Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Hazard Detection for Cyclists

Author:
Peter Hedley

Supervisor:
Dr Stefan Leutenegger

Submitted in partial fulfillment of the requirements for the MSc degree in Computing of Imperial College London

September 2015

Abstract

Recently in the news, and as any London cyclist has experienced, there are numerous potentially fatal hazards on a morning commute. Cars turn directly in front of you, doors often open unexpectedly and pedestrians amble into the road taking a selfie. This project is aimed at detecting potential hazards through the use of an on-board camera, and alerting the cyclist and pedestrian/car to it. The problem requires computer vision techniques for detection and SLAM algorithms for spacial understanding.

This project investigates state-of-the-art methods in image segmentation, object tracking, and develops a novel architecture for cyclist path prediction. The results of which, although not real-time provide a solid foundation for further development, and suggest potential directions of future research.

Acknowledgments

Many thanks to my supervisor Dr. Stefan Leutenegger for his continued support throughout the process. As a student with less than a year of computer science experience, this project would not have been possible without him.

Also thanks to the other students on my course who have helped with ideas about how the project should progress.

Contents

Chapter 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, and could save lives.

1.0.1 Road Map

The report develops a system for collision detection and is split into two main sections, object detection and understanding the 3D environment. The sections are developed simultaneously throughout the report and are discussed within the following chapters.

Background This chapter discusses the current research into the area of instance segmentation, recurrent networks, optimisation, and SLAM. It first describes recent advances in computer vision relevant to the current state-of-the-art method Mask RCNN. Mask RCNN combines a number of advances in different fields, namely image classification, detection, and segmentation. The chapter then continues with discussion around recurrent networks for path prediction, and methods of optimising these networks. Finally a brief discussion on SLAM and object tracking which are both highly active and voluminous fields of research in their own right.

Contribution The contribution chapter discusses depth approximations when there is a depth camera failure, object tracking, and prediction. In particular, it discusses combining various tracking methods into a comprehensive system that both combines the advantages of the respective algorithms and mitigates their potential weaknesses. Once the ability to track objects is discussed, models for prediction of future locations are developed. This leads to a novel recurrent network to predict future bicycle locations from image, and pose data gathered from previous frames. Finally, a method of combining these predictive models, and determining whether a collision is likely to occur is discussed in the latter part of the chapter. These sections all talk through the various modules required in the system which is described below in Figure 1.1

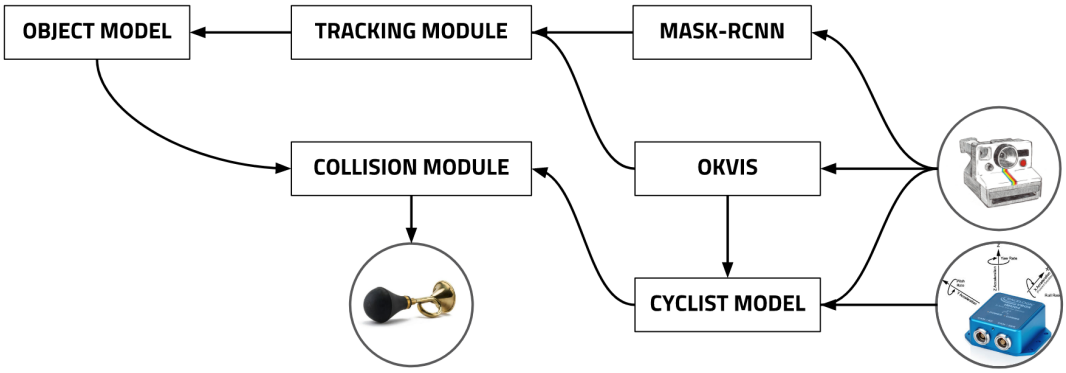

Figure 1.1: System Design

System Evaluation This section describes the potential problems with the system and includes an image sequence from a staged hazard. The final system can be seen to detect a crossing pedestrian and determine whether a collision is likely.

Conclusion This chapter discusses and summarises the system, it also suggests further areas of research for the system to be taken further.

Ethics The final section discusses any ethical issues within the project and contains a completed ethical check-list

Chapter 2

Background

2.1 Background, Computer Vision

2.1.1 Deep Learning Preliminaries

Computer vision and its applications have been revolutionised recently by Deep Learning frameworks. These have led to super-human accuracy in image recognition, for example Inception v3 [4], released in 2016, reached a 3.46% error rate on the ImageNet [5] dataset. This is known 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) 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 that can be carried on a bicycle, it will be of concern in this report.

2.1.2 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 which was 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.

A regular neural network, as mentioned earlier Section 2.1.1, 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 2.1). In this architecture, unlike in the fully connected case, there is a significant reduction in the parameter space [7].

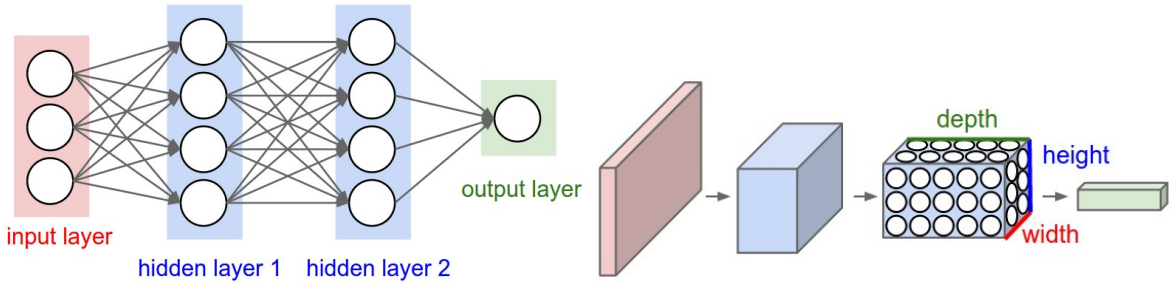

Figure 2.1: Regular Neural Network (Left) [7] , CNN (right) [7].

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 the dot product is taken between the filter values and the inputs to that filter (Figure 2.2). The filter slides across with a respective stride, which is a hyper-parameter, 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 are created, relating to the layer depth which is a tune-able hyper-parameter.

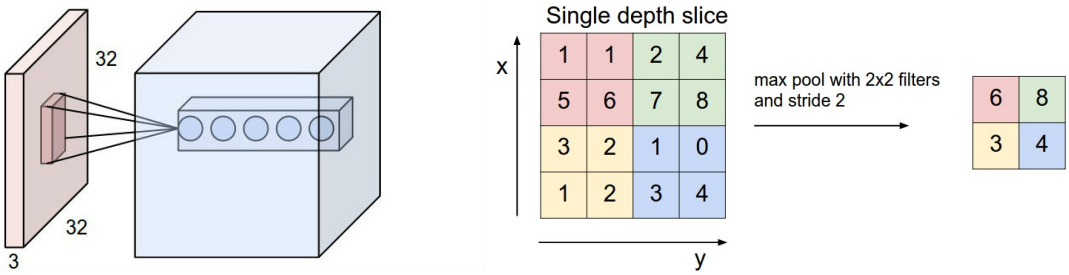

Figure 2.2: CNN Filter (left) [8], Max Pooling [9]

Pooling Layers

Pooling layers are used to reduce the number of parameters in a network. This is a much debated topic, Geoffrey Hinton discusses the problems of convolutional networks in his 2017 paper [10] 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 2.2) 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. This process reduces the input size with respect to the kernel size and the implemented stride.

2.1.3 Image Detection

Image detection is an area of research that uses the CNN networks described in Section 2.1.2. 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.

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 passed through a large CNN, and each region is classified using class-specific linear SVMs. This process equates to a 53.7% mean average precision (mAP) compared to 35.1% by state-of-the-art methods 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 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 during the growth period and also limit one region's growth at the expense of others (hence the size metric).

The results of the region proposal (Figure 2.3) 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 dimension's of the CNN network that follows, containing five convolutional layers and two fully connected layers. The output of the CNN is then fed as a feature vector to a set of trained SVMs, one for each class, that is 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 time-scale. A later iteration of this method Fast-RCNN is discussed in Section 2.1.3.

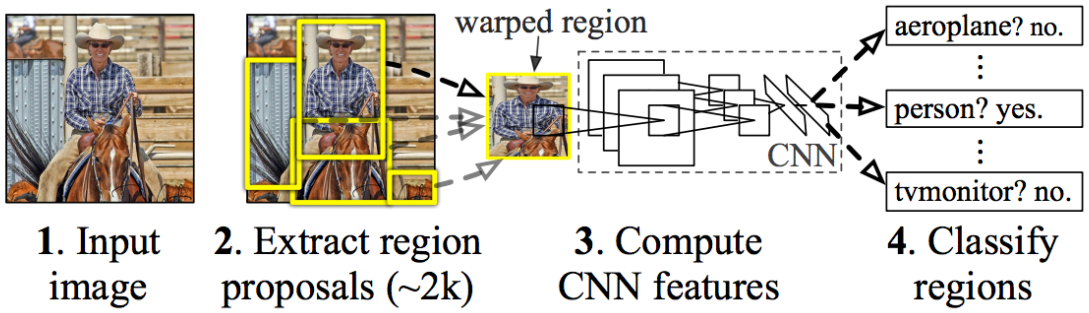

Figure 2.3: R-CNN Region Proposal & Architecture [11]

Fast-RCNN

R-CNN performs a CNN forward pass for each region proposal and does not share computation between regions [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 classifying 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 2.4). 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 2.4 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 2.1).

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v) \tag{2.1}$$

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:

$$L_{loc} = \sum_{i \in \{x,y,w,h\}} smooth_{L_1}(t_i^u - v_i), \tag{2.2}$$

in which

$$smooth_{L_1} = \begin{cases} 0.5x^2 & \text{if } x < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{2.3}$$

is a L_1 loss. This architecture allows for fast training with the forward pass of the network and therefore processes images 146$\times$ faster than R-CNN due to sharing computation between layers and the improved pipeline design.

Figure 2.4 depicts the network architecture, this incorporates a 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 hyper-parameters Each RoI is a rectangular window defined by a tuple of four elements (r, c, h, w), where (r, c) specify

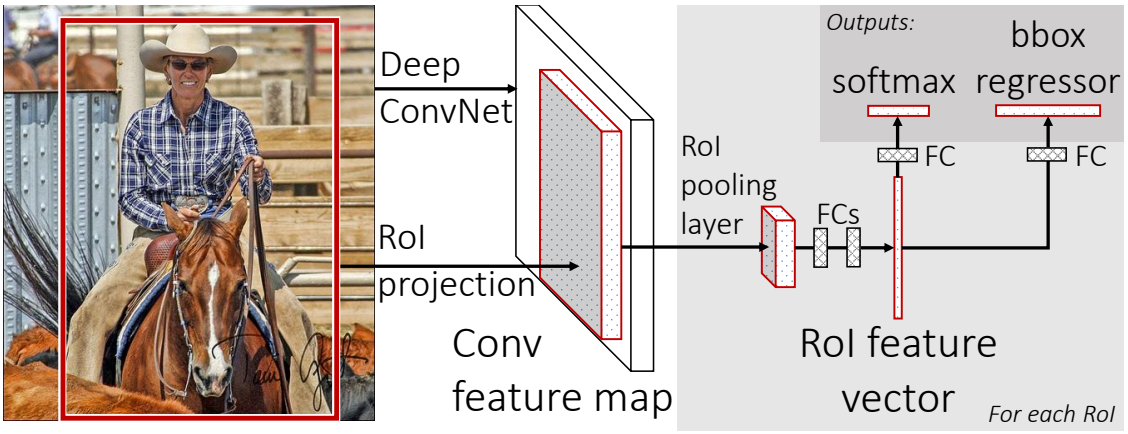

Figure 2.4: Fast R-CNN Architecture [15]

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.

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.

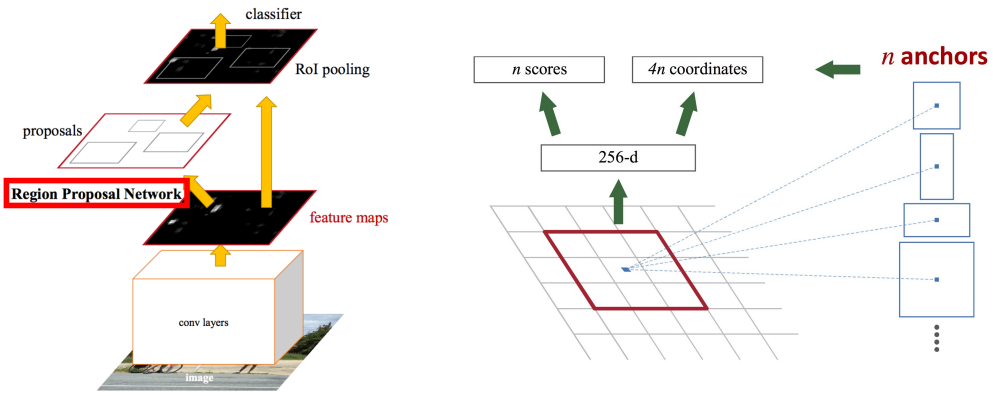

Figure 2.5: Fast R-CNN Architecture [15]

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 by sliding a smaller network of size $n \times n$ over the convolutional feature map (Figure 2.5). 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 represents 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 cls has $2k$ outputs. This results in a convolutional layer output of depth nine.

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 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:

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(pi, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{2.4}$$

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:

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

where x, y, w, and h are the box centre coordinates plus width and height. Predicted box, and 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 suppression is used on their cls score with an IoU threshold of 0.7 to leave around 2000 proposals per image. Non-maximum suppression 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.

2.1.4 Image Segmentation

Unlike the image detection problem described in Section 2.1.3, this task aims to segment an image on a pixel-by-pixel basis into respective classes (Figure 2.6). 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 heat-map output (Figure 2.6).

The heat-maps produced from the network then need to be up-sampled from their respective dimensions to that of the output image. This process is essentially a reverse convolution or

Figure 2.6: Image Segmentation [19], Convolutional Heat Map [18]

deconvolution. Up-sampling, therefore, with factor f is convolution with a fractional input stride of $\frac{1}{f}$. Essentially each filter is 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 patch-wise 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 Drop-connect-like mask [20] between the output and the loss. A Drop-connect 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), Long at al. [18] go on to design a bespoke network that achieves 62.7 mean IU. Mean pixel intersection over union (Mean IU) is a standard performance metric where the 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.

2.1.5 Instance Segmentation

Instance segmentation is a problem that combines research in Sections 2.1.3 & 2.1.4, it aims to not only perform pixel-wise segmentation of classes but also determine instances of objects within and image. As discussed in Section 2.1.5 there is a simple method of combining these networks into one classification pipeline.

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 2.1.3 which builds from advances in Sections 2.1.3 & 2.1.3. The method works by simply adding a branch from the existing network for predicting segmentation masks as in Section 2.1.4 (Figure 2.7). The branch is a small FCN that is applied to each RoI for segmentation. There are some slight modifications to each approach to produce the final Mask R-CNN that are documented below.

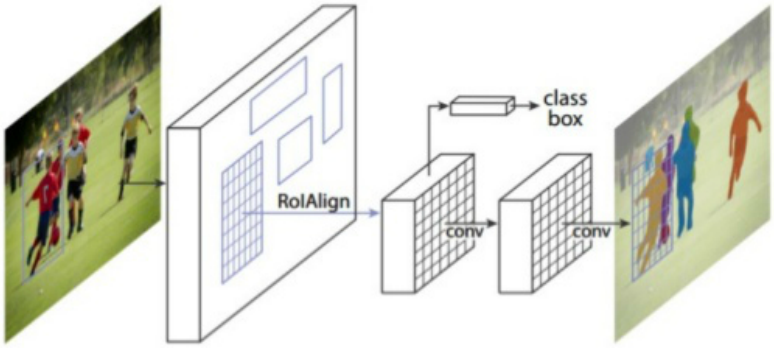

Figure 2.7: Mask R-CNN [23]

RoIs are produced from the RPN as in Faster R-CNN [17] then a $m \times m$ mask is predicted from each RoI without the use of a fully-connected (fc) layer and therefore keeps spacial information as in Section 2.1.4. 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 optimisation of the loss function, 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, and bounding-box prediction (Equation 2.5).

$$L = L_{cls} + L_{box} + L_{mask} \tag{2.5}$$

where L_{cls} & L_{box} are identical to Section 2.1.3. 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 2.1.3 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 2 dimensions) to compute the exact values of input features.

This network architecture is the final piece required to segment the images provided by the camera and is the model used in the cycling system. The next section will go on to talk about the other networks used for things such as position prediction.

2.1.6 Recurrent Neural Networks (RNN):

Traditional RNN Layer: A normal dense network layer has no memory of the previous features that pass through it. This would mean that in order to classify a sequence the data must be passed through the layer simultaneously. The advantage of a recurrent layer its ability to remember. A recurrent layer has a weighted channel that inputs the output of the previous state as part of the next, see Figure 2.8 for more details. Each state $\{1, 2, ..., t-1, t\}$ is connected almost like a chain.

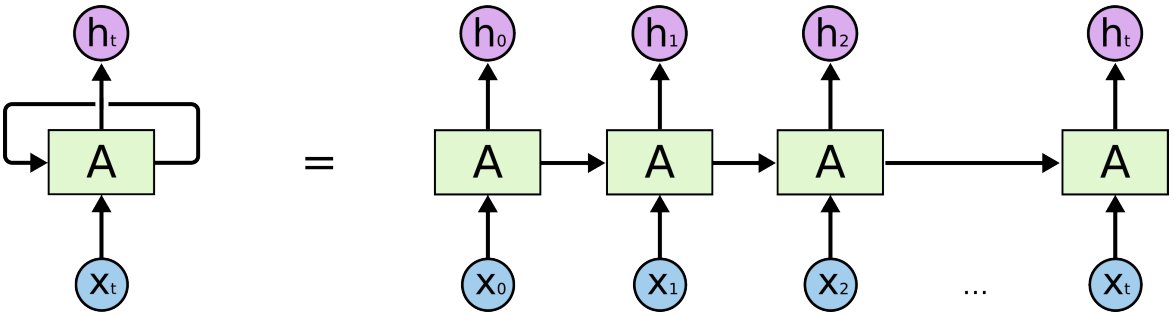

Figure 2.8: A Recurrent Layer [24] . The output of the cell at time t, from inputs x_t is h_t

The main issue with this recurrent layer is how far back the network is capable of remembering. In short sequences the information will pass effectively through the chain, but often in longer sequences the network is less capable of remembering previous states.

Long-Short Term Memory (LSTM) Layer

LSTM layers are far better at remembering, and do not get the vanishing gradient problem that RNN layers are subject to. LSTMs contain a cell state which passes through the layer, a forget, input, and output gate. These gates determine whether to modify the cell state of the layer; it acts as the memory of the later and is similar to the links in the RNN chain (Figure 2.8). The gates contain sigmoid functions which can evaluate between to 0 and 1, and determine how much of a component is to be let through.

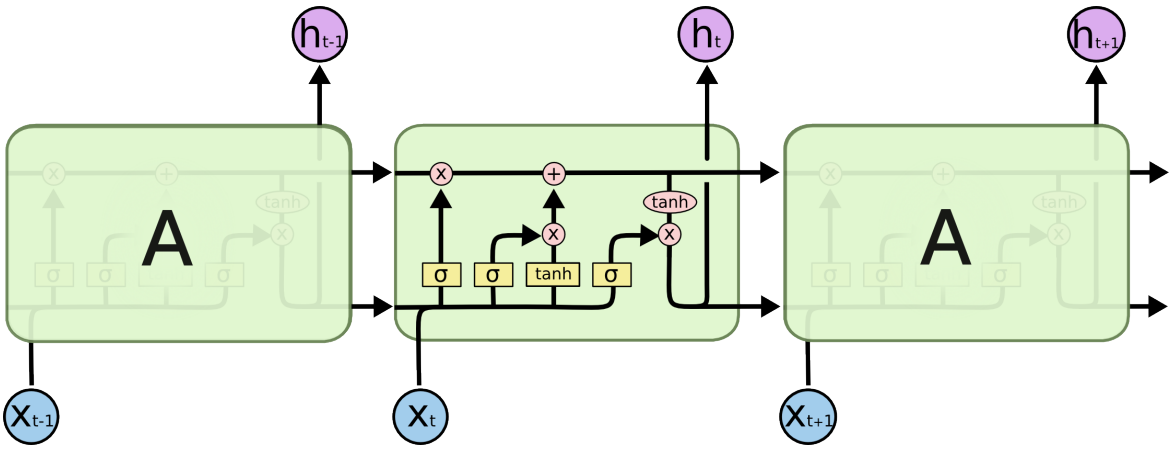

Figure 2.9: A LSTM (Long-Short Term Memory) Layer [24]

Forget Gate: The forget gate determines how much of the cell state to keep by looking at h_{t-1} and x_t. The gate simply acts like a perceptron, multiplying inputs by a weight, adding a bias and then passing the result through the activation (Sigmoid) function (Equation 2.6).

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.6}$$

Once again h_{t-1} is the output of the layer from the last state, x_t is the layer input at time t.

Input Gate: When the gate has determined what to forget from the last state it needs to decide what to add from the new state x_t for a correct prediction h_t. This introduces the next gate in the system which combines an input sigmoid gate that decides which values of x_t to add (Equation 2.7), and a $tanh$ section that creates the new vector $\tilde{C}_t$ to add to the current state C_t, Equation 2.8.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.7}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.8}$$

The input gate and forget gate then influence the state via weighting C_{t-1} by the forget metric f_t and $\tilde{C}_t$ by the input scale, Equation 2.9.

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \tag{2.9}$$

Output gate: The final step of a LSTM is to decide the output. This involves deciding which parts of the cell state to output and scaling to (-1,1) via the use of another $tanh$ function, Equations 2.10, 2.11.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.10}$$

$$h_t = o_t \times tanh(C_t) \tag{2.11}$$

Summary: The LSTM layer is highly effective at predicting sequences, it is used later within the system to predict where the bike will be in the future. It provides an effective and sufficiently complex method of modelling bicycle movement including determining from picture inputs the most likely path along a road that the bike will take. Now that the networks have been defined, they must be optimised to perform well at their respective tasks. Mask-RCNN is kept as default in this report but the prediction models architecture is bespoke and thus must be optimised for its use-case.

2.1.7 Bayesian Optimisation of Layer Dimensions:

In order to get the best network architecture, and hyper-parameters such as learning rate, it is possible to naively try combinations and record which performs the best. However, when trying to optimise networks efficiently it is better to provide a range of possible working parameters and optimise them in an informed and, more importantly, automated manner. Bayesian optimisation allows such a method and can quickly check the parameter space for a good combination to minimise the network loss. Bayesian optimisation uses an acquisition function which evaluates a Gaussian process. It is cheaper to evaluate a proxy function that approximates the true function rather than run the neural network. Hence it is a very fast and effective method of optimisation.

Gaussian Processes: In a typical regression sample, the aim is to determine the parameters to a function (or family of functions) that best describe some data. A Gaussian process involves making the assumption that a function can be described simply by an infinite number of points $f(x) = [f_1, f_2, ...f_\infty]$, a finite number of which are Gaussian distributed. The aim is to then best approximate the relationship between these points via a covariance kernel $k(\cdot, \cdot)$ and a mean function $m(\cdot)$. A Gaussian distribution is described by Equation 2.12 and a kernel function (Matern) by Equation 2.13.

$$p(x|\mu, \Sigma) = (2\pi)^{\frac{D}{2}} |\Sigma|^{-\frac{1}{2}} e^{(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu))} \tag{2.12}$$

$$k_{Mat,3/2}(x_i, x_j) = \sigma_f^2 \left(1 + \frac{\sqrt{3}||x_i - x_j||}{l}\right) e^{\left(-\frac{\sqrt{3}||x_i - x_j||}{l}\right)} \tag{2.13}$$

where l is a hyper-parameter that describes the smoothness of the data and σ_f the amplitude of the latent function. An example of a Matern kernel is given in Figure 2.10.

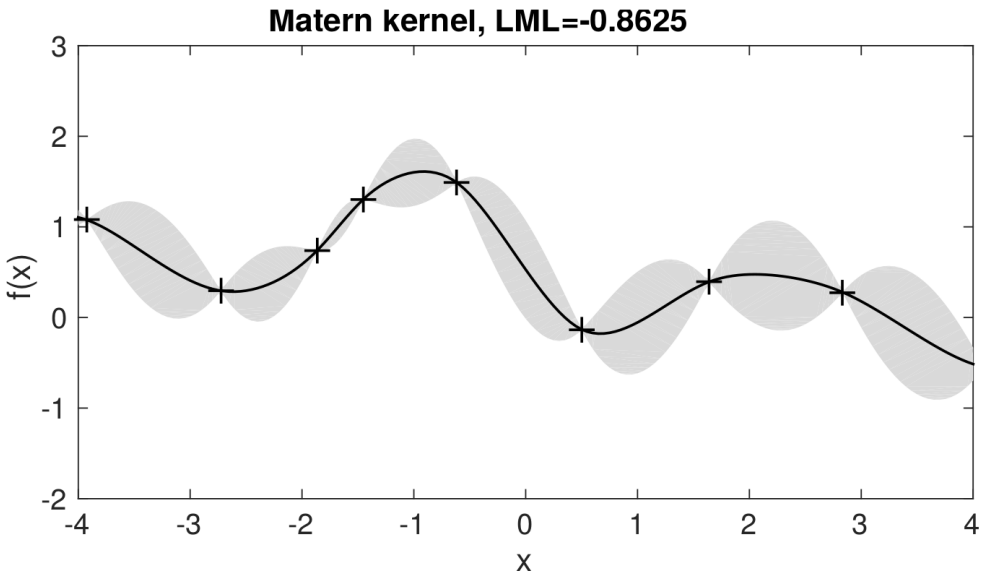

Figure 2.10: Gaussian Process: Matern Kernel [25]

Bayesian Optimisation: Bayesian Optimisation is a process that can maximise a set of hyper-parameters with a reduced evaluation of the true function. It uses an acquisition function that balances state exploration and exploitation i.e. how much to search local minima for the best solution vs exploring for global minima. The true function (NN model) is evaluated and data collected to initialise the process. Then the acquisition function determines where to evaluate next in the true function by querying the proxy function. An example acquisition function, the lower confidence bound [26] is presented in Equation 2.14.

$$\alpha(\mathbf{x}) = -(\mu(\mathbf{x}) - (k)\sigma(\mathbf{x})) \text{ , } k > 0 \tag{2.14}$$

The lower confidence bound balances exploration of global minima by the covariance function and weighting k, which increases at each iteration, with exploitation of local minima with the mean function. The Bayesian optimisation process roughly translates to:

Algorithm 1: Algorithm for Bayesian Optimisation

update_matched_rois ($old\_matched, new\_matched, old\_frame, new\_frame$)

Input :

$parameters$ // A list of parameters to optimise

$parameter\_limits$ // A list of ranges to optimise the parameters within

$function$ // A pointer to the function to optimise

Output:

$parameters$ // The optimal parameters

// Note that if there are no objects in old frame $new\_frame.id$ = zeros

$D = [\{x_0, y_0\}]$ /* Initialise the dataset */

foreach $step \in iterations$ **do** /* For every not-matched new RoI */

  $update\_gp(D[step - 1])$ /* Update the Gaussian process with data from the last step */

  $x_{step} = argmax(\alpha(x), parameter\_limits)$ /* Evaluate the acquisition function */

  $y_{step} = function(x_{step})$ /* Evaluate true function (NN model) */

  $D.append(\{x_{step}, y_{step}\})$ /* Add data to dataset */

end

$return(X_{best})$ /* Return optimal parameters */

2.2 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 2.1 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.

2.2.1 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 2.11). 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 set-up are due to juddering of the bike on the road; this could cause tricky footage to analyse and data loss when the hard-drive is writing to disk. The latter issue can easily be solved by writing to an SSD while cycling.

Figure 2.11: Intel Realsense Camera (ZR300)

Camera Distortion

In order to fully understand the bike's location, the environment can be considered from two frames of reference, the world and camera frame. The camera, however, cannot be assumed to produce a perfect image and is hence prone to distortion - some types of camera distortion are shown in (Figure 2.12).

A well-known distortion model is "radial-tangential" distortion. As the name implies there is both a radial and tangential component to the distortion (Equation 2.16)

$$r^2 = x_1^{'2} + x_2^{'2} \tag{2.15}$$

$$\mathbf{x}'' = \mathbf{d}(\mathbf{x}') = \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} \begin{bmatrix} x_1' \\ x_2' \end{bmatrix} + \begin{bmatrix} 2p_1 x_1' x_2' + p_2(r^2 + 2x_1^{'2}) \\ p_1(r^2 + 2x_2^{'2}) + 2p_2 x_1' x_2' \end{bmatrix} \tag{2.16}$$

where x_1', x_2' is the 3d point mapped to the unit plane in z, $k_1 - k_6$ are radial distortion parameters, p_1, p_2 are tangential distortion parameters. The model parameters can be determined during camera calibration.

Figure 2.12: No Distortion (left)[27], Positive/negative radial distortion (middle/right)

2.2.2 Camera to World Frame

Calculating the 3D Ray

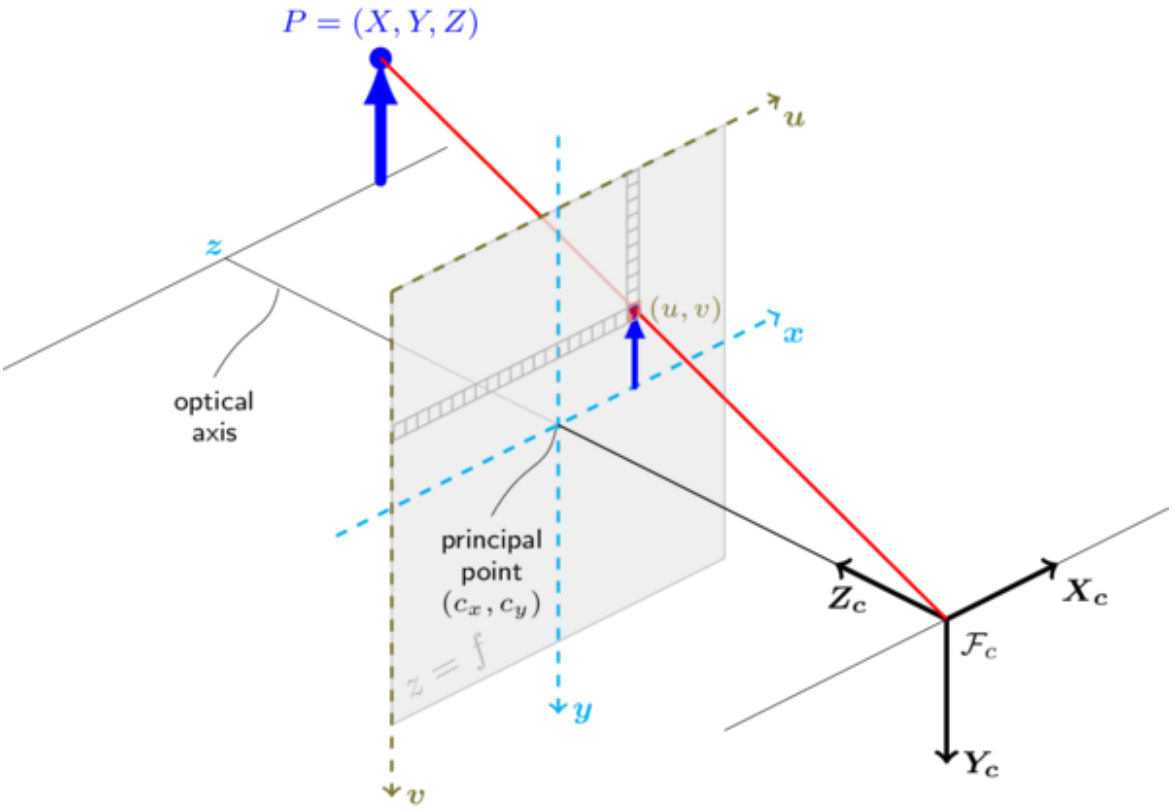

Figure 2.13: Pinhole Camera Model

Once an object or a point is detected in an image, it is important to transform it into the world frame; to track and gain a 3d appreciation of the environment. For visual understanding and context of the problem see the pinhole camera model in Figure 2.13. The only difference, in

practice, is an intermediate un-distorting step to get the point's ray. This is due to the focal lens in the camera, which is not represented in the pinhole model, but Equation 2.16 shows how to account for this. Hence to find a ray describing a 3d point, the image must first be scaled, (Equation 2.17)

$$\mathbf{x''} = \mathbf{k^{-1}}(\mathbf{u}) = \begin{bmatrix} \frac{1}{f_1} & 0 \\ 0 & \frac{1}{f_2} \end{bmatrix} \left(\mathbf{u} - \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \right) \tag{2.17}$$

where f_1, f_2 are x/y focal lengths in pixels, and c_1, c_2 is the principal point (image centre) in pixels. The point can be projected to its undistorted location using the reverse of Equation 2.16.

$$\mathbf{x'} = \mathbf{d^{-1}}(\mathbf{x''}) \tag{2.18}$$

and finally the ray is calculated by remembering that the z component is of length 1.

$$\mathbf{x} = \begin{bmatrix} \mathbf{x'} \\ 1 \end{bmatrix} \tag{2.19}$$

Calculating the 3D Location

Now that the ray has been determined we have a point on the 3D unit plane of $z = 1$. To find the 3D point in the camera frame, a depth measurement, obtained from the depth camera at the same (undistorted pixel location) is required. Hence the final 3D point is:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x'} \times depth(\mathbf{x'}) \\ depth(\mathbf{x'}) \end{bmatrix} \tag{2.20}$$

Assuming a static camera location, this would be all that is needed to map the space in-front of the camera. However, as the camera moves, the origin, and orientation of the camera frame changes with respect to the world frame. Hence the software must not only determine the 3D location of a point in the camera frame but also translate that to a world frame representation.

2.2.3 Projecting to World Frame

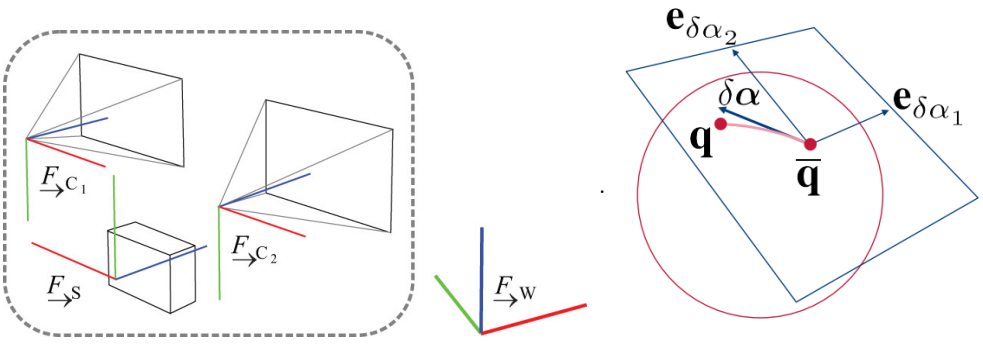

Figure 2.14: World vs Camera Frame (left)[28], Hamiltonian Quaternion (right)

To convert between world ($\overset{F}{\rightarrow}_W$), camera frame ($\overset{F}{\rightarrow}_{C_i}$), and IMU frame ($\overset{F}{\rightarrow}_S$), 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 the orientation of an object in 3D space, it does this by use of a 4D representation of one real part, and 3 imaginary parts;

$$\begin{aligned} q &= q_w + q_x i + q_y j + q_z k, \\ i^2 &= j^2 = k^2 = ijk = -1 \end{aligned} \tag{2.21}$$

Further details of a Quaternion, and how to map to/from the quaternion space can be found in any robotics textbook. With this extra representation of orientation it is possible to map the camera frame to the world frame. Given a position vector in the camera frame $p_c = [x, y, z, 1]$, it can be converted as follows:

$$\begin{aligned} T_{WC} &= T_{WS} \cdot T_{SC} \\ p_W &= T_{WC} \cdot p_C \end{aligned} \tag{2.22}$$

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. It is composed of a position and orientation transformation:

$$T_{WC} = \begin{bmatrix} C_{WC} & r_{WC} \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{2.23}$$

where $T_{WC} \in \mathbb{R}^{4\times 4}$, the position translation matrix $r_{WC} \in \mathbb{R}^{3\times 1}$, and the orientation matrix $C_{WC} \in \mathbb{R}^{3\times 3}$. Note also that the inverse of this matrix can be easily calculated as C_{WC} is symmetric and therefore positive definite and it holds that $C_{CW} = C_{WC}^{-1} = C_{WC}^{T}$.

$$T_{CW} = T_{WC}^{-1} = \begin{bmatrix} C_{WC}^{T} & -C_{WC}^{T} r_{WC} \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{2.24}$$

These transformation matrices must be determined from a SLAM algorithm such as OKVIS which is discussed in the following sections.

Camera to World Summary

Algorithm 2: Algorithm to transform camera point to world frame

cam2world $(cameraPoints, focalMatrix, imageCentre, T_{ws}, T_{sc}, cameraModel)$

Input :

$cameraPoints$ // N points in the camera frame $\in R^{3\times N}$

$focalMatrix, imageCentre$ // Focal Length Matrix, Principal Points [x,y]

T_{ws}, T_{sc}, $cameraModel$ // IMU to World/Camera Matrix, Camera Model $\in R^{4\times 4}$

Output: N points in the world frame $worldPoints \in \mathbb{R}^{3,N}$

$x = scale\_point(imageCentre, focalMatrix, cameraPoints)$ /* Equation (2.17) */

$x = undistort(cameraModel, x)$ /* Equation (2.16) */

$T_{wc} = matrix\_multiply(T_{ws}, T_{sc})$

$x = add\_depth(depth, x)$ /* Equation (2.20), and Vstack [depth,1] to bottom of x */

$worldPoints = transform\_to\_world(T_{wc}, x)$ /* Equation (2.22) */

2.2.4 World to Camera Frame

In order to correctly plot the Kalman filter and give other tracking algorithms the expected location on the picture, the ability to transform from world to camera frame is needed. The process is similar to the camera to frame except the reverse, namely:

Algorithm 3: Algorithm to transform world point to camera frame

world2cam $(worldPoints, focalMatrix, imageCentre, T_{ws}, T_{sc}, cameraModel)$

Input :

$worldPoints$ // N points in the camera frame $\in R^{3\times N}$

$focalMatrix, imageCentre$ // Focal Length Matrix, Principal Points [x,y]

T_{ws}, T_{sc}, $cameraModel$ // IMU to World/Camera Matrix, Camera Model $\in R^{4\times4}$

Output: N points in the camera frame $cameraPoints \in \mathbb{R}^{3,N}$

$T_{wc} = matrix\_multiply(T_{ws}, T_{sc})$

$T_{cw} = inverse(T_{wc})$ /* Equation (2.24) */

$x = transform\_to\_camera(T_{cw}, worldPoints)$ /* Equation (2.22) */

$x = distort(cameraModel, x)$ /* Equation (2.16) */

$x = remove\_depth(depth, x)$ /* Reverse Equation (2.20) */

$cameraPoints = scale\_point(imageCentre, focalMatrix, x)$ /* Equation (2.17) */

2.2.5 Bicycle States

In order to find the bike location and orientation in each frame; a complete representation of the bike's state is needed. At each taken image (k) the bike can be assumed to have a set state $\mathcal{X}_R^k$. Contained within $\mathcal{X}_R$ are a set of pose states, $\mathcal{X}_T$, and speed/bias states $\mathcal{X}_{sb}$, Equation 2.25. Landmarks are contained within the set $\mathcal{X}_L$.

$$\begin{aligned} \mathcal{X}_T &:= [{}_W r_S^T, q_{WS}^T]^T \\ \mathcal{X}_{sb} &:= [{}_s v^T, b_g^T, b_a^T]^T \\ \mathcal{X}_R &:= [{}_W r_S^T, q_{WS}^T, {}_s v^T, b_g^T, b_a^T]^T \end{aligned} \tag{2.25}$$

The elements ${}_W\mathbf{r}_S^T$, $\mathbf{q}_W^T$ are the position and the quaternion body orientation, ${}_s\mathbf{v}^T$ the velocity, $\mathbf{b}_g$ & $\mathbf{b}_a$ the accelerometer and gyroscope biases.

2.2.6 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.

$$\tilde{\mathbf{z}} = \mathbf{b}_c + s\mathbf{M}_z + \mathbf{b} + \mathbf{n} + \mathbf{o} \tag{2.26}$$

where $\mathbf{z}$ is the correct measurement, $\mathbf{b}_c$ is a long-term constant bias, s is a scaling factor, $\mathbf{M}$ - misalignment, $\mathbf{b}$ - time varying bias, n - noise, and o contains other un-modelled influences.

Historically, before the recent advances in SLAM, it was first assumed that errors have a compounding effect. The position of a bike, would become increasingly unknown as the environment is explored and the bike is displaced from the origin. This makes intuitive sense if say there is only IMU measurements available. At each step, more uncertainty is introduced from the noise and biases (shown in Equation 2.26) and the bike's position becomes more and more uncertain.

For some time it was also presumed to be the case even when external measurements of the environment are taken. This, however, is not the case as proved by Durrant-Whyte [29] 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 2.15). As an example of this (Figure 2.15 : 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 $\mathbf{m}_i - \mathbf{m}_j$ is well-known.

In the case of the bicycle set-up, there is both an IMU, and a camera. These provide the basic inputs to the OKVIS software [28]. Okvis is able to determine the bike's location by optimising the total error, hence combining both the camera and IMU error with respect to the bicycle state. This optimisation process is discussed in the following sections. Firstly, however, it is important to detect landmarks from one frame to another, which is discussed in the next section.

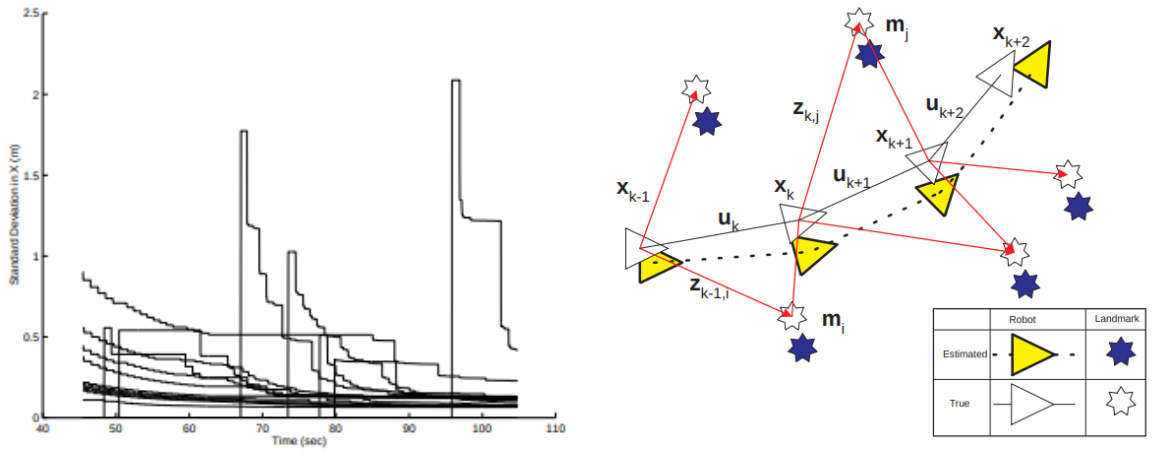

Figure 2.15: Left: Convergence in landmark uncertainty. Over time, standard deviations reduce monotonically to a lower bound [30]. Right: Error between estimated and true landmarks is common.

2.2.7 Frame to Frame Matching

Keypoint Detection The Okvis [28] paper uses an corner detection method to determine keypoints and then matches these keypoints using a binary descriptor (BRISK [31]). In the BRISK paper a modification of the FAST feature extractor is used, however, in the OKVIS implementation a modified Harris Corner detection algorithm is utilised. A simplistic way

to think of this is how one would go about determining a corner in an image. Imagine an edge, it is simply a location on an image where the brightness gradient changes rapidly in one direction (perpendicular to the edge). Hence a corner is a similar change but in two such directions. Imagine a location (x, y) and a small displacement from that location $(x + \Delta x, y + \Delta y)$. An area of rapid gradient change would have very different intensity values at (x, y) and surrounding points. Hence a corner will maximise the sum of squared differences between points.

$$f(x, y) = \sum_{x_k, y_k \in W} (I(x_k, y_k) - I(x_k + \Delta x, y_k + \Delta y))^2 \tag{2.27}$$

This can be further distilled by the Taylor expansion to:

$$f(x, y) = \sum_{(x,y) \in W} (I_x(x, y)\Delta x + I_y(x, y)\Delta y)^2 \tag{2.28}$$

or in matrix form:

$$M = \begin{bmatrix} \sum_{(x,y)\in W} I_x^2 & \sum_{(x,y)\in W} I_x I_y \\ \sum_{(x,y)\in W} I_x I_y & \sum_{(x,y)\in W} I_y^2 \end{bmatrix} = R^T \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R \tag{2.29}$$

$$f(x, y) = (\Delta x\ \Delta y) M \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{y} \end{bmatrix} \tag{2.30}$$

where M is the structure tensor (also commonly known as the Second Moment Matrix). The Harris corner response then is taken from the eigenvalues of M, intuitively since the gradient changes rapidly, the structure tensor eigenvalues are large in that direction Figure 2.16.

$$R = det(M) - ktrace(M)^2 = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{2.31}$$

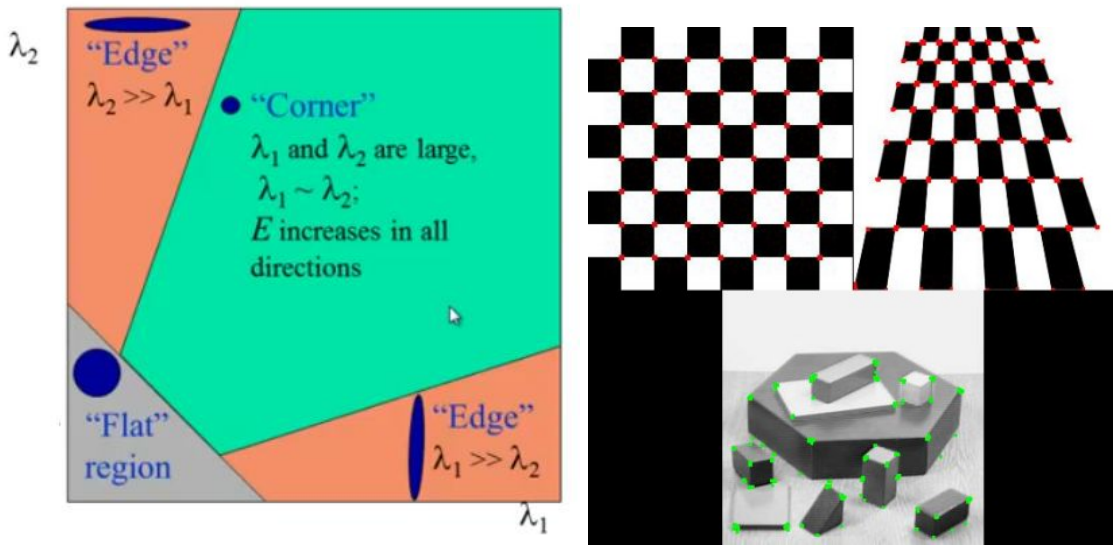

Figure 2.16: Left: Structure Tensor Eigenvalues, Right: Harris corner detection result [32]

Keypoint Matching Binary descriptors use a sampling pattern to create pairs of points (lines). The algorithm then compares the two points $(p1, p2)$ and if $p1 > p2$ a 1 is recorded and otherwise 0. This creates a binary string, that along with orientation representation describe a keypoint. When matching keypoints between images a simple, and fast $\sum XOR$ reveals keypoint similarity from which a threshold can be used to determine matches.

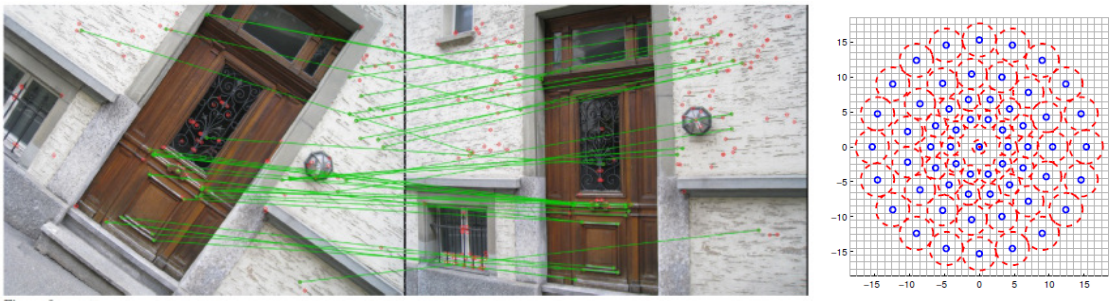

Figure 2.17: Left: Brisk keypoint matching example, Right: Brisk Sampling Pattern [31]

The BRISK sampling pattern (Figure 2.17: Right) uses a standard Gaussian smoothing at each sample point. The standard deviation of the sample is represented by the red circles in the figure. This ensures the sample is less prone to noise than other traditional binary detectors e.g. BRIEF.

Batch Non-linear Least Squares

Okvis formulates and error equation that combines both the IMU error and camera re-projection error. The state of the bike can then be determined by minimising this error function. Given the error function for the camera and IMU in the form:

$$J(\mathbf{x}) := \frac{1}{2}\sum_{n=1}^{N} \mathbf{e}_n(\mathbf{x})^T \mathbf{W}_n \mathbf{e}_n(\mathbf{x}) \tag{2.32}$$

where $\mathbf{e}_n(\cdot)$ is an error term that is weighted by $\mathbf{W}_n$, and would consist, in this case of the camera and IMU error. The only way to minimise this error is through observations which are also presumed to have an associated noise term.

$$\mathbf{z}_n = g_n(\mathbf{x}) + \mathbf{v}_n, \tag{2.33}$$

where z_n is the observation, $g_n(\mathbf{x})$ the observation model, and v_n $\mathcal{N}(0, Q_n)$ and are independent. Hence as v_n is mean 0, the error can be simply formulated by:

$$\mathbf{e}_n(\mathbf{x}) = z_n - g_n(\mathbf{x}) \tag{2.34}$$

$$error = observation - predicted\ observation \tag{2.35}$$

Since observation noise between the IMU and camera v_n are independent the covariance matrix is diagonal. Hence $W_n = E[\mathbf{e}_n \mathbf{e}_n^T]^{-1} = Q^{-1}$. The aim of least squares is to find the state vector $\mathbf{x}$ that minimises the error function, hence the optimisation problem is posed as:

$$\mathbf{x} = argmin(J(\mathbf{x})) \tag{2.36}$$

If the error term $\mathbf{e}_n(\mathbf{x})$ was linear w.r.t $\mathbf{x}$ the solution would be as simple as setting the $\frac{\delta J(\mathbf{x})^T}{\delta \mathbf{x}}$ to zero and solving the respective simultaneous equations. However this is often not

the case, and so the solution to the equation must be solved iteratively, with for example, Gauss-Newton formulation of the equations and a solver such as Google-Ceres.

SLAM Summary

The information in this section has covered the building blocks of a SLAM system in limited detail. It should provide an idea of how the system works, from the intuitive understanding of landmark correlation, the implementation of Harris corner detection, and finally solving for cyclist state from respective error functions. For further information the reader is directed towards the Okvis paper [28], and other reports of a more in-depth nature in this field.

2.3 Background Object Tracking

Object tracking has been recently advanced by means of discriminative learning methods, these methods try to distinguish between the object and the environment that it is in. The model must therefore learn between positive and negative samples in order to gain an understanding of the relevant object environment.

2.3.1 Kernelised Correlation Filter

In the recent paper by Henriques et al. [34] the tools are developed to supply thousands of negative samples without iterating over them explicitly. The paper makes use of circulant matrices to provide negative samples. These are obtained by translating the data (a positive example) using a *cyclic shift operator*, which in the 1D signal case is:

$$P = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix} \tag{2.37}$$

Therefore the product of $P\mathbf{x} = [x_n, x_1, x_2, ..., x_{n-1}]$ and any shift can be modelled using $P^u\mathbf{x}$. Because of its cyclic nature, the signal $\mathbf{x}$ repeats when $u = n$ and so the full set of shifted signals is obtained with $P^u\mathbf{x}|u = 0, ..., n-1$. An example of this is shown in the 1D case by Figure 2.18.

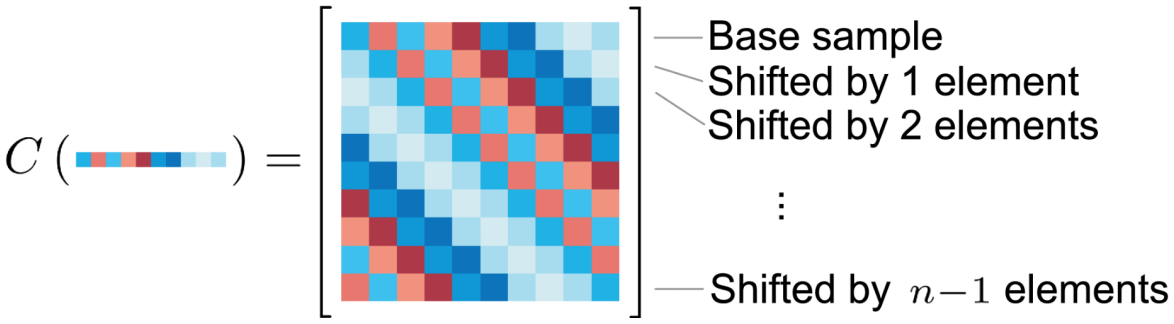

Figure 2.18: Illustration of circulant matrix.[34]

Therefore to compute the regression with shifted samples, the data matrix X takes the form:

$$X = C(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 & \dots & x_{n-1} & x_n \\ x_n & x_1 & \dots & x_{n-2} & x_{n-1} \\ x_{n-1} & x_n & \dots & x_{n-3} & x_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_2 & x_3 & \dots & x_n & x_1 \end{bmatrix} \tag{2.38}$$

From the training data X the model can be trained to classify between positive and negative examples. Noting that all circulant matrices can be diagonalised by the Discrete Fourier Transform (DFT) X can also be written as:

$$F(\mathbf{z}) = \sqrt{n}F\mathbf{z} \tag{2.39}$$

$$\hat{\mathbf{x}} = F(\mathbf{x}) \tag{2.40}$$

$$X = Fdiag(\hat{\mathbf{x}})F^H \tag{2.41}$$

This is possible because the DFT is a linear operation.

Kernels

Now that the KCF has a number of examples to learn from it turns into a simple regression problem. From any machine learning textbook it is known that by using a kernel, a linear model can implicitly learn a high-dimensional feature space. The kernel function used to do this (e.g. Gaussian), Equation 2.42 creates an $n \times n$ kernel matrix $K_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$

$$k(\mathbf{x}, \mathbf{x}') = exp(\frac{1}{\sigma^2}||\mathbf{x} - \mathbf{x}'||^2) \tag{2.42}$$

and therefore the function becomes:

$$f(\mathbf{z}) = \mathbf{w}^T\mathbf{z} = \sum_{i=1}^{n} \alpha_i k(\mathbf{z}, \mathbf{x}_i) \tag{2.43}$$

with the solution of Kernelised Ridge Regression given by:

$$\alpha = (K + \lambda I)^{-1}\mathbf{y} \tag{2.44}$$

where K is the kernel matrix and α is the vector of coefficients α_i. Given a kernel function that combines data through a commutative operation, it can be proven that given circulant data the kernel matrix K is also circulant and therefore the DFT trick can be used. This leads to a diagonalisation of Equation 2.44 to Equation 2.45.

$$\hat{\alpha} = \frac{\hat{\mathbf{y}}}{\hat{\mathbf{k}}^{\mathbf{xx}} + \lambda} \tag{2.45}$$

where $\hat{\mathbf{k}}^{\mathbf{xx}}$ is the first row of the kernel matrix $K = C(\hat{\mathbf{k}}^{\mathbf{xx}})$, and a $\hat{}$ denotes the DFT of a vector. This means that only an $n \times 1$ kernel matrix needs to be computed, due to the circulant kernel property and DFT. This makes the method far faster than conventional kernel methods that require an $n \times n$ sized kernel matrix.

Detection

The basic premise is that the α term in Equation 2.45 is now initialised to a learnt set of coefficients that correctly describe the object. Therefore, in order to detect the object of interest the regression function $f(\mathbf{z})$ must be evaluated at a number of candidate patches. These patches can themselves also be described by circulant shifts. Defining the kernel matrix K^z between cyclic shifts of base samples x and z, with elements $k(P^{i-1}z, P^{j-1}x)$. The cyclic property, as described in Section 2.3.1 allows only the first row of the kernel matrix to be relevant, and hence :

$$K^z = C(\mathbf{k^{xz}}) \tag{2.46}$$

where k^{xz} is the kernel correlation of x and z. Finally the regression equation can be computed for all candidate patches by:

$$f(z) = (K^z)^T \alpha \tag{2.47}$$

This can be further sped up by the correct choice of kernel function and a clever kernel trick to modify the computational cost of kernel computation to $O(nlogn)$. The details of which are beyond the scope of this report but can implementation can be found in the relevant paper [34].

Chapter 3

Contribution

3.1 Software Solution

The system consists of a number of modules, each responsible for a specific aspect, see Figure 3.1 for more details. These will now be summarised below, then a more in-depth review will occur in upcoming sections of the report. The Realsense camera feeds data into three modules, Okvis for SLAM and a global representation of the environment, the cyclist model for path prediction and finally Mask-RCNN to segment the image and detect objects in the camera frame. Okvis also provides information to the cyclist model which needs to understand the bike's location in the world frame to predict future positions.

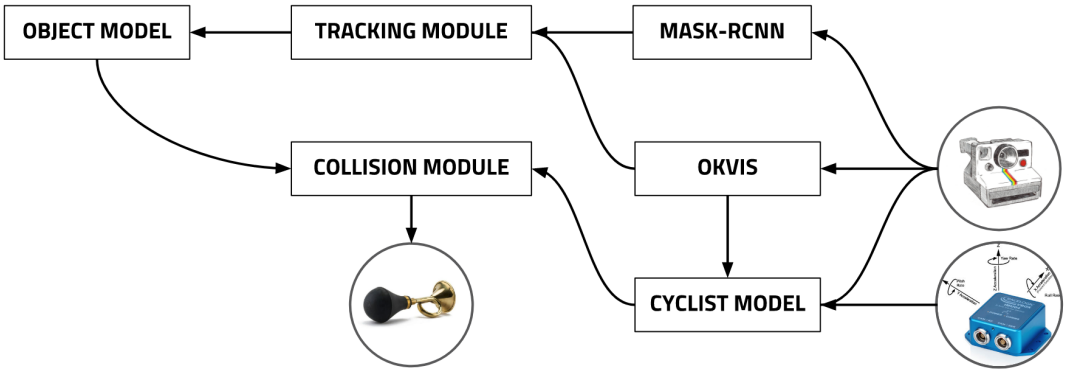

Figure 3.1: System Design

Tracking Module The tracking module takes the segmented image masks from Mask-RCNN, the passed-on depth image, and Okvis pose output to determine the location of a detected object. The distance of an object is inferred from the depth camera, and masks, which can then be converted into the world frame using the Okvis pose estimate. From these outputs, and a passed image, various features from objects can be matched from frame-to-frame. These features include global position, BRISK keypoints, and the KCF model of the object. Once the objects from the previous frame are correctly identified in the new frame, the object model can predict future movement.

Collision Module The collision module takes predicted positions from the object models and cyclist models. It then determines whether there is a large likelihood of intersection between the cyclist and objects in the future. The idea of this is to then notify both the object (possibly a person or car), and the cyclist that they are about to collide. Hopefully both parties, once aware of the unfolding situation can take preventive measures to avoid each other.

Limitations Sadly the system currently cannot run in real-time due to Mask-RCNN, the current state of the art segmentation network taking c. 0.5s to predict an image. There is a future possibility of incorporating a simpler, and faster network such as YOLO which foregoes the advantage of segmentation (useful for depth approximation) but would run in real-time.

3.2 Depth Measurement Outlier Removal

In order to get the depth measurement of an object, the mask output from Mask R-CNN is overlayed onto the depth map and a median depth is taken. A number of points are not detected and therefore large quantities of the depth map is 0. Therefore, the median depth is selected only from coordinates at which $depth > 0$. Figure 3.2 shows some examples of a depth camera image taken in the Imperial Library.

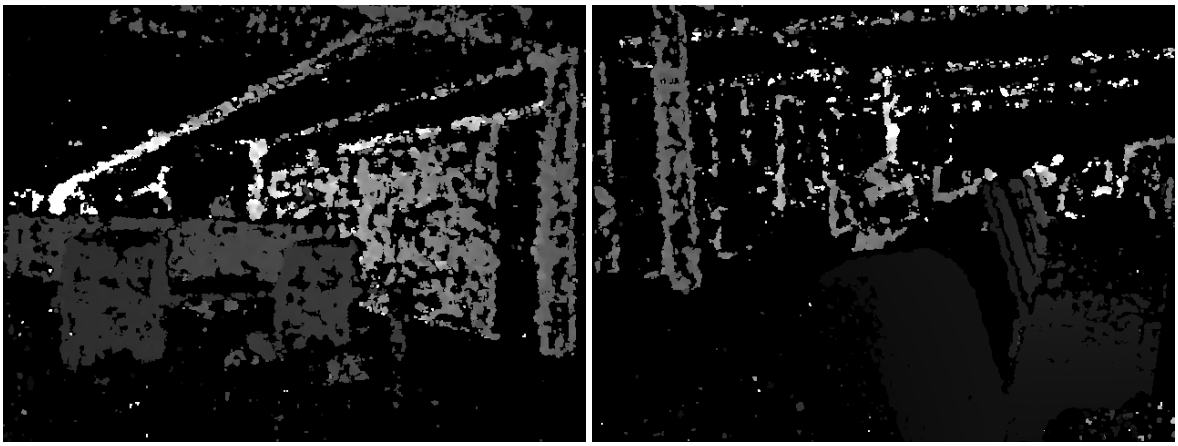

Figure 3.2: Depth Camera Output

As can be seen from Figure 3.2 the data is sparse and unreliable for objects at a distance. This is less so as the depth is scaled [0-255] for a picture and so some measurements seem less reliable than they are but it is still a challenge. Often when the depth measurements are incorrect an infeasibly large value is returned $> 50m$. This is a real problem for tracking objects, not necessarily after a period of successful tracking as the value can be discarded as an outlier but on initialisation or after only a couple of data points. Not only is the depth incorrect but that problem is propagated into all directions when transformed to the world frame. An example of this phenomenon is displayed in Figure 3.3 where a bottle is tracked from frame to frame, which also demonstrates the ability of the tracking method to be non-cycling domain specific.

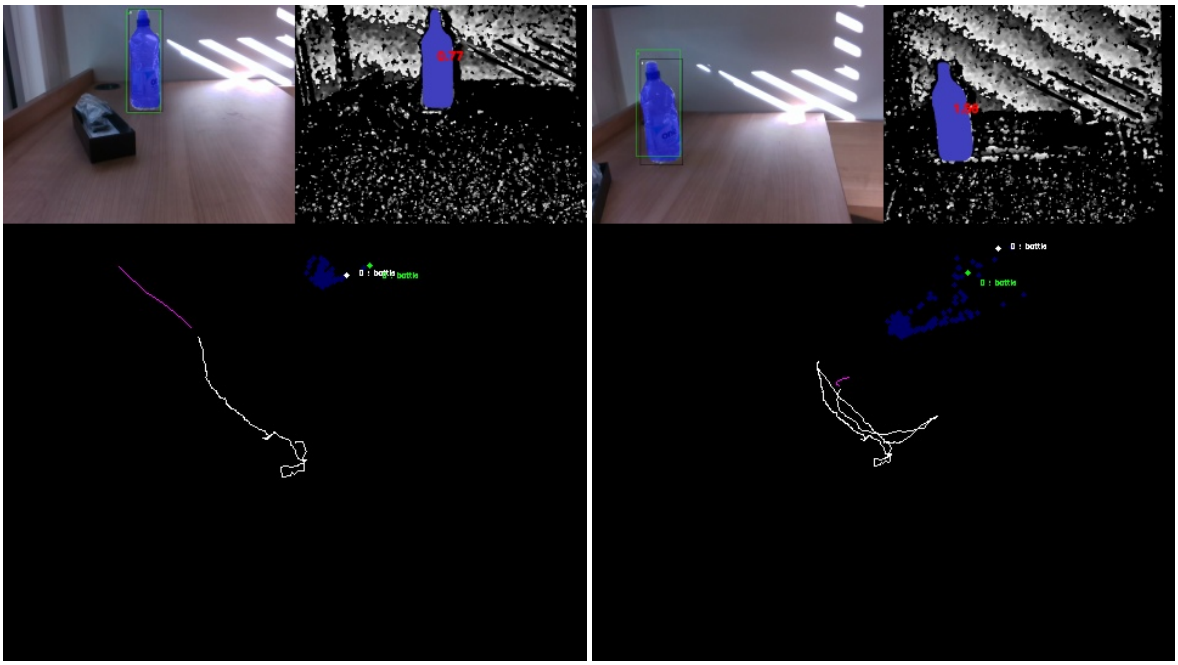

Figure 3.3: Left: Bottle with working depth, Right: Incorrect Depth measurement. In both cases the white dot is the bottle's location, the green dot is the Kalman filter prediction, blue dots are previous locations (the bottle did not move), and the white line is the camera location.

This can occur in a number of instances and far away objects are most prone as there is little area covered on the depth image to measure the distance from. However, since the object is defined by a bounding box, prior knowledge of the classes can be used to detect such an outlier. Intuitively if two objects are the same size in an image and the first is closer than the second, then the latter object must be larger than the first. Hence using the camera to world model, if the depth is incorrectly large the height of the object in the world frame will be far larger than is known to be possible i.e. a 10m tall human.

Given the knowledge that human sizes roughly vary from a 4ft child to a 7ft adult, poor depth predictions convert people to outside of that range. If that is the case then it must be modified to an approximate value, say the depth of an averagely sized human $1.5m$. This logic can be extended to other objects such as cars and bikes. Therefore, assuming a bounding box of $[x_{tl}, y_{tl}, x_{br}, y_{br}]$, where tl denotes top-left corner and br is the bottom right, the matrix P can be formed.

$$\mathbf{P} = \begin{bmatrix} x_{tl} & x_{tl} \\ y_{tl} & y_{br} \end{bmatrix} \tag{3.1}$$

Completing the initial steps of Cam2World (Section 2.2.2) until Equation 2.20, and adding the 1 required for the proceeding transform, the process yields:

$$\mathbf{K} = \begin{bmatrix} k_{(1,1)}\,d & k_{(1,2)}\,d \\ k_{(2,1)}\,d & k_{(2,2)}\,d \\ d & d \\ 1 & 1 \end{bmatrix} \tag{3.2}$$

Where $[k_{(1,1)}, k_{(2,1)}]$ are from $[x_{tl}, y_{tl}]$, $[k_{(1,2)}, k_{(2,2)}]$ from $[x_{tl}, y_{br}]$, and $d = depth(\mathbf{x}')$ from

the depth image. For the next steps, the transformation matrix T_{WC} is used to calculate the world position.

$$\mathbf{T_{WC}} = \begin{bmatrix} \dots r_1 \dots \\ \dots r_2 \dots \\ \dots r_3 \dots \\ \dots r_4 \dots \end{bmatrix} \tag{3.3}$$

where r_i denotes row i of T_{WC}, and $r_4 = [0_{3\times1}, 1]$. Therefore replicating a simple matrix multiplication in Equation 3.4, the row r_3 is the only row from $\mathbf{T}_{WC}$ that is needed for the object height.

$$\mathbf{WorldPoints} = \begin{bmatrix} x_{w,tl} & x_{w,bl} \\ y_{w,tl} & y_{w,bl} \\ z_{w,tl} & z_{w,bl} \\ - & - \end{bmatrix} = \mathbf{T}_{WC}\mathbf{K} = \begin{bmatrix} \dots r_1 \dots \\ \dots r_2 \dots \\ \dots r_3 \dots \\ \dots r_4 \dots \end{bmatrix} \begin{bmatrix} k_{(1,1)}d & k_{(1,2)}d \\ k_{(2,1)}d & k_{(2,2)}d \\ d & d \\ 1 & 1 \end{bmatrix} \tag{3.4}$$

Hence the height of the object $z_{w,tl} - z_{w,bl}$ is obtained from:

$$z_{w,tl} - z_{w,bl} = r_3\mathbf{K}[:,1] - r_3\mathbf{K}[:,2] = r_3(\mathbf{K}[:,1] - \mathbf{K}[:,2]) \tag{3.5}$$

where $\mathbf{K}[:,1]$ defines the first column of K, and defining vector A as:

$$Ad = \mathbf{K}[:,1] - \mathbf{K}[:,2] = \begin{bmatrix} k_{(1,1)}d - k_{(1,2)}d \\ k_{(2,1)}d - k_{(2,2)}d \\ 0 \\ 0 \end{bmatrix} = d \begin{bmatrix} k_{(1,1)} - k_{(1,2)} \\ k_{(2,1)} - k_{(2,2)} \\ 0 \\ 0 \end{bmatrix} \tag{3.6}$$

then an approximation of the depth, given the average object height o_{av} is:

$$depth \approx \frac{o_{av}}{r_3 A} \tag{3.7}$$

This depth approximation formula can both be used to detect anomalies and to approximate the true depth, discarding the incorrect camera depth measurement. This allows better modelling and tracking of object movement from frame-to-frame.

3.3 Tracking Module

For each frame Mask RCNN outputs a set of bounding boxes, and masks. These are not within any particular order and so a method must be devised to track objects from one frame to the next. This is a hard challenge due to object occlusion, variation in shadow/lighting conditions and rapid in-frame movement. An obvious method is intersection over union (IOU) of the bounding boxes on a frame-by-frame basis. This is good in theory, however, it lacks the ability to compensate for rapid movement (of the object and the camera angle), occlusion and object overlapping. A set of methods have therefore been developed to overcome this problem, which combined provide a robust tracking system.

The method must both track objects in the frame, determine which objects are no-longer in-frame and find any new objects that enter the frame. Information about each object must be stored as to determine the velocity and direction of travel/ if a collision is likely between the cyclist and the object in question. Figure 3.4 displays an example of a working system.

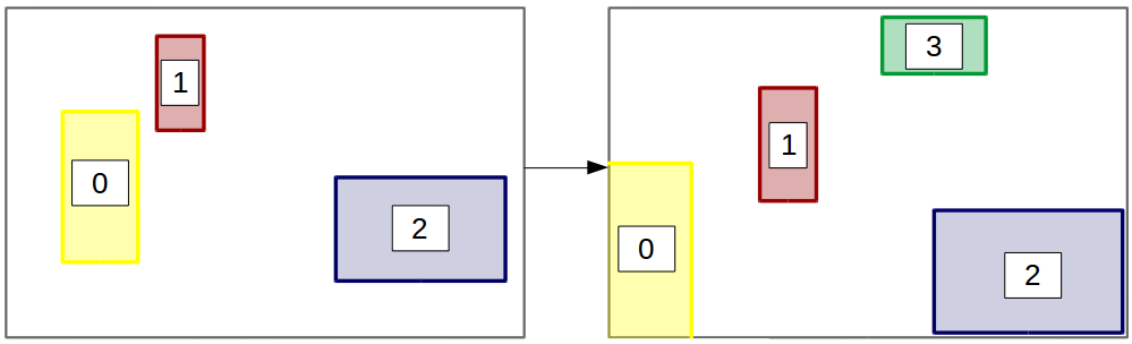

Figure 3.4: RoI Old to New Frame Example.

3.3.1 Kernelised Correlation Filter

A Kernelised correlation filter - one of the state-of-the-art methods used today for tracking objects from one frame to another is described in Section 2.3.1. A minor modification is required for use in a multi-object scenario which is described below:

Modifications: Since the bbox prediction is so fast for the KCF tracker; it can be used in conjunction with the other tracking methods mentioned in Sections 3.3.2, 3.3.3, and 3.3.4. The kcf tracking algorithm outputs a bounding box which is the predicted coordinates of the object in the next frame. This can be compared to Mask-RCNNs output through a simple matching process. A box in the new frame predicted by Mask RCNN $[x_{mask}$,y_{mask},$width_{mask}$,$height_{mask}]$ can be scored by sampling the pdf from a multivariate Gaussian, in 4D, with mean $[x_{kcf}, y_{kcf}, width_{kcf}, height_{kcf}]$ and covariance matrix. The covariance matrix is an arbitrary choice but determines how far the system looks for matches from the predicted kcf location. The covariance matrix in the system is set to $20 \times I_{4\times4}$ which translates to a $\sigma^2 = 20$ pixels for $x, y, width, height$ respectively, and assumes independence between predicted variables. This probability score can then be used in Section 3.3.6 to be combined with the other tracking modules.

3.3.2 BRISK - Selector

BRISK as described earlier in Section-2.2.7 is a binary descriptor that can match keypoints across two photographs. This can be used also in the tracking section to match keypoints across RoIs, although not thoroughly accurate, it offers a method of matching RoIs regardless of location. This should help re-detect objects after periods of occlusion, invariant of distance, which other methods are unable to do. The process is simple to match the RoIs. First keypoints and descriptors are calculated for each RoI, these are then compared using the fast binary matching process of BRISK. The number of matches in each RoI determines how well they match.

3.3.3 Class Matching

An obvious additional requirement is the matching of classes between frames, as a person's RoI should not match to the RoI of a car in the next frame. Hence for each RoI, the class is stored for access by the next frame. This can both reduce expensive computation time of other metrics but provides an easy RoI comparison tool. It can be noted although that Mask

RCNN is not perfect and often miss-classifies objects, it is reliable enough to be assumed as correct.

3.3.4 Kalman Filter RoI Matching

A Kalman filter, developed by Rudolf E. Kalman has two main parts, a predictive step and an update. The predictive step tries to find a-priori estimates of the state for the next time-step by projecting forward the current state and error covariance estimates. This estimate is then updated using measurements of the object location. The iterative process continues improving the Kalman filter estimates at each stage.

This was first tested with just a camera frame state i.e. using the [x, y, width, height] positions and distances in pixels of each RoI on the input image. It was quickly discovered that the model was not sufficiently robust as it did not take into account the bike movement. Hence the filter was unable to track objects efficiently. However, once the Okvis framework was added, the filter was converted into tracking world frame coordinates which is far more successful.

Kalman Predictive Step

Assuming a linear system with Gaussian process, the state can be modelled as:

$$\hat{X}_k = F_k \hat{X}_{k-1} + w_{k-1} \tag{3.8}$$

where k is the time step, $\hat{X}_k \in \mathbb{R}^n$ is the state vector, F_k is the $n \times n$ state transition matrix from state k to $k+1$ and w_k is $\mathcal{N}(0, Q)$. The state vector is described as follows:

$$\hat{X}_k = [Cx, Cy, Cz, w, h, Cx_{vel}, Cy_{vel}, Cz_{vel}, w_{vel}, h_{vel}] \tag{3.9}$$

where (Cx, Cy) is the centre point of the bounding box of the object,Cz is the median distance of the masked RoI, w the width and h the height. Cx_{vel} describes the centre point velocity in the x direction, and $Cy_{vel}, Cz_{vel}, w_{vel}, h_{vel}$ the y/z direction, w the width and h the height. $[Cx, Cy, w, h]$ describe the measurable variables of the system, and there are no controllable parameters in this system. From basic equations of motion, a general measurable parameter γ, and corresponding velocity γ_{vel} follows the update rule:

$$\begin{aligned} \gamma_{k+1} &= \gamma_k + \gamma_{vel} \times \Delta t \\ \gamma_{vel,k+1} &= \quad \gamma_{vel,k} \end{aligned} \tag{3.10}$$

where Δt is the time step between each frame. Therefore, the update matrix for the filter $F_k \in R^{10\times10}$ is as follows:

$$F_k = \begin{bmatrix} I_5 & \Delta t \times I_5 \\ 0_{5\times5} & I_5 \end{bmatrix} \tag{3.11}$$

The final term $w_k = \mathcal{N}(0, Q)$ describes the process noise in the system and is normally distributed with error covariance matrix Q. This is updated by the kalman filter at each step. Since the state describes 3D coordinates it is useful to determine how far the average walking human would walk (in m) from one frame to the next, given an average walk of 1.5m/s. These parameters $dist_h$ for a human, $dist_b$ for other cyclists and $dist_v$ for a motor vehicle are listed below:

$$\begin{aligned} dist_h &= 1.5m/s \\ dist_b &= 10m/s \\ dist_v &= 15m/s \end{aligned} \tag{3.12}$$

The variance in the process noise would also be expected to increase dependent on the type of object concerned and thus they are initialised differently for each respective object type. Also it is beneficial to note that the width and height are in pixel space and so can change more rapidly, and so is scaled accordingly. The variance in process noise can't possibly be more than the movement of each object per frame so this is a good place to initialise the kalman filter.

$$\begin{aligned} v_h &= [1.5, 1.5, 1.5, 7, 7, 0.15, 0.15, 0.15, 0.7, 0.7] \\ v_b &= [10, 10, 10, 7, 7, 1, 1, 1, 0.7, 0.7] \\ v_v &= [15, 15, 15, 7, 7, 1.5, 1.5, 1.5, 0.7, 0.7] \\ Q_k &= diag(v) \end{aligned} \tag{3.13}$$

where $Q_k \in \mathbb{R}^{10\times10}$ and is the diagonalised matrix of v.

Kalman Update Step

The measurement step can also predict the current state. With $Z_k \in \mathbb{R}^{10}$ as the system measurement vector:

$$Z_k = H_k X_k + v_k \tag{3.14}$$

where H_k is an 10×5 measurement matrix relating X_k to the measurement Z_k, and v_k is $\mathcal{N}(0, R)$. Since only the Cx, Cy, Cz, w, h is measurable H_k takes the form:

$$H_k = \begin{bmatrix} I_5 & 0_{5\times5} \end{bmatrix} \tag{3.15}$$

and the measurement noise $v_k = \mathcal{N}(0, R_k)$ is set with covariance matrix. As discussed earlier the depth measurement is most prone to error, and the width/height are in pixel space. The Kalman measurement noise covariance therefore starts as:

$$\begin{aligned} r &= [0.2, 0.2, 0.2, 20, 20] \\ R_k &= diag(v) \end{aligned} \tag{3.16}$$

where $R_k \in \mathbb{R}^{5\times5}$ and is the diagonalised matrix of r. Note also that the measurement error is in the sensors and therefore is not dependent on the type of object and respective presumed velocities.

Kalman Equations

The prior estimates for the next time step by the Kalman predictions step (no control vector):

$$\begin{cases} \hat{X}_k(-) = F_k \hat{X}_{k-1}(+) \\ P_k(-) = F_k P_{k-1}(+) F_k^T + Q_k \end{cases} \tag{3.17}$$

where $\hat{X}_k(-)$ is a priori state estimate and $P_k(-)$ is the priori estimate error at step k. It is essentially the previous state $\times$ the transition matrix to the new state ($\hat{X}_k(-)$). Then

a similar approach for the error covariance between the previous and current state. The correction step is then:

$$\begin{cases} \tilde{y}_k = Z_k - H_k\hat{X}_k(-) \\ S_k = R_k + H_k P_k(-) H_k^T \\ K_k = P_k(-) H_k^T S_k^{-1} \\ \hat{X}_k(+) = \hat{X}_k(-) + K_k\tilde{y}_k \\ P_k(+) = (I - K_k H_k) P_k(-) (I - K_k H_k)^T + K_k R_k K_k^T \\ \tilde{y}_{k|k} = Z_k - H_k\hat{X}_k(+) \end{cases} \tag{3.18}$$

where K_k is the $n \times m$ Kalman gain matrix, $\hat{X}_k(+)$ is the posterior state estimate according to the actual measurement Z_k and the predicted measurement $H_k\hat{X}_k(-)$ and $P_k(+)$ is the posterior state estimate error covariance.

Kalman Matching RoIs

The system needs a metric to match a box from a prior frame to the current frame. Using the Kalman filter, a prediction of where the RoI is at time k is easily obtained. But this does not give a likelihood measure of the boxes in the new frame. However, in the kalman update step $\tilde{y}_k$, the innovation residual, and S_k, the innovation covariance matrix are calculated. These can be used to determine the marginal likelihood of a newer RoI's $[Cx, Cy, w, h]$ given the kalman $P_k(-)$ and $\hat{X}_k(-)$. Skipping the formulation, the log-marginal likelihood is:

$$l = log\ p(\mathbf{z}) \tag{3.19}$$

$$\begin{cases} l^{(-1)} = 0 \\ l^{(k)} = l^{(k-1)} - \frac{1}{2}(\tilde{y}_k^T S_k^{-1} \tilde{y}_k + \log|S_k| + d_y \log 2\pi) \end{cases} \tag{3.20}$$

where d_y is the dimension of the measurement matrix. The part dependent on the new RoI is $\tilde{y}_k^T S_k^{-1} \tilde{y}_k$ as S_k, from Equation 3.18 does not involve $\tilde{y}_k$. Therefore the matching step can be performed by simply matching each ROI_{old} in the old frame to the ROI_{new} in the new frame that has the lowest scalar value from:

$$M = \tilde{y}_k^T S_k^{-1} \tilde{y}_k \tag{3.21}$$

with a threshold value to ensure no incorrect matches. The matching procedure must be the same for all methods and matches the maximum value. Therefore the *Match Metric* is as follows:

$$\textit{Match Metric} = \begin{cases} 0 & \textit{if } K_{threshold} - M \leq 0 \\ K_{threshold} - M & \textit{otherwise} \end{cases} \tag{3.22}$$

3.3.5 RoI Lives

Each region of interest may be tracked from one frame to the next. However, if that object is occluded then it may not be spotted in the next frame. This, unless accounted for, could lead to information loss regarding that object's state. Therefore a life system is in place whereby an object has a number of lives that are used when the object is not spotted in the next frame. These are decremented until 0 when the object is forgotten and no longer tracked.

3.3.6 Combining Matching Methods

For each ROI_{old} and each ROI_{new} the match metrics for each method (KCF,BRISK,Kalman) is calculated. This forms a matrix $matches \in \mathbb{R}^{new \times old \times methods}$ where old, new are the number of RoIs in the old and new frames, and $methods$ is the number of matching methods implemented (in this case two) but is designed to be easily extendable to more. The algorithm for matching RoIs is presented in the appendix, Algorithm 7.

After the matching process is complete, there are a number of extra steps that must take place before continuing to the next frame. For example, any old RoIs that are not detected in the new frame must have a life decremented and be added to the list of new frame RoIs with the relevant location and information, Algorithm 4.

Algorithm 4: Algorithm to append old RoIs that aren't detected in new frame

add_old_rois (old_matched, old_frame, new_frame)

Input :

$old\_matched$ // A list of matched old RoIs, $old\_matched[i]$ matches to $new\_matched[i]$

old_frame // RoIs as structures from the old frame

new_frame // RoIs as structures from the new frame

Output:

new_frame // An updated list of new RoI objects

```
// Update newROIs to include undetected RoIs from old frame.
foreach old ∉ old_matched do                    /* For every old roi not found in new frame */
    older = old_frame[old]                                          /* Get olderROI object */
    if older.lives > 1 then                          /* If the RoI has a life to spare */
        older.lives -= 1                                            /* Decrement a life */
        update_location(older, T_CW)          /* Update world location from Kalman prediction */
        add_information(new_frame, older)     /* Add older RoI information to new frame */
              /* (masks,lives,id,class,colour,briskKeypoints,briskDescriptors,KCFtracker) */
end
```

Then the matched RoIs pairs from the old and new frame must exchange relevant information in preparation for the next frame, Algorithm 5.

Algorithm 5: Algorithm to update new ROIs with matched old ROIs

update_matched_rois ($old\_matched, new\_matched, old\_frame, new\_frame$)

Input :

$old\_matched$ // A list of matched oldROIs, $old\_matched[i]$ matches to $new\_matched[i]$

$new\_matched$ // A list of matched newROIs, $new\_matched[i]$ matches to $old\_matched[i]$

old_frame // RoIs as structures from the old frame

new_frame // RoIs as structures from the new frame

Output:

new_frame // An updated list of newROI objects

```
// Add older RoI details to matched new ROoI
foreach old, new ∈ old_matched, new_matched do /* For every matched old/new RoI pair */
    older = oldROIs[old]                        /* Get olderROI object */
    newer = newROIs[new]                        /* Get newerROI object */
    newState = newer.worldPoint                 /* Get new location of RoI */
    older.kalman.correct(newState)              /* Update Kalman Filter with new location */
    if older.lives < lifeThreshold then         /* If lives < threshold */
        older.lives += 1
    update_object(newer, older)                 /* Update new RoI with older RoI information */
                                                /* (Kalman Filter,lives,colour,id,model) */
end
```

Finally any new RoIs that were not in the old frame need to be assigned an individual ID to be recognised over the next frames, Algorithm 6.

Algorithm 6: Algorithm to create new IDs for object in new frame

update_matched_rois ($old\_matched, new\_matched, old\_frame, new\_frame$)

Input :

$old\_matched$ // A list of matched oldROIs, $old\_matched[i]$ matches to $new\_matched[i]$

$new\_matched$ // A list of matched newROIs, $new\_matched[i]$ matches to $old\_matched[i]$

new_frame // RoIs as structures from the new frame

Output:

new_frame // An updated list of newROI objects

```
// Note that if there are no objects in old frame new_frame.id = zeros
maxId = new_frame.id.max() + 1 ;                    /* Get current maximum ID + 1 */
foreach new ∉ new_matched do                        /* For every not-matched new RoI */
    newer = new_frame[new] ;                        /* Get newerROI object */
    newer.id = maxId ;                              /* Assign new ID */
    maxId += 1 ;                                    /* Increment the maximum ID */
end
```

3.4 Movement Prediction

Each object and the bike must be modelled from frame-to-frame and the subsequent path predicted for collision detection. The further in the future that can be accurately modelled, the better the warning system. The following section describes both the more complex bicycle model and the simple path prediction for pedestrians.

3.4.1 Datasets

The great advantage of the system is that for movement prediction it already has the core components required to create unlimited training data. All that is required is to cycle around and record the images and pose with Okvis. This can then be passed into the model for training. One caveat to this is that due to the poor depth camera readings outside, Okvis sometimes struggles to maintain the correct position and can drastically miss-predict the speed of the bicycle. This means that a number of runs are required to get a working dataset. Hence three main datasets were collected, a Blackfriars dataset which contains three runs of the route on separate days, an Imperial dataset (around Queen's Tower), and a dataset from walking around the Tate Modern.

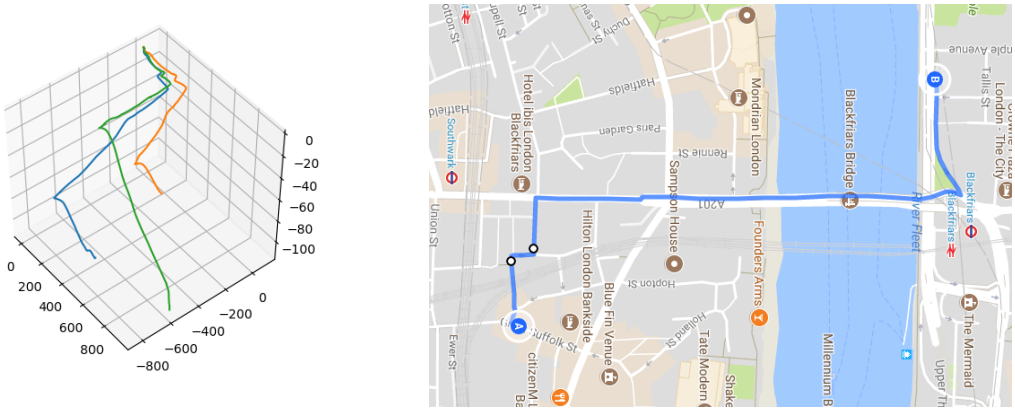

Figure 3.5: Okvis Paths: Left, Google Map: Right.

Although the Okvis paths look vastly different, the model is only interested in short segments and hence the variation is less than perceived in Figure 3.5. The main problem with models is the ability to generalise, hence training on a limited, unbalanced dataset is tricky. So for the prediction it is assumed that someone is able to record their daily commute and train the model on that route. So the validation and test sets consist of a same route on a different

day with the obvious varied lighting, traffic conditions, and traffic lights that come with the average daily commute variation.

3.4.2 Objects Prediction

The objects are modelled by the Kalman filter from frame to frame and this provides a good estimate of both the direction and predicted position. Hence at a future position, say in 10 frames, would simply be obtained from multiplying by a modified state-transition model.

$$F_{k+n\_seconds} = \begin{bmatrix} I_5 & n\_seconds \times I_5 \\ 0_{5\times5} & I_5 \end{bmatrix} \tag{3.23}$$

where $n\_seconds$ is the number of seconds in the future to predict the position at.

$$\hat{X}_{k+n\_seconds}(-) = F_{k+n\_seconds}\hat{X}_{k-1}(+) \tag{3.24}$$

The uncertainty of that position also increases with time, and can be taken directly from the error covariance prediction matrix. This however is updated iteratively and so it is easy enough just to repeat Equation 3.25 $\left(\frac{n\_seconds}{\Delta t}\right)$ times.

$$P_{k+\Delta t}(-) = F_k P_{k-\Delta t}(+)F_k^T + Q_k \tag{3.25}$$

However, for speed of computation an approximation can be used:

$$P_{k+n\_seconds}(-) \approx F_{k+n\_seconds}P_{k-\Delta t}(+)F_{k+n\_seconds}^T + \frac{Q_k \times n\_seconds}{\Delta t} \tag{3.26}$$

$$radius = \sqrt{P_{k+n\_seconds}(-)} \tag{3.27}$$

This method then produces a predicted position and the uncertainty in that position which can be used to create spheres of possible object positions in future frames. These can then be checked for intersection with the bike model predictions.

3.4.3 Bike Model

The bike is deemed to be the most important model and therefore predictions should take into account higher order features, for example the current state of the bicycle in the frame and perhaps external factors such as the surrounding environment. The problem is therefore split into two branches, one for sequence prediction of Okvis pose outputs and the other of predictions using image inputs.

Okvis Sequence Inputs: Since the model is required to learn a sequence a recurrent neural network is the obvious choice. This mimics memory from one-frame to the next, providing n previous frames to predict the $(n+1)th$ frame. The proposed architecture for this model, by no means fine-tuned but merely to provide a proof-of-concept is as follows:

Pose RNN Model: The main problem is to generalise the model to any world point. For example, the bike could be at any location or offset from the world origin and thus a huge amount of data would be needed for learning if the world location was an input. Instead, the model simply learns the difference in camera frame coordinates from the $0th$ frame in the sequence. Note however, things such as orientation can make a large difference, if the

bike is at a big angle in the world frame it will be turning quickly. This information cannot be transferred from the camera frame orientation. Hence the tilt part of the world to camera transformation matrix for the nth frame is also added as an input.

$$X_{frame_n} = \left[[I_{1\times 3} \quad 0] T_{WC_0}^{-1} T_{WC_n} \quad , \quad C_{ez} \right] \tag{3.28}$$

$$C_{ez} = (T_{WS} T_{SC})^{-1} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix}^T \tag{3.29}$$

$$y = r_{WC_{n+1}} \tag{3.30}$$

The model is formed of the layers detailed on the right of Figure 3.6. Given a limited input space $X = (len_{sequence}, 16)$ where the transformation matrix is flattened for use in the LSTM layers to a matrix of size 12. The model is trained in a regression setting with labels $\mathbf{y} = [x, y, z]$ which represent the displacement from frame last of the sequence at the subsequent frame. As it would be too granular to learn from every frame, a larger time-gap between frames can be used and hence the model sequence can consist of every kth frame from the camera. The loss function used is mean squared error and is formulated as follows:

$$mse = \sum_{i=0}^{n} \sum_{j=0}^{m} (\mathbf{y}_{i,j} - \hat{\mathbf{y}}_{i,j})^2 \tag{3.31}$$

where i is the sequence number, j is the variable prediction and therefore $m = 3$ for $[x, y, z]$. The model simple as the number of features is small so learning from the pose inputs will not require a complex model. The structure is formulated below:

Layer Number	Layer	Size
1	LSTM	32
2	LSTM	32
3	LSTM	32
4	Dense	3

Table 3.1: Pose Model Dimensions

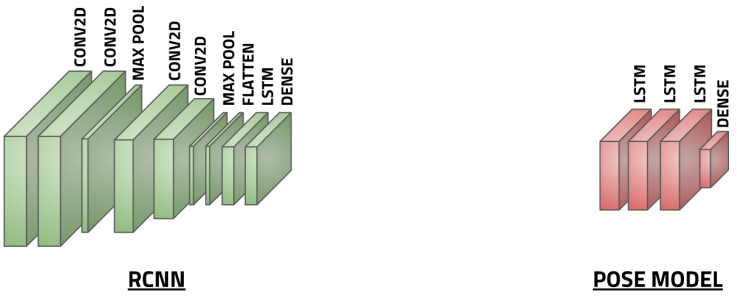

Figure 3.6: RCNN Architecture: Left, Pose Architecture: Right.

Images RCNN Model: For the image section of the model, a standard convolutional learning approach is adopted as described in Section 2.1.2. The loss function for the RCNN model is the same as Equation 3.31, as once again this is a regression problem. The images are fed into the network and the difference predicted much like before. The only variation is the data input and the flattening layer before the LSTM. A convolutional layer outputs a number of 3D tensors (w, h, d) which is not compatible with the LSTM or dense matrices. Hence a flattening layer converts the tensors into a shape $(sequence\_length, w \times h \times d)$ for input into the final network layers. Additionally some extra pre-processing is required, since images are scaled from 0 to 255 it is hard to learn. Hence the images are simply scaled to between $(-1, 1)$ by $(image - 127.5)/127.5$ before being passed into the model. Also note that images must be resized to around $200px, 200px$ in order for the model to fit into GPU memory. This is particularly prevalent when classifying a sequence.

Layer Number	**Layer**	**Size**
1	Convolutional	32
2	Convolutional	32
3	Max Pooling	N/A
4	Dropout (.25)	N/A
5	Convolutional	16
6	Convolutional	16
7	Max Pooling	N/A
8	Dropout (.25)	N/A
9	Flatten	N/A
10	LSTM	64
11	Dense	32
12	Dense	3

Table 3.2: RCNN Dimensions

Optimising Learning Rate and Decay: Adam optimiser was used for the model, it is a common solver for deep learning problems requires a number of parameters which are all set as the default recommended from the paper [35] $\beta_1 = 0.9$,$\beta_2 = 0.999$,$\epsilon = 10^{-8}$ these are adaptive moment parameters that control the algorithm's descent into minima. However learning rate and learning rate decay are problem specific and can vastly impair the ability of a model to learn correctly. Hence a logical approach to selection of the right values is required. A grid search method was chosen and loss characteristics at epoch 200 compared to get a reasonable approximate for the correct learning rate.

Both the image and pose model appear to prefer higher learning rates around the $10^{-4}, 10^{-3}$ region. Learning rate decay has little influence to the results of the images network which has an optimal values of:

From experience, and logically, it is preferable to take an optimal value with decay for longer runs of the network. As the optimisation is performed over only 200 epochs often the lower learning rates with decay perform better in the long-run. Hence the parameter choice ranked no.2 in Table 3.3 is used for the final model. A part to note is that the loss function is mean squared error and so 0.11 loss is roughly equivalent to $0.3m$. The pose model interestingly

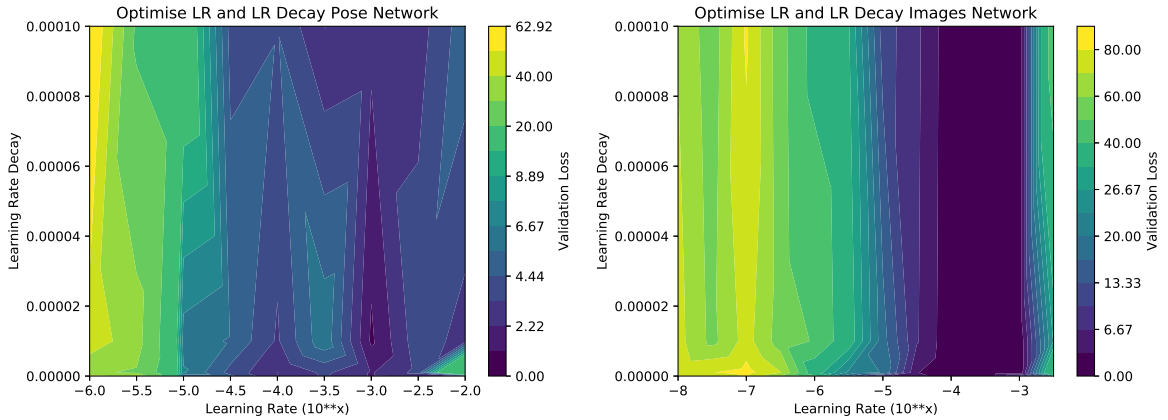

Figure 3.7: Pose Lr and Lr Decay Grid: Left, Images Network Optimisation: Right.

Rank	Learning Rate	Decay	Validation Loss
1	$10^{-3.5}$	0	0.11
2	10^{-3}	10^{-6}	0.14
3	$10^{-3.5}$	10^{-5}	0.19

Table 3.3: Image Learning Rate vs Learning Rate Decay

has a higher optimal loss than the images model, but this is likely due to the number of epochs and model complexity in using the same route for the test set (different days but same route). The pose model is simple and therefore can train much faster than the images model so it should be trained for longer for a fair comparison. The images model has more layers and thus may over-fit the route whilst the pose model may perform better on unknown routes.

Rank	Learning Rate	Decay	Validation Loss
1	10^{-3}	10^{-5}	0.91
2	10^{-3}	10^{-6}	2.11
3	10^{-4}	10^{-6}	2.20

Table 3.4: Pose Learning Rate vs Learning Rate Decay

The highest ranked parameters in Table 3.4 are taken forwards to the next stage of the process, optimising the layer dimensions.

Bayesian Optimisation Layer Dimensions:

The images and pose model are first optimised with respect to learning rate and learning rate decay with a grid search. From there each is optimised with respect to layer dimensions via Bayesian Optimisation. It is questionable as to the order that this should occur in but for simplicity learning rate and decay are optimised and then layer size.

Pose Model: As the pose model was once again optimised with a small number of epochs it has slightly higher loss than the images model. The results from optimising the LSTM layers

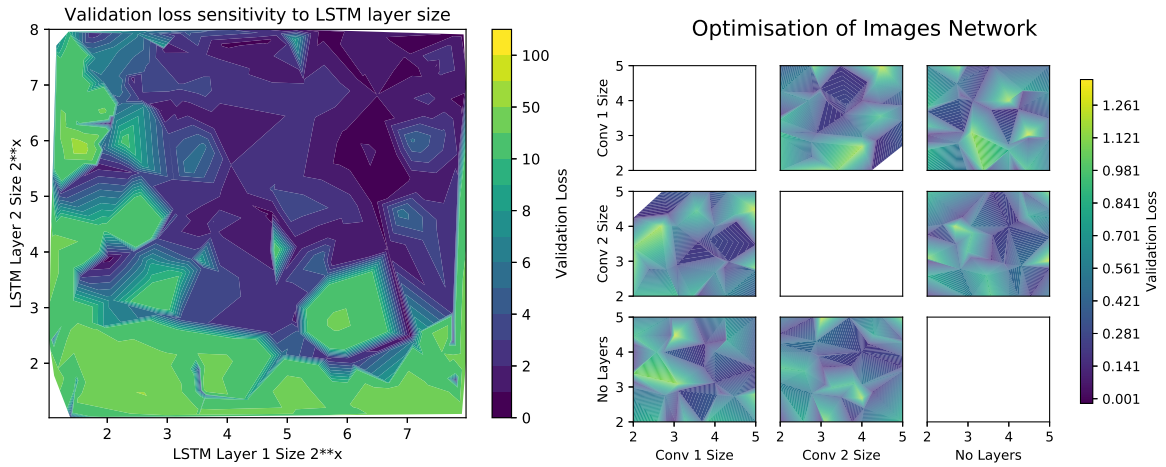

Figure 3.8: Pose Layer Optimisation: Left, Images Network Optimisation: Right.

are as follows:

Rank	LSTM 1 Size	LSTM 2 Size	Validation Loss
1	$2^{6.44} = 87$	$2^{6.25} = 77$	0.42
2	$2^{6.04} = 63$	$2^{5.7} = 53$	0.43
3	$2^{2.03} = 4$	$2^{3.9} = 15$	0.48

Table 3.5: Pose Layer Size Bayesian Optimisation

The second best and best result are narrowly different, so the second best, and simpler model from Table 3.5 was then taken forward as the final dimensions for the network, and is used in the complete model.

Images Model: The images model is optimised in 3 dimensions, two involve the size of CNN layers and one is the number of layers in the network. The network has blocks of convolutional layers (Figure 3.6), optimising the number of layers determines whether there are one or two convolutional layers in each block. Simultaneously, the size of the layers in those blocks is also optimised.

Rank	Block 1 Size	Block 2 Size	Block 1 Layers	Block 2 Layers	Validation Loss
1	$2^{4.34} = 20$	$2^{5.0} = 32$	1	1	0.1
2	$2^{3.57} = 12$	$2^{5.0} = 32$	2	2	0.11
3	$2^{5.0} = 32$	$2^{5.0} = 32$	2	1	0.13

Table 3.6: Images Model Bayesian Optimisation

Interestingly the images model prefers a simpler setup with only one convolutional layer per block. This may be due to overfitting, but since extra layers are added when combined into the total model it is beneficial to keep the simpler architecture as it is likely to perform better. Hence the final model architecture is presented in Figure 3.9

Final Optimised Models

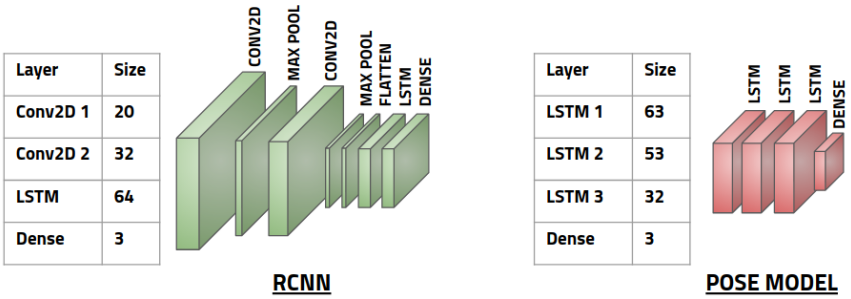

Figure 3.9: Final Images Model: Left, Pose Model: Right.

Total Model:

Both models discussed in this section can be trained separately to obtain predictions of the future pose. A naive estimation would take weighted predictions and sum for a final prediction result. The final layers of these networks however could infer details from each-other to inform the pose estimation. This can only be done in a branched network that uses both inputs effectively and concatenates the outputs for some final layers. The proposed model is graphed in Figure 3.10, it has two input tensors of the images plus pose data and three output variables which is the predicted $[x, y, z]$.

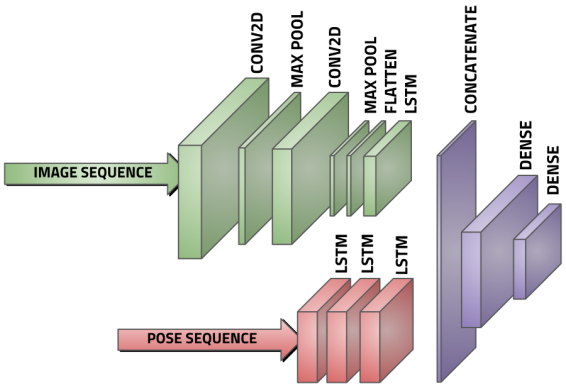

Figure 3.10: Complete Network Architecture.

Training Regime: The network is large and so a careful training regime is required to ensure convergence to a reasonable minimum. The two initial models, the pose and RCNN images are first pre-trained to predict poses. These are the complete models described in Figure 3.6. Once this training process is complete the weights are used in the larger branched

model. These layers are then frozen and the final dense layers are trained. Finally all layers are unfrozen and the whole model is trained with a lower learning rate to fine-tune the weights. For the final training, the frame gap was increased to $1s$ and hence there is a higher loss from the data. It is not excessive however with a final loss of 1.8 which equates to $1.3m$ uncertainty which is acceptable for future use. This uncertainty can be used in the next section to determine the possible locations of the bike in the future and detect if a collision is likely.

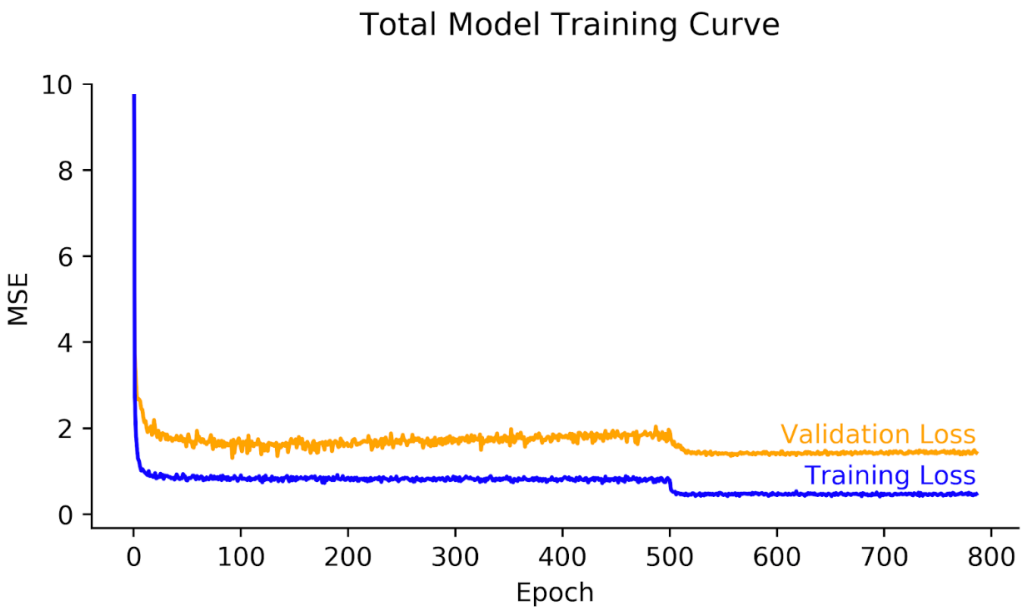

Figure 3.11: Complete Network Training.

Summary In this section a new model for cyclist movement prediction has been explored, the combination of images and pose data. This is expected to perform well with larger times between predictions. However, at smaller time gaps the model is too complex and over-fits the data which means that a linear model often performs better on the validation set. It is therefore a balance between model complexity and the required time-frame of prediction. In this case a timeframe of c.1-2s is reasonable as the Mask-RCNN model cannot detect small objects that are further away with adequate accuracy and a bike can move at around 10m/s. At this time-frame there is a narrow advantage to using the proposed architecture and so the decision would be hardware specific. This is due to computational complexity of a neural network being higher than that of a linear model.

Since models such as this are complex and require significant computational power it may be beneficial to join the image-segmentation network and prediction network. This would allow them to share some of the convolutional layers and reduce computational overhead but requires extensive modifications to Mask-RCNN. The outcome of this section is a reliable model that can be used in the next section for collision detection between moving objects and the cyclist.

3.4.4 Collision Detection Module

The collision detection module takes the predicted paths from both the cyclist and object models (Sections 3.4.3,3.4.2), and their respective uncertainties. It can then check the distance between the predicted points:

$$dist = \sqrt{((x_c - x_o)^2 + (y_c - y_o)^2} \tag{3.32}$$

where x_c is the predicted cyclist location and x_o is the predicted object location at time t. This can then be used with the uncertainty circle to see if the points could overlap.

$$\begin{cases} dist < R_{bike} + R_{obj} & \textbf{Check IOU} \\ dist > R_{bike} + R_{obj} & \textbf{Continue} \end{cases} \tag{3.33}$$

$$R_k = \sqrt{var_k} \tag{3.34}$$

One problem is that when the Kalman filter is highly uncertain it may predict a potential collision when there is little likelihood of it occurring so a solution can be to look at the intersection over union of the two circles.

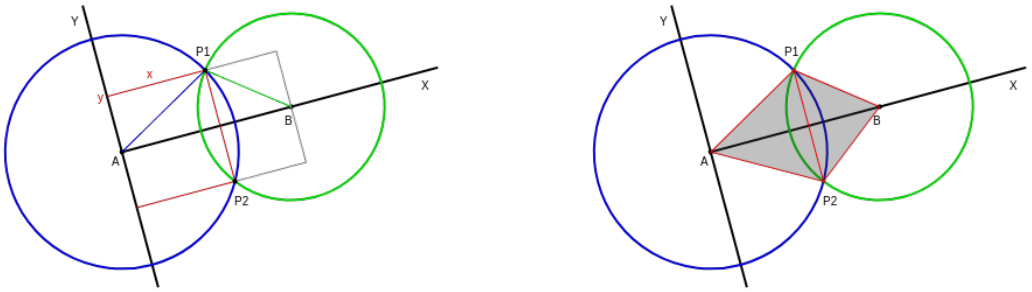

Figure 3.12: Intersecting Points on Circle [36], Right: Area of Two Intersecting Circles [37]

Points of intersection The first step to calculate where the circles overlap. This is detailed in figure 3.12. An intersection point can form a triangle in each circle between the y axis, x axis and a line of length R_o, R_c passing through the circle's origin.

$$\begin{aligned} x^2 + y^2 &= A_r^2 \\ (d^2 - x^2) + y^2 &= B_r^2 \end{aligned} \tag{3.35}$$

Solving for x and y yields:

$$\begin{aligned} x &= \frac{A_r^2 - B_r^2 + d^2}{2d} \\ y_{1,2} &= \pm\sqrt{A_r^2 - x^2} \end{aligned} \tag{3.36}$$

Transforming the values into unit vectors and solving for the final point locations gives:

$$\overrightarrow{e_1} = \frac{\overrightarrow{B} - \overrightarrow{A}}{|\overrightarrow{B} - \overrightarrow{A}|} = \frac{1}{d}\begin{bmatrix} B_x - A_x \\ B_y - A_y \end{bmatrix} \tag{3.37}$$

$$\overrightarrow{e_2} = rot(90)\overrightarrow{e_1} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \overrightarrow{e_1} \tag{3.38}$$

$$\overrightarrow{P_{1,2}} = \overrightarrow{A} + x \cdot \overrightarrow{e_1} \pm y \cdot \overrightarrow{e_2} \tag{3.39}$$

The final check is to determine if one circle is within the other and hence there are no intersection points. This is given by checking that the distance is greater than the absolute difference between radii, i.e.

$$dist < |B_r - A_r| = \textbf{Circle Inception} \tag{3.40}$$

Area of Intersection Given the equations for x and y shown in the last section. The angles θ_A and θ_B can be calculated. The idea for calculation is simple, the intersection area is the area of the segment minus the area of the triangle.

$$\begin{aligned} \theta_A &= 2sin^{-1}\left(\frac{y}{A_r}\right) \\ \theta_B &= 2sin^{-1}\left(\frac{y}{B_r}\right) \end{aligned} \tag{3.41}$$

From this the area of the two triangles are:

$$\begin{aligned} area_{t,1} &= y\sqrt{A_r^2 - y^2} \\ area_{t,2} &= y\sqrt{B_r^2 - y^2} \end{aligned} \tag{3.42}$$

Finally the area of the intersection is calculated using:

$$area = A_r^2 sin^{-1}(\frac{y}{A_r}) + B_r^2 sin^{-1}(\frac{y}{B_r}) + y\left(\sqrt{A_r^2 - y^2} + \sqrt{B_r^2 - y^2}\right) \tag{3.43}$$

Intersection Over Union In order to better understand the likelihood the circle approximate is used. This, although not exact as Gaussians are a bell shape, is a fast and closed form approximate for the probability of collision. The IOU of the circles is the best measure, a higher IOU clearly means a more probably crash. It can be calculated from previous results:

$$IOU = \frac{area_{intersection}}{area_{c1} + area_{c2} - area_{intersection}} \tag{3.44}$$

Moving Objects It is important to check that the object itself is moving. If the cyclist is heading towards a parked car it is less likely that the cyclist has not seen it or that the object will cross their path. So the final check is a simple threshold of the Kalman filter prediction. As part of the state, the velocity of the object is predicted which informs the collision detection module if the object in question is moving or not.

Chapter 4

Experimental Results

4.1 System Evaluation

4.1.1 Mask-RCNN

In applying Mask-RCNN to the specific urban environment there have been a number of interesting outcomes. On the whole, the system is robust, often detecting images well and so there are rarely gaps when an object is detected in one frame and not the next. A number of incorrect classifications often provide a worry, for example, the classification of road paint can cause a problem to the system (Figure 4.1). This often happens with road markings of bicycles that are relatively convincing examples, this is perhaps solvable with using these examples as part of the null class during training and could be investigated in future work.

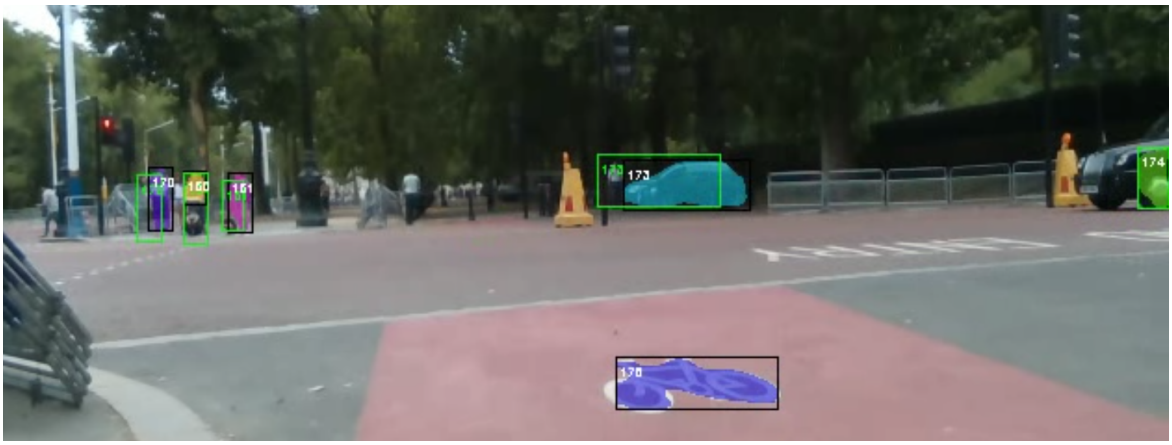

Figure 4.1: Mask-RCNN failure

4.1.2 Cyclist Model

The cyclist model is effective as proven in Section 3.4.3, however it currently requires a new model, with respective inputs and convolutional layers. This is both slow and memory intensive, the best solution would be to integrate the model into Mask-RCNN and share some of the convolutional layers. It is a significant modification however as Mask-RCNN does not use sequences of data so a novel architecture would need to be investigated.

The other, far simpler solution is to make use of the Kalman filter mentioned in Section 3.3.4, this is less complex and also faster than a neural network model. It does however have drawbacks in accuracy that the network does not, so implementation would be on a hardware specific basis. The faster the hardware, the more viable a complex model is.

4.1.3 Depth Camera

The Intel Realsense depth camera is typically used in indoor environments as it operates with a small laser. This, as is expected works less well in highly lit environments. The extent of the problem, however is often drastic and can cause Okvis to perform poor estimations of pose and hence is unable to determine the bicycle location. This problem could be eliminated completely by using a stereo camera system or by integrating and relying on GPs in bright lighting conditions.

4.1.4 Crash Sequence Evaluation

In order to determine the effectiveness of the system, it is important to know that the system will not produce numerous false positives and be an irritation to users, but also correctly activate when required. Therefore there are two sequences of images collected, one with a staged incident where the values have been tweaked to be more sensitive, and hence reduce the danger in conducting the experiment. The other sequence contains a parked car, which is detected but also determined not to be moving and hence of no danger to the rider.

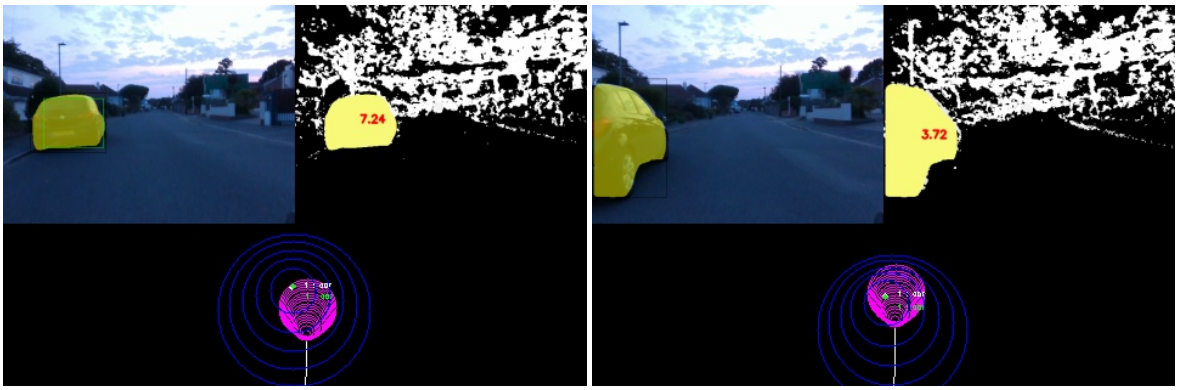

Figure 4.2: Parked Car Evaluation

It is hard to get a sequence into the report but comparison to the staged crash shows that it is evident the car is detected but not deemed a threat to the cyclist. A common problem with the system is the depth perception - this can often cause some objects to appear to move slightly as the depth measurements past around 10m are inconsistent. So a movement threshold approach is used, and slight movement cannot set off the alarm. The circles around the bike and the car represent the future positions, the centre is the predicted position and the radius the uncertainty.

The values used in this experiment were tweaked to make it slightly more responsive. This is done by adjusting the IOU threshold of collision discussed in Section 3.4.4. It was simply because it is hard to find a willing volunteer to be cycled at with enough speed and proximity

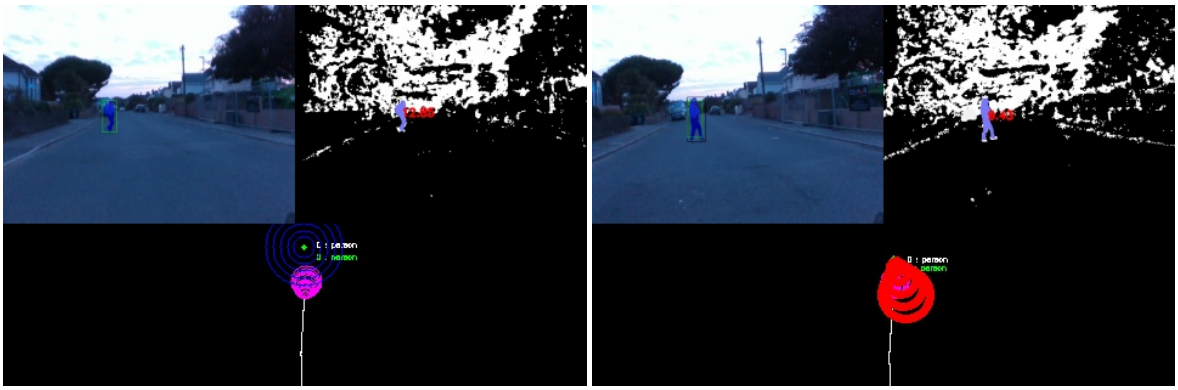

Figure 4.3: Left: Pedestrian First Detected, Right: System detects potential collision

to warrant a warning, so slower speeds and higher sensitivity was used. Both sequences can be found in the appendix (7.1,7.2), it is clear that the system is not perfect but works effectively enough to alert a cyclist of oncoming danger. With real-time capability it could become a very useful tool for a modern city cyclist.

Chapter 5

Conclusion

5.1 Future Work

5.1.1 Speed

It is evident that this product is much needed in the arsenal of a cycling commuter but in its current state it is not able to be implemented. The sub-real-time results are too slow for effective use. Therefore, with simpler updated hardware and software it may become less accurate but would be useful in the cycling environment until newer technology is available that can increase accuracy to the current level. Also a pure implementation in a faster language such as $c++$ could help with this, most of the functions used e.g. (Keras, numpy, Tensorflow) implement c sub-routines, but other parts of the code could equally be sped-up by being ported to c or $c++$ instead of python.

5.1.2 Object Detection

At the time of writing, that the only real-time method of object detection that could be applicable to cycling is tiny-yolo [12]. This is a bounding box approach that runs at c. 80fps on a laptop gpu, and can now be used on a mobile. With this, however comes the deficit of being unable to segment the image like in Mask-RCNN for depth prediction. A lightweight version therefore could use the depth approximation developed in Section 3.2. Although not perfect, combined with a 3D appreciation of the environment it could be effective, and most importantly fast enough for use.

5.1.3 SLAM

Although Okvis is effective, the poor quality depth camera images render it less reliable outside than indoor environments. This would typically not be acceptable on a final system so perhaps incorporating GPS measurements as well as Okvis would be beneficial. Okvis is great for areas where there is no GPS available, such as tunnels, where lighting is also reduced. Okvis is capable of running from a single camera and so the removal of a depth stream would also not hinder it greatly, especially if a stereo camera system was employed instead which could lead to a cheaper final implementation.

5.2 Contributions

In section 3.2, it was stated how poor hardware accuracy with regards to depth estimates can be modified with prior knowledge of object sizes in segmented images. This both reduces depth outliers and provides the Kalman filter with less noisy data to track.

Section 2.3.1 discusses how to modify the KCF tracker (Section 2.3.1) for use in the multi-class case. This is then later used in combination with Brisk, class, and Kalman matching for object identification between each frame Section 3.3.6.

The final contributions of this report involve the tracking and prediction models for both objects (Section 3.4.2) and cyclists (Section 3.4.3). The object model used a modification to a Kalman filter to get the predictions and uncertainties whilst a novel recurrent neural network architecture was developed for bike model prediction. The network in question learns from both image and pose data to predict the future bicycle state. From here the object collision module can use the states and uncertainties to calculate the intersection over union of the future states and determine if a collision is likely.

Summary This project combines numerous fields of research into a comprehensive hazard detection system. Each section of the report briefly explains the detailed understanding of the respective methods that was required to integrate them into the final system. It is by no means a completed system but shall continue to be developed after this report is submitted. The goals of the project have been met, namely to be capable of detecting and alerting the cyclist to potential hazards. The limiting factor to a real-time solution is that the state-of-the-art technology available is yet to run real-time (even on dedicated hardware). However, the system is designed in a modular fashion so that when advances in respective fields occur it can be modified accordingly.

Chapter 6

Ethics

6.0.1 Modification for Military Purposes

The code presented in this report, although strictly for civilian applications does have the potential to be modified for military use. It obviously provides the ability to track people and objects from frame-to-frame. This however, is not in a real-time context at present and would require significant modification for use in such an environment that would be conducive to a military application. However, that being said little modification is needed for a simple tracking application from CCTV data for example. This would not need to run at real time and could be a potential source of misuse.

6.0.2 Other Ethics Issues

The code developed is intended to offer a basis for an early warning system to cyclists in urban environments. Hence a number of the inherent ethical problems with, for example the autonomous vehicle space are also applicable to this project. The potential over-reliance on such a system of cyclists could be dangerous and allow for lack of attention and potential accidents when the system is unable to detect a collision. That having been said, it is evident that there is a need for such a system for cyclists and so the benefits would clearly outweighs the disadvantages in such a case.

6.1 Ethics Checklist

	Yes	**No**
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		✓
Does your project involve the use of human embryos?		✓
Does your project involve the use of human foetal tissues / cells?		✓
Section 2: HUMANS		
Does your project involve human participants?		✓
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from Human Embryos/Foetuses i.e. Section 1)?		✓

Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?		✓
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		✓
Does it involve processing of genetic information?		✓
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		✓
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		✓
Section 5: ANIMALS		
Does your project involve animals?		✓
Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?		✓
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		✓
Could the situation in the country put the individuals taking part in the project at risk?		✓
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		✓
Does your project deal with endangered fauna and/or flora /protected areas?		✓
Does your project involve the use of elements that may cause harm to humans, including project staff?		✓
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		✓
Section 8: DUAL USE		
Does your project have the potential for military applications?	✓	
Does your project have an exclusive civilian application focus?	✓	
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		✓
Does your project affect current standards in military ethics e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		✓
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		✓

Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?	✓	
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		✓
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		✓
Section 10: LEGAL ISSUES		
Will your project use or produce software for which there are copyright licensing implications?		✓
Will your project use or produce goods or information for which there are data protection, or other legal implications?		✓
Section 11: OTHER ISSUES		
Are there any other ethics issues that should be taken into consideration?		✓

Chapter 7

Appendix

7.0.1 RoI Matching Algorithm

Algorithm 7: Algorithm to match old and new frames

match

$(old\_frame), new\_frame, [matching\_methods(oldROI, newROI))], matches, threshold)$

Input :

old_frame // RoIs as structures from the old frame

new_frame // RoIs as structures from the new frame

$[matching\_methods(oldROI, newROI)]$ // List of pointers to functions for matching RoIs

// (KCF,Kalman Filter,Brisk)

$matches$ // A matrix of zeros $\in \mathbb{R}^{(methods, oldROIs, newROIs)}$

$threshold$ // The minimum match score considered to denote an inter-frame RoI match

Output:

$old\_matched$ // A list of matched oldROIs, $old\_matched[i]$ matches to $new\_matched[i]$

$new\_matched$ // A list of matched newROIs, $new\_matched[i]$ matches to $old\_matched[i]$

// Calculate score of match $(oldROI \rightarrow newROI)$ for each $method, oldROI, newROI$

foreach $oldROI \in old\_frame$ **do** /* For every RoI in last frame */

  foreach $newROI \in new\_frame$ **do** /* For every RoI in current frame */

    foreach $method \in matching\_methods(oldROI, newROI)$ **do** /* For each method */

      $matches[method_i, oldROI_i, newROI_i] = matching\_methods(oldROI, newROI)$

    end

  end

end

$matches = matches[0, :, :]$

```
// Multiply along methods axis to produce one combined match matrix.
foreach method ∈ matching_methods(oldROI, newROI) do        /* For every method */
    if method_i > 0 then                                    /* If method index is > 0 */
        matches[0, :, :] = mult(matches[0, :, :], matches[method_i, :, :]) ; /* Multiply along axis
         */
end
```

```
   // Match new and old RoIs.
max = matches.max()
 while max > threshold and len(new_matched) < len(new_frame) do
    old, new = matches.max()_{i,j} ;                        /* Find index of max element */
    if old ∉ old_matched and new ∉ new_matched then         /* If not already matched */
        new_matched.append(new)                             /* Add new index to already matched list */
         old_matched.append(old)                            /* Add old index to already matched list */
    matches[new, old] = −1
    max = matches.max()
end
```

7.1 Crash Sequence

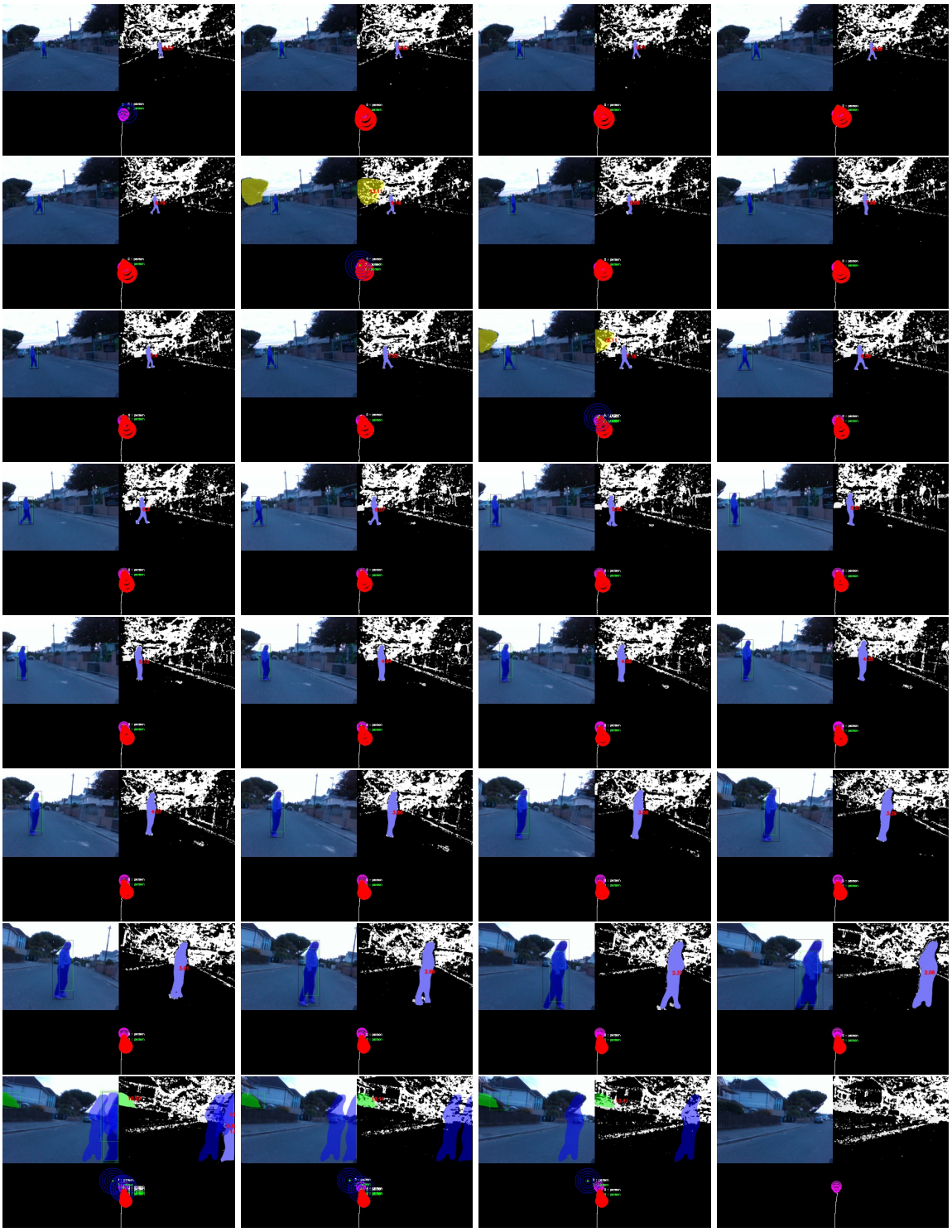

Figure 7.1: Staged Crash Sequence

7.2 Parked Car Sequence

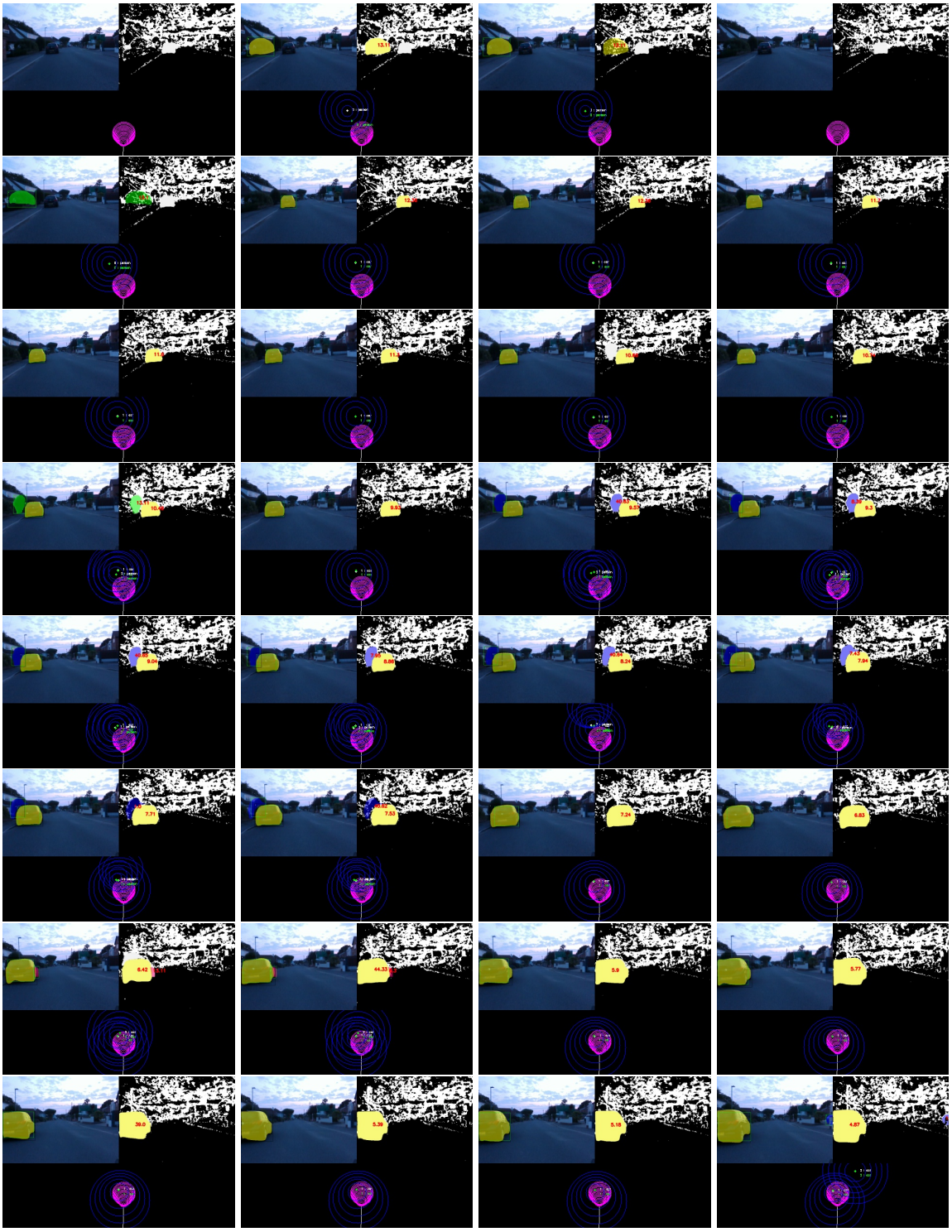

Figure 7.2: Parked Car Sequence

Bibliography

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

[2] D. F. Transport, "Ras50001: Contributory factors in reported accidents by severity, great britain, 2016," report, Dft, 2016. pages 1

[3] D. F. Transport, "Focus on cycling in reported road casualties great britain 2013," report, Dft, 2013. pages 1

[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. pages 3

[5] O. R. et al.and Jia 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. pages 3

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," vol. 25, 01 2012. pages 3

[7] A. Karpathy, "Networks," 2017. pages 4

[8] A. Karpathy, "Networks," 2017. pages 4

[9] A. Karpathy, "Networks," 2017. pages 4

[10] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *CoRR*, vol. abs/1710.09829, 2017. pages 4

[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. pages 5, 6

[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. pages 5, 49

[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"*, "2013". pages 5, 7

[14] P. F. "Felzenszwalb and D. P. Huttenlocher, ""efficient graph-based image segmentation"," *"International Journal of Computer Vision"*, "2004". pages 5

[15] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. pages 6, 7

[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. pages 6

[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. pages 7, 10

[18] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014. pages 8, 9

[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. pages 9

[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. pages 9

[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. pages 9

[22] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. pages 9

[23] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. pages 10

[24] C. Olah, "Understanding lstms blog." pages 11

[25] M. Deisenroth, "Gaussian processes and bayesian optimisation," 2018. pages 13

[26] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," 2010. pages 13

[27] O. docs, "Images," 2012. pages 16

[28] 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. pages 17, 20, 23

[29] H. F. Durrant-Whyte, "Uncertain geometry in robotics," vol. 4, pp. 23 – 31, 03 1988. pages 20

[30] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): Part i," 10 2006. pages 20

[31] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *2011 International Conference on Computer Vision*, pp. 2548–2555, Nov 2011. pages 20, 22

[32] O. docs, "Images," 2012. pages 21

[33] P. Furgale, “Extensions to the visual odometry pipeline for the exploration of planetary surfaces,” 01 2011. pages

[34] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, “High-speed tracking with kernelized correlation filters,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, pp. 583–596, March 2015. pages 23, 25

[35] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2014. pages 39

[36] R. Eisele, “Circle intersection,” 2016. pages 44

[37] R. Eisele, “Circle intersection,” 2016. pages 44