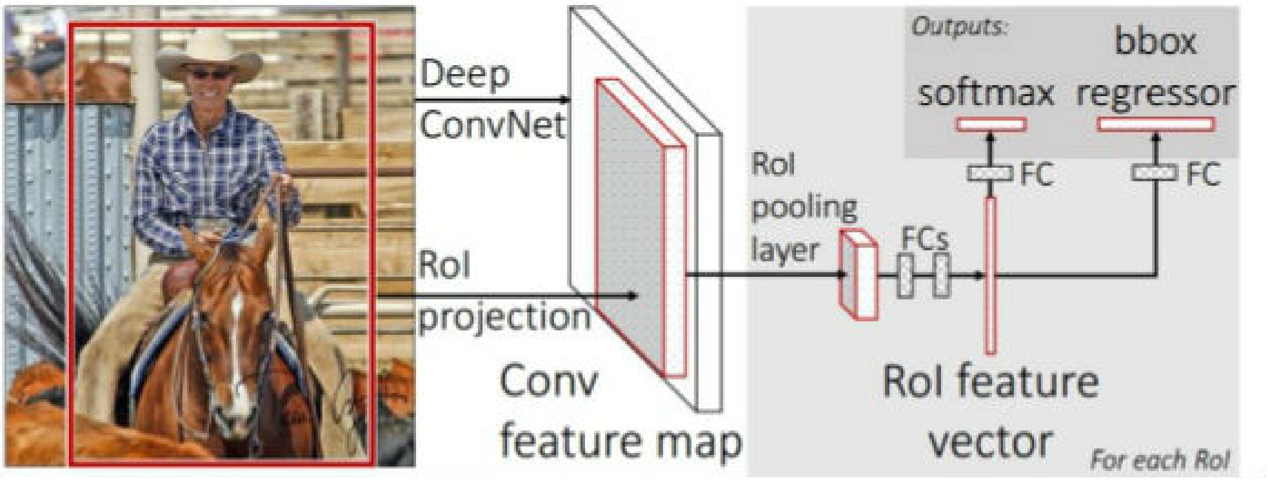


Figure 4: Fast R-CNN Architecture [15]

Figure 2.3.2 depicts the pipeline architecture, this incorporates an RoI pooling layer which maps the convolutional feature map from an RoI (created by selective search) to a fixed extent feature map of a set $H \times W$, where H and W are layer hyperparameters. Each RoI is a rectangular window defined by a tuple of four elements (r, c, h, w), where (r, c) specify the top left corner and (h, w) the width. RoI max pooling then divides the RoI window into a grid and then max-pools each grid into the corresponding output cell. The size of the grid is proportionate to the RoI and so the output shape is constant regardless of RoI input size.

2.3.3 Faster R-CNN

Although the previous method [15] removed the need for a multi-stage classification pipeline (CNN and SVM), it still relies on the selective search [13] method for region proposal. This part is now the computational bottleneck that slows down the process. Faster R-CNN proposed by Ren et al. [17] seeks to alleviate this issue by combining region proposal into the CNN. The paper suggests the use of a region proposal network (RPN) to produce the RoIs used later in the network.

The region proposal network (RPN) outputs a set of rectangular object proposals, each of these proposals has an objectiveness score i.e. how likely the network thinks the proposal contains an object. This part of the network comes after a number of convolutional layers (in this paper 5 or 13 such layers). The region proposals are then created off the feature map from these layers by sliding a smaller network of size $n \times n$ over the convolutional feature map (Figure 5. At each sliding window location a number of region proposals are predicted simultaneously. These consist of 3 aspect ratios and 3 scales, therefore 9 total region predictions or anchors at each location. The network is then followed by two sibling 1×1 convolutional layers, one for *reg* (region proposal) and one for *cls* (probability of object or not). The *reg* layer outputs $4k$ results, which represent the encoding coordinates of the k boxes and the *cls* later scores the probability that there is an object or no object in each proposal, therefore $2k$ outputs. This results in a convolutional layer output of depth 9.

The loss for the network is calculated from two metrics, the output of the *reg* layer is a binary i.e. positive example or not. A positive example is defined by how much the box overlaps with the ground truth box of the object and so an IoU (Intersection-over-union) overlap ¿ 0.7 is considered

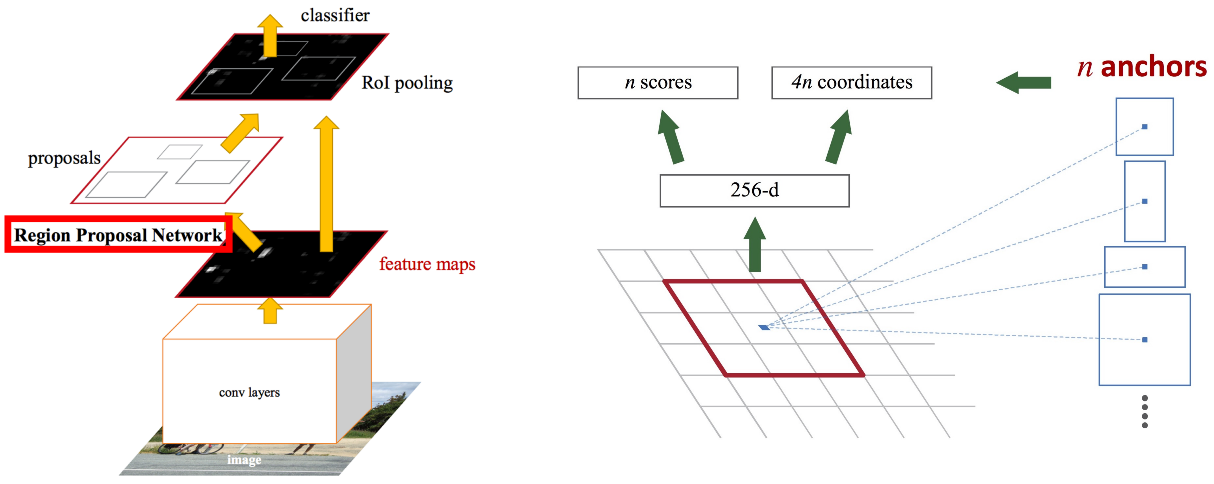


Figure 5: Fast R-CNN Architecture [15]

a positive example and an IoU ¡ 0.3 is considered a negative label. For the first positive case if no positive examples are found the boxes with highest IoU are labelled positive. With this it follows that:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(pi,p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i,t_i^*) \tag{4}$$

Where i is the anchor index, p_i is the predicted probability of anchor i containing an object. The ground truth label p_i is denoted as a 1 for a positive anchor and 0 for a negative therefore L_{reg} is only activated for a positive example, t_i and t_i^* represent the 4 coordinates of the bounding boxes, and L_{cls} is the classification loss. The two terms are normalised (N_{cls} and N_{reg}) and balanced by a weighting parameter λ. The final term to discuss, the bounding box regression is calculated, simply, as follows:

$$\begin{aligned} t_x = (x - x_a)/w_a, \quad t_y = (y - y_a)/h_a, \\ t_w = log(w/w_a), \quad t_h = log(h/h_a), \\ t_x^* = (x^* - x_a)/w_a, \quad t_y^* = (y^* - y_a)/h_a \\ t_w^* = log(w^*/w_a), \quad t_h^* = log(h^*/h_a) \end{aligned} \tag{5}$$

where x, y, w, and h are the box centre coordinates plus width and height. Predicted box, anchored box are denoted by x, x_a, x^* (likewise for y, w, h).

Finally after the regions have been proposed c.6000 per image, some regions overlap with others, and so, to reduce redundancy non-maximum supression is used on their cls score with an IoU threshold of 0.7 to leave around 2000 proposals per image. Non-maximum supression is a technique that sets all neighbouring areas to zero around a local maximum and hence only keeps the maximum value and suppresses the others.

2.4 Image Segmentation

Unlike the image detection problem described in Section 2.3, this task aims to segment an image on a pixel-by-pixel basis into respective classes (Figure 6). A conventional classification network takes an image and outputs a prediction of which class the image is i.e. a vector of probabilities or a 'score'. This process uses fully-connected layers to produce the final output vector which throws away important spacial information used in a segmentation task. Therefore Long et al. [18] propose converting these final layers to convolutional layers, maintaining the spacial information and creating a heatmap output (Figure 6).

The heatmaps produces from the network then need to be upsampled from their respective dimensions to that of the output image. This process is essentially a reverse or deconvolution. Upsampling, therefore, with factor f is convolution with a fractional input stride of 1/f. Essentially each filter is

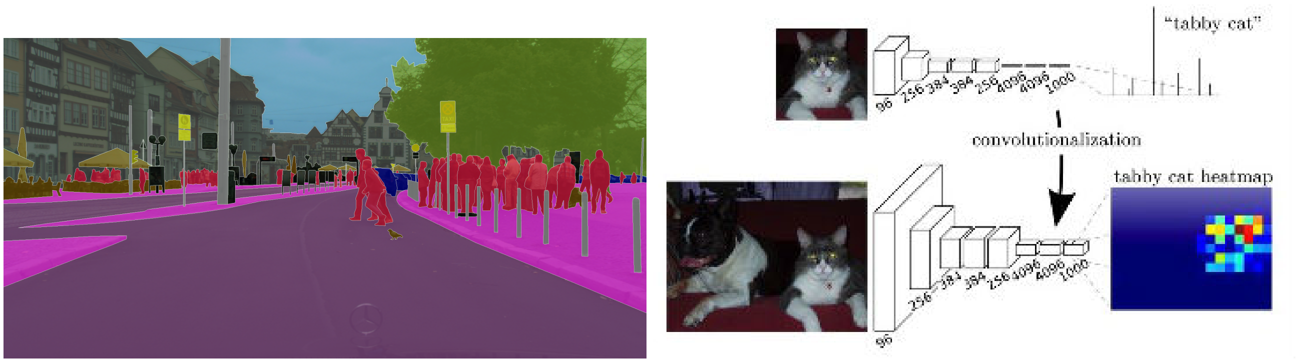


Figure 6: Image Segmentation [19], Convolutional Heat Map [18]

placed on the output image and then multiplied by the input pixel. Hence the image increases in size depending on stride and kernel size as one pixel value is interpolated over a grid. The final problem therefore is how to choose the filter values, this can be done in a number of ways but the main accepted approach is to simply learn them as part of the tuning process.

An important concept is that of patchwise training which feeds the network a number of patches from an input image (small patches surrounding objects of interest) instead of an entire input image. This both helps to balance the classes, ensures the input has enough variance, and is a correct representation of the input set. However, this paper argues that this can be done from a fully convolutional training regime with incorporating a DropConnect-like mask [20] between the output and the loss. A DropConnect mask is similar to the proposed regularisation technique of Dropout [21] except that it randomly sets the network weights to 0 rather than the activation functions. This, on the output layer (connected to the network loss) is found to have the same effect as patch-wise sampling in Long et al.'s paper.

Although conventional, proven classification architectures performed to state-of-the-art when modified for segmentation 56.0 mean IU it was found that a novel and specific architecture performed best 62.7 mean IU. Mean pixel intersection over union (Mean IU) is a standard performance metric where mean is taken over all classes, including the background. The paper describes a network architecture that uses skips [22]. Skips allow predictions from lower-level coarser layers to interact with finer, latter layer predictions. This lets the model make local predictions (finer) within the context of the global structure (coarser). This addition made for a marked increase in performance by simply up-sampling earlier layers and summing with latter layer outputs for a final image prediction.

2.5 Instance Segmentation

Instance segmentation is a problem that combines research in Sections 2.3 & 2.4, it aims to not only perform pixel-wise segmentation of classes but also determine instances of objects within and image. As discussed in Section 2.5.1 there is a simple method of combining these networks into one simple classification pipeline.

2.5.1 Mask R-CNN

Mask R-CNN was proposed by Facebook AI Research (FAIR) namely Kaiming He et al. in March 2017. It uses the Faster R-CNN network described in Section 2.3.3 which builds from advances in Sections 2.3.1 & 2.3.2. The method works by simply adding a branch from the existing network for predicting segmentation masks as in Section 2.4 (Figure 7). The branch is a small FCN that is applied to each ROI for segmentation. There are a number of slight modifications to each approach to produce the final Mask R-CNN that are documented below.
RoIs are produced from the RPN as in Faster R-CNN [17] then an $m \times m$ mask is predicted from each RoI without the use of a fully-connected (fc) layer and therefore keeps spacial information as in

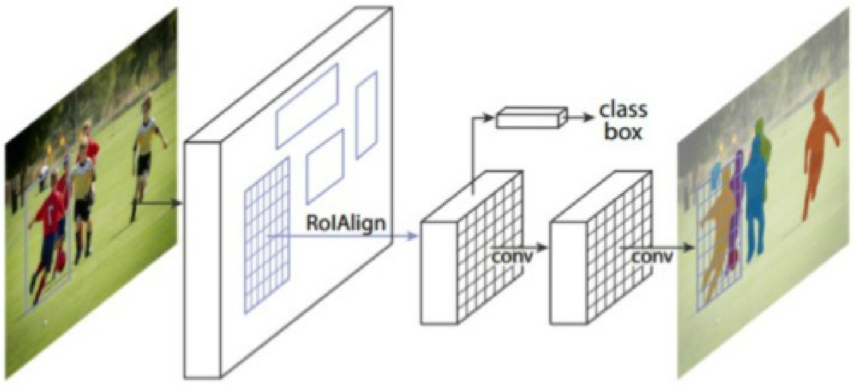


Figure 7: Mask R-CNN [23]

Section 2.4. It was found during experimentation that it is preferable to decouple the class classification and the mask production of the network. This stops multiple classes disrupting each-other during mask production, hence a mask is produced for each class. The class of each respective mask is then determined from the class output branch and that binary mask is selected.

The loss function for the network must now incorporate the three different branches; mask, class prediction, bounding-box prediction (Equation 6).

$$L = L_{cls} + L_{box} + L_{mask} \tag{6}$$

Where L_{cls} & L_{box} are identical to Section 2.3.2. The mask branch loss is determined by a per-pixel sigmoid between the ground-truth class (k) and the corresponding kth binary mask (excluding other classes) and is calculated with an average binary cross-entropy loss.

RoI pool as discussed in Section 2.3.2 is modified in this approach. Instead of subdividing an RoI into discrete regions e.g. a grid of 4×5 cells. An RoI does not always divide equally into these grid cells and so rounding of grid sizes is needed, this quantisation of sub-windows can cause misalignments of the RoI and extracted features. The new RoIAlign layer introduced in this paper mitigates this problem by not rounding grid-cells. Mask R-CNN uses bi-linear interpolation (linear interpolation with 2d) to compute the exact values of input features at four regularly samples locations in each RoI window.

3 Background, SLAM

In order to detect collisions between the bike and obstacles the software must have an understanding of the global environment. The detection described in Section 2 allows mapping, and tracking of dynamic objects in the frame, but distance, path prediction and eventually collision prediction rely strongly on an accurate understanding of 2.5-3D space.

3.1 Sensors

The sensors used in this project are an RGBD camera and an IMU. These are both available in the Intel Realsense Camera (ZR300, Figure 8). This camera has a USB cable that both provides the power and sends camera information. An extension cable allows for a laptop to be attached to the camera, from a backpack whilst cycling. Potential problems with this setup are due to the potential juddering of the bike during operation. This could cause tricky footage to analyse and can cause data loss when the hard-drive is writing to disk. The later issue can been easily solved by writing to an SSD while cycling.

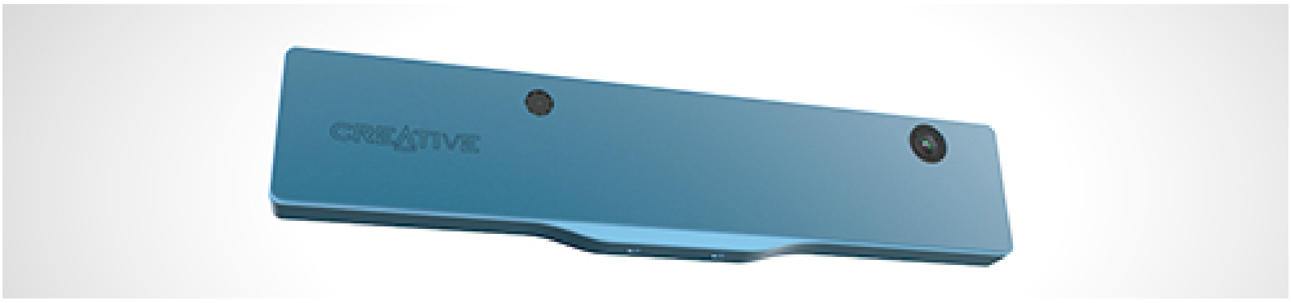
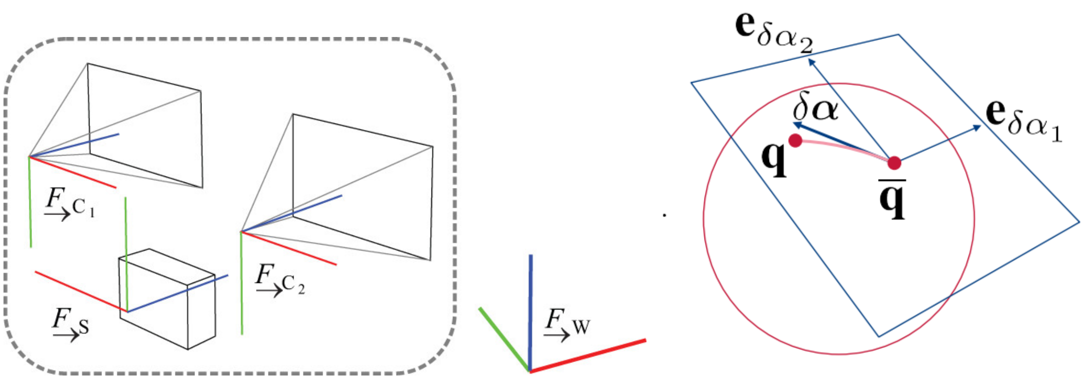


Figure 8: Intel Realsense Camera (ZR300)

3.2 World and Camera Frame

In order to fully understand the bike's location, the environment can be considered from two frame's of reference, the world and camera frame.



$\underset{\to}{F}_{C_1}$
$\underset{\to}{F}_{S}$
$\underset{\to}{F}_{C_2}$
$\underset{\to}{F}_{W}$
$\mathbf{e}_{\delta\alpha_2}$
$\delta\boldsymbol{\alpha}$
$\mathbf{q}$
$\bar{\mathbf{q}}$
$\mathbf{e}_{\delta\alpha_1}$

Figure 9: World vs Camera Frame [24]

To convert between world $(\overset{F}{\longrightarrow}_W)$, camera frame $(\overset{F}{\longrightarrow}_{C_i})$, and IMU frame $(\overset{F}{\longrightarrow}_S)$, an appropriate method of describing the orientation, and location of an object in 3D space is needed. A Hamiltonian Quaternion is capable of correctly mapping an object in 3D space, it does this by use of a 4D representation of one real part, and 3 imaginary;

$$\begin{gathered} q = q_w + q_x i + q_y j + q_z k, \\ i^2 = j^2 = k^2 = ijk = -1 \end{gathered} \tag{7}$$

A small rotation vector can easily be converted to quaternion, think of a small step $\delta textbf f\alpha \in R^3$on a 2D plane mapped to a 3D sphere (Figure 7, Equation 8. The Vectors expressed within a frame are denoted as $\mathbf{p}_A$ or $\mathbf{p}_A^{BC}$ where B and C are start and end points. A transformation between frames is represented as $\mathbf{T}_{BA}$ where B and A represent reference frames, so $\mathbf{T}_{WC}$ converts between the world frame and camera frame.

$$\delta textbf f q(\delta textbf f\alpha) = \begin{bmatrix} \sin \|frac\delta textbf f\alpha 2\| frac\delta textbf f\alpha 2 \\ \cos \|frac\delta textbf f\alpha 2\| \end{bmatrix}^{\oplus} \tag{8}$$

$$textbf f\bar{q}^{k+1} = \delta textbf f q(\Delta textbf f\alpha) \bigotimes textbf f\bar{q}^{k} \tag{9}$$

3.3 The SLAM Problem

If measurements were exact, mapping of an environment would be easy. This, however is not the case, there is often noise associated with each taken measurement and movement. For example distance measurements and IMU readings are subject both to a noise and a bias (calibration error) which can both be assumed to be Gaussian distributed.

$$\tilde{z} = \mathbf{b}_c + s\mathbf{M}_z + \mathbf{b} + \mathbf{n} + \mathbf{o} \tag{10}$$

where $\mathbf{z}$ is the correct measurement, $\mathbf{b}_C$ is a long-term constant bias, s is scaling, $\mathbf{M}$- misalignment, b- time varying bias, n - noise, and o is other un-modelled influences. Therefore the kinematics of the IMU can be modelled as:

$$\begin{aligned} \dot{\mathbf{p}}_W^{WS} &= \mathbf{v}_W^{WS}, \\ \dot{\mathbf{q}}_{WS} &= \frac{1}{2}\Omega(\tilde{\omega}_S^{WS}, \mathbf{w}_g, \mathbf{b}_g)\mathbf{q}_{WS}, \\ \dot{\mathbf{v}}_W^{WS} &= \mathbf{C}_{WS}(\tilde{\mathbf{a}}_S^{WS} + \mathbf{w}_a - \mathbf{b}_a) + \mathbf{g}_W, \\ \dot{\mathbf{b}}_g &= \mathbf{w}_{bg}, \\ \dot{\mathbf{b}}_a &= -\frac{1}{\tau}\mathbf{b}_a + \mathbf{w}_{ba}, \end{aligned} \tag{11}$$

where:

$$\Omega(\tilde{\omega}_S^{WS}, \mathbf{w}_g, \mathbf{b}_g) = \frac{1}{2}\begin{bmatrix} \tilde{\omega}_S^{WS} + \mathbf{w}_g - \mathbf{b}_g \\ 0 \end{bmatrix}^{\oplus} \tag{12}$$

The elements $\mathbf{p}_W^{WS}$,$\mathbf{v}_W^{WS}$ are the position and velocity, $\dot{\mathbf{p}}_W^{WS}$ represents the derivative. $\mathbf{q}_W^{WS}$ is the quaternion body orientation, $\mathbf{b}_g$ & $\mathbf{b}_a$ the accelerometer biases, $\tilde{\mathbf{a}}_S^{WS}$ accelerometer measurements, $\mathbf{g}_W$ earth's gravitational acceleration vector, and finally w:= $[wg^T, wa^T, wbg^T, wba^T]$ are independent Gaussian white noise processes with zero mean.

This means that no measurement can be trusted exclusively for state-space information. When moving therefore, it was first assumed that errors have a compounding effect. So the position of a bike, would become increasingly unknown as the environment is explored and the bike is displaced from the origin. This, however is not the case as proved by Durrant-Whyte **??** since there is a high degree of correlation between estimates of different landmarks and locations in a map, these correlations also increase with successive observations (Figure 10). As an example of this (Figure 10 : Right) displays error in measurements of landmark observation, the mean error is common between all landmarks and so with successive landmark observations the errors in location estimates become highly correlated. This means that although $\mathbf{m}_i$'s location may be quite uncertain but the relative location between two landmarks $\mathbf{m}_i - \mathbf{m}_j$ may be relatively certain.

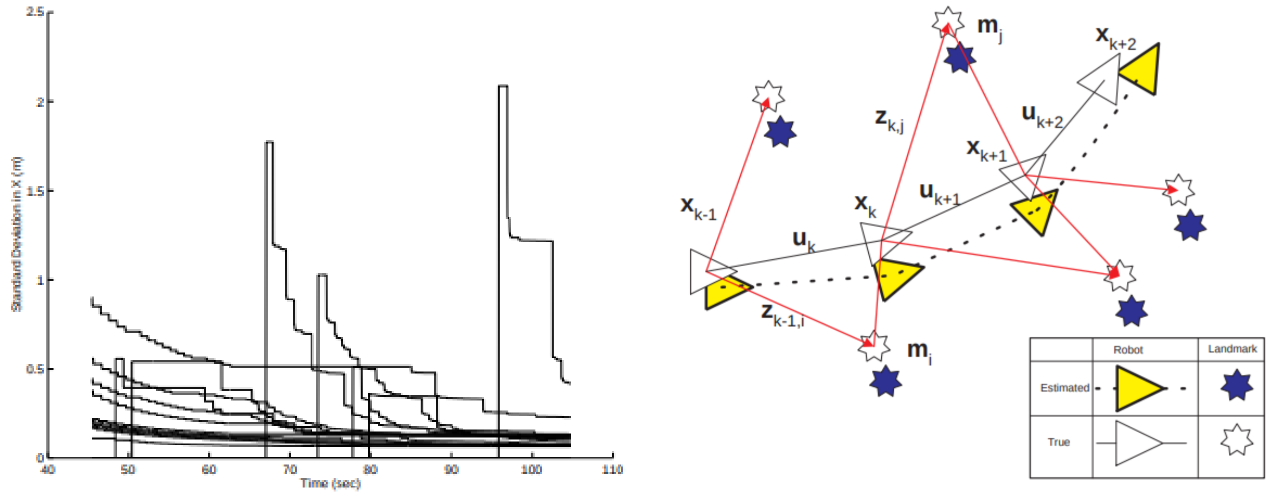


Figure 10: Left: Graphical depiction of convergence in landmark uncertainty. Time history of standard deviations of a set of landmark locations. Over time, standard deviations reduce monotonically to a lower bound [25]. Right: Error between estimated and true landmarks is mostly common between landmarks.

References

[1] D. F. Transport, "Table ras30001: Reported road casualties by road user type and severity, great britain, 2016," report, Dft, 2016.

[2] D. F. Transport, "Ras50001: Contributory factors in reported accidents by severity, great britain, 2016," report, Dft, 2016.

[3] D. F. Transport, "Focus on cycling in reported road casualties great britain 2013," report, Dft, 2013.

[4] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[5] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," vol. 25, 01 2012.

[7] A. Karpathy, "Networks," 2017.

[8] A. Karpathy, "Networks," 2017.

[9] A. Karpathy, "Networks," 2017.

[10] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *CoRR*, vol. abs/1710.09829, 2017.

[11] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013.

[12] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[13] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, pp. 154–171, Sep 2013.

[14] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, pp. 167–181, Sep 2004.

[15] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *CoRR*, vol. abs/1406.4729, 2014.

[17] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[18] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014.

[19] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," *CoRR*, vol. abs/1604.01685, 2016.

[20] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pp. III–1058–III–1066, JMLR.org, 2013.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[22] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[23] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017.

[24] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visualinertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[25] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): Part i," 10 2006.