STM JOURNALS

International Journal of
Machine Systems and
Manufacturing Technology

Volume 1, Issue 2, 2023
July—December
DOI (Journal): 10.37591/IJMSMT

https://journals.stmjournals.com/ijmsmt

Research IJMSMT

Cyclist Safety Enhancement: A Multi-Modal Hazard Detection System

Naval Joshi[1,*], Vaishnavi Bhutada[2], Manaswini Chittepu[3]

Abstract

This study presents a multi-modal hazard detection system to enhance cyclist safety in urban environments. Lever- aging a combination of computer vision, object tracking, and predictive modeling, the system offers a comprehensive approach to identifying and mitigating potential risks. Key contributions include improved depth estimation through object size priors, multi-class tracking utilizing KCF and Brisk, and a novel recurrent neural network architecture for predicting bicycle movement. The system's collision detection module evaluates the likelihood of collisions by analyzing future object and cyclist positions, incorporating uncertainties. While real-time implementation remains challenging due to hardware limitations, the modular design allows for future optimizations and integration with faster technologies. This research represents a valuable step toward safeguarding cyclists and paves the way for more effective hazard detection systems in the future.

Keywords: KCF, cyclist safety, urban environments, SLAM, RCNN

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, protecting and modifying bikes with a comprehensive safety system is important and could save lives.

Road Map

The report develops a collision detection system 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.

***Author for Correspondence**
Naval Joshi
E-mail: n6832319@gmail.com

[1]Student, University of Petroleum and Energy studies, Uttarakhand, India
[2]Student, Bharati Vidyapeeth's College of Engineering for Women, Pune, Maharashtra, India
[3]Student, Amrita Vishwa Vidyapeetham, Hyderabad, India

Received Date: March 14, 2024
Accepted Date: March 20, 2024
Published Date: May 09, 2024

Citation: Naval Joshi, Vaishnavi Bhutada, Manaswini Chittepu. Cyclist Safety Enhancement: A Multi-Modal Hazard Detection System. International Journal of Machine Systems and Manufacturing Technology. 2023; 1(2): 35–83p.

Background

This chapter discusses the current re- search into instance segmentation, recurrent networks, optimization, and SLAM. It first describes recent advances in a related computer vision methods i.e., Mask RCNN. Mask RCNN combines several advances in different fields: image classification, detection, and segmentation. The chapter then continues with a discussion of recurrent networks for path prediction and methods of optimizing these networks. Finally a brief discussion on SLAM and object tracking, both highly active and voluminous research fields in their own right.

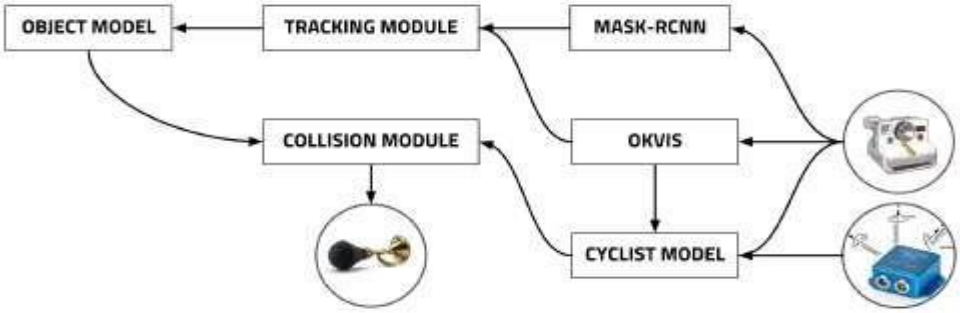


Figure 1. System design.

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 combines the advantages of the respective algorithms and mitigates their potential weaknesses. Once the ability to track objects is discussed, models for predicting future locations are developed. This leads to a novel recurrent network to predict future bicycle locations from image and pose data 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 in Figure 1.

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.

BACKGROUND, COMPUTER VISION

Deep Learning Preliminaries

Deep learning frameworks have recently revolutionized computer vision and its applications. This is an image classification task; the photo contains one object in the dataset, which generally occupies most of the frame. Similar networks are used with additional features for image detection and segmentation, integral to detecting moving objects, such as pedestrians and vehicles in an urban environment.

Cutting-edge versions deploy neural networks. These networks are characterized by a series of non-linear nodes organized in layers. At each layer, the network progressively learns abstract and higher-level features of the image to be detected until the final (usually dense layer), which determines the predicted class. More layers translate into a more profound complexity of features that a network can learn. This is, however, only a basic representation of the problem, and some clever and shallower architectures can outperform deeper networks. It is important to note that the depth and width bear a negative cost that adversely affects prediction speed while increasing memory requirements 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.

Convolutional Networks

A CNN generally contains convolutional, pooling, fully connected, and normalization layers. This form of network became particularly popular in computer vision after Geoffrey Hinton's 2012 paper [4], with an ImageNet classification error of c.16%, significantly lower than the best of contemporary techniques, which peaked at 26%. Convolutional networks assume each input to be an image and then build specific characteristics or features directly into how the network is set up. This allows a significant reduction in the number of network parameters.

A regular neural network, as mentioned earlier in Section III-A, has numerous densely interconnected layers, meaning that all neurons within a given layer are individually connected with corresponding neurons in the preceding and succeeding layers. However, convolutional neural network sizes are described based on how wide, high, or deep they are (Figure 2). In this architecture, unlike in the fully connected case, there is a significant reduction in the parameter space [5].

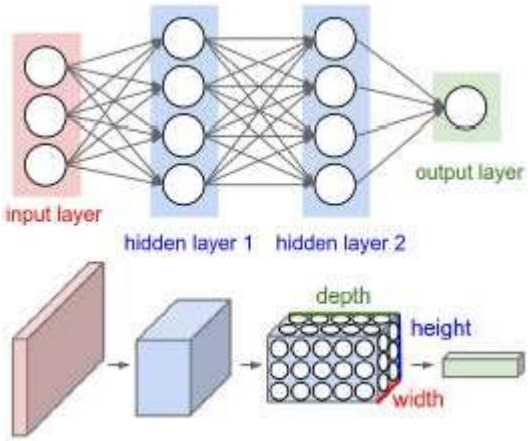


Figure 2. Regular Neural Network (Left) [5], CNN (right) [5].

Convolutional Layers

A convolutional layer learns a set of filters, each filter is compact, say (7×7×3). This filter slides (convolves) over the image, and then the dot product is taken between the filter values and the inputs to that filter (Figure 3). The filter slides across with a respective stride, a hyper-parameter; a stride of one equates to moving one pixel at a time. A convolutional layer can, therefore, pick up features such as a corner or higher-level features such as fur texture.

This process creates a 2D activation map containing the filter's response at each position on the input volume. Several filters are passed over, and many activation maps relating to the layer depth are created, a tune able hyper-parameter.

Pooling Layers

They can be utilized to minimize how many parameters are required in a network. This is a much-debated topic; Geoffrey Hinton discusses the problems of convolutional networks in his 2017 paper [8] and especially focuses on the disadvantages of pooling layers. It is, however, a current, although fading, method of parameter reduction. This is still a recognized and useful method for optimizing network architecture for smaller hardware, such as portable on a bike.

A pooling layer such as that depicted in (Figure 3) passes a kernel over the input (much like in a convolutional layer), and at each spatial location, it calculates a metric. For example, max pooling takes the largest value contained within the kernel grid at each consecutive spatial location. This process reduces the input size with respect to the kernel size and the implemented stride.

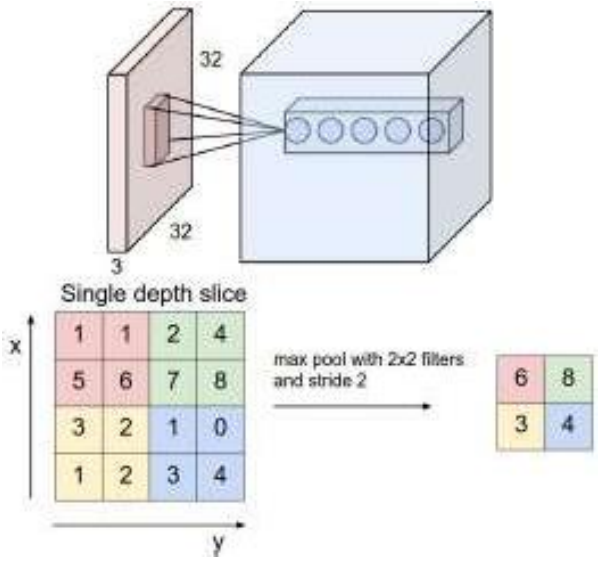


Figure 3. CNN Filter (left) [6], Max Pooling [7].

Image Detection

Image detection is an area of research that uses the CNN networks described in Section III-B. 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 it at each location. When the network classifies a cat at a particular window location, a cat is likely located in that area.

Although the aforementioned approach is valid, it would be prolonged; not only would it be necessary to slide the window over the whole image, but the size 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 several differing approaches to this problem R-CNN [9] and other later iterations vs Y-O-L-O [10]. The R-CNN approach resembles 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. [9] 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 aver- age 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 [11]. This uses a fast segmentation algorithm by Felzenszwalb [12] 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 several metrics to pick up the differences between regions during the growth period and limit one region's growth at the expense of others (hence the size metric).

The results of the region proposal (Figure 4) are a series of windows that are likely to contain an object. These are then warped to 227*227 pixels to fit the input dimensions of the following CNN network, 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 III-C2.

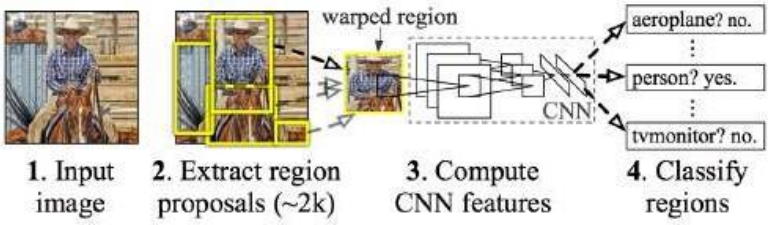


Figure 4. R-CNN Region Proposal & Architecture [9].

Fast-RCNN

R-CNN performs a CNN forward pass for each region proposal and does not share computation between regions [13]. Spatial pyramid pooling networks (SPPnet) [14] 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 about 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 5). Each RoWe is then pooled into a fixed-size feature map as in SPPnet and then mapped to a feature vector by fully connected layers. As seen in Figure 5 the network 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 1).

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

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

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

in which

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

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

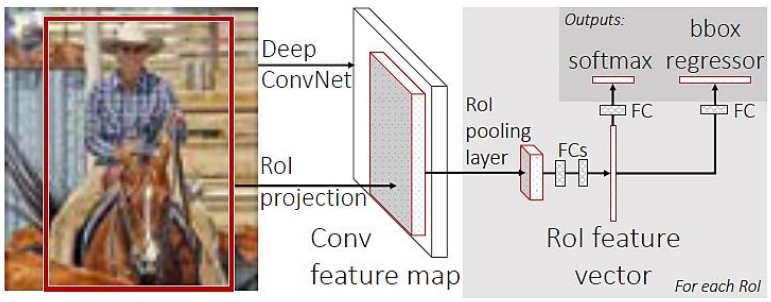


Figure 5. Fast R-CNN Architecture [13].

Figure 5 depicts the network architecture, this incorporates a RoWe pooling layer which maps the convolutional feature map from an RoWe (created by selective search) to a fixed extent feature map of

a set *H W*, where *H* and *W* are layer hyper-parameters Each RoWe is a rectangular window defined by a tuple of four elements (*r, c, h, w*), where (*r, c*) specify the top left corner and (*h, w*) the width. RoWe max-pooling divides the RoWe window into a grid and then max-pools each grid into the corresponding output cell. The grid size is proportionate to the RoWe, so the output shape is constant regardless of RoWe input size.

Faster R-CNN

Although the previous method [13] re- moved the need for a multi-stage classification pipeline (CNN and SVM), it still relies on the selective search [11] method for region proposal. This part is now the computational bottleneck that slows down the process. Faster R-CNN proposed by Ren et al. [15] seeks to alleviate this issue by combining region proposal into the CNN. The paper suggests using a region proposal network (RPN) to produce the RoIs used later in the network.

The region proposal network (RPN) outputs a set of rectangular object proposals, each with an objectiveness score, i.e., how likely the network thinks the proposal contains an object. This part of the network comes after several 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 6). At each sliding window location, several 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 4*k* results, which represent the encoding coordinates of the *k* boxes, and the *cls* later scores the probability that there is an object or no object in each proposal; therefore, cls has 2*k* 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. If no positive examples are found for the first positive case, the boxes with the highest IoU are labeled positive. With this, it follows that:

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

Where *i* is the anchor index, p_i is the predicted probability of anchor *i* containing an object. The ground truth label p_i is denoted as a 1 for a positive anchor and 0 for a negative; therefore, L_{reg} is only activated for a positive example, t_i and t^*i represent the 4 coordinates of the bounding boxes, and L_{cls} is the classification loss. The two terms are normalized (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 pro- posals per image. Non-maximum suppression is a technique

that sets all neighboring areas to zero around a local maximum and, hence only keeps the maximum value and suppresses the others.

Image Segmentation

Unlike the image detection problem described in Section III-C, this task aims to segment an image on a pixel-by- pixel basis into respective classes (Figure 7). 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 spatial information used in a segmentation task. Therefore, Long et al. [16] propose converting these final layers to convolutional layers, maintaining the spacial information and creating a heat- map output (Figure 7).

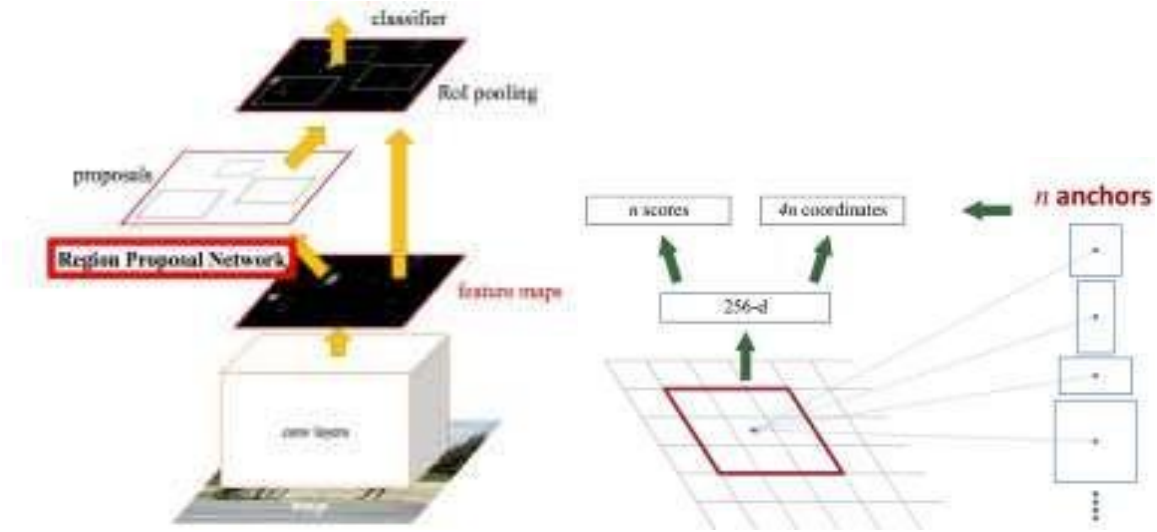


Figure 6. Fast R-CNN Architecture [13].

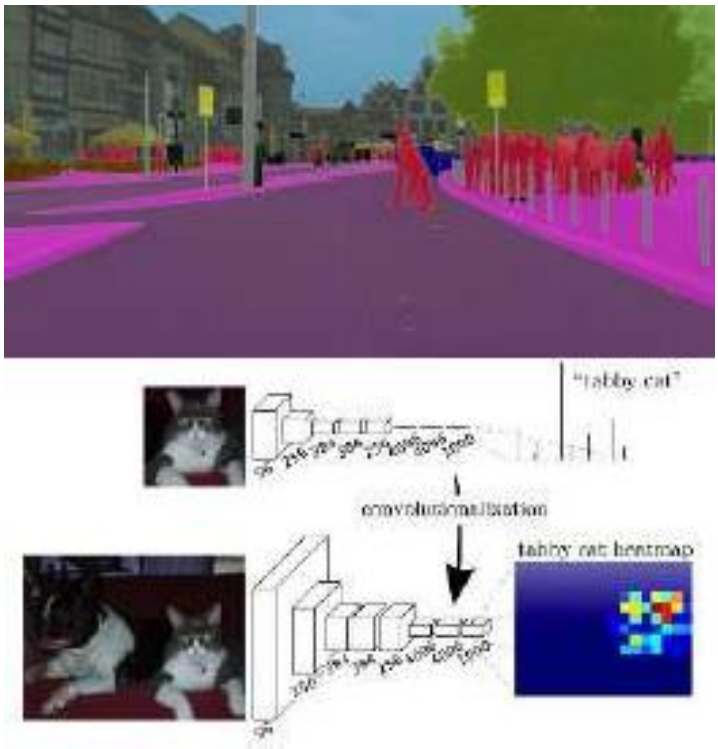


Figure 7. Image Segmentation [17], Convolutional Heat Map [16].

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 deconvolution. Up-sampling, therefore, with factor f is convolution with a fractional input stride of $\frac{1}{f}$. Each filter is placed on the output image and 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 several ways but the main accepted approach is to learn them as part of the tuning process simply.

An important concept is patch-wise training, which feeds the network several patches from an input image (small patches surrounding objects of interest) instead of an entire image. This helps balance the classes, ensures the input has enough variance, and correctly represents the input set. However, this paper argues that this can be done from a fully convolutional training regime by incorporating a Drop-connect-like mask [18] between the output and the loss. A Drop-connect mask is similar to the proposed regularisation technique of Dropout [19] 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. [16] 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 [20]. 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.

Instance Segmentation

Instance segmentation is a problem that combines research in Sections III-C & III-D; it aims to not only perform pixel- wise segmentation of classes but also determine instances of objects within an image. As discussed in Section III-E1, combining these networks into one classification pipeline is a simple method.

Mask R-CNN

Mask R-CNN was proposed by Facebook AWe Research (FAIR), namely Kaiming He et al [14]., in March 2017. It uses the Faster R-CNN network described in Section III-C3, which builds from advances in SectionsIII-C1& III-C2. The method works by simply adding a branch from the existing network for predicting segmentation masks as in Section III-D (Figure 8). The branch is a small FCN applied to each RoWe for segmentation. Some slight modifications to each approach to produce the final Mask R-CNN are documented below.

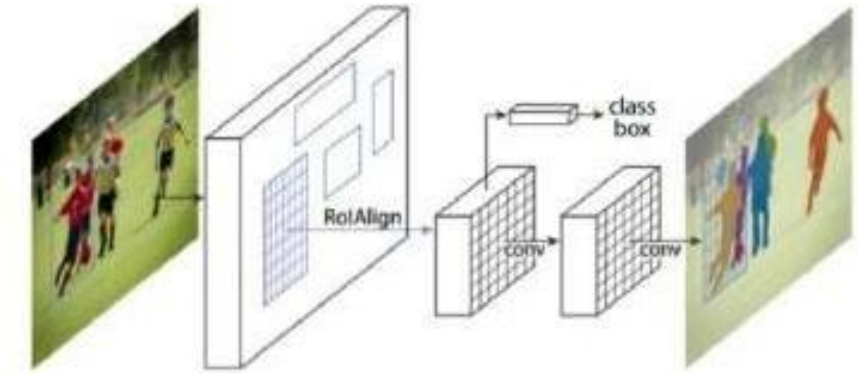


Figure 8. Mask R-CNN [21].

RoIs are produced from the RPN as in Faster R-CNN [15] then a $m \times m$ mask is predicted from each RoWe without the use of a fully-connected (fc) layer and therefore keeps spacial information as in Section III-D. 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 from disrupting each other while optimizing 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 branches: mask, class prediction, and bounding-box prediction (Equation 5).

$$L = L_{cls} + L_{box} + L_{mask} \tag{5}$$

where L_{cls} & L_{box} are identical to Section III-C2. 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.

RoWe pool as discussed in Section III-C2 is modified in this approach. Instead of subdividing a RoWe into discrete regions, e.g., a grid of 4 × 5 cells. An RoWe does not always divide equally into these grid cells, so rounding of grid sizes is needed; this quantization of sub-windows can cause misalignments of the RoWe and extracted features. The new RoIAlign layer introduced in this paper mitigates this problem by not rounding grid cells. Mask R-CNN uses bilinear interpolation (linear interpolation with 2 dimensions) to calculate the provided characteristics.

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

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 to classify a sequence, the data must be passed through the layer simultaneously. The advantage of a recurrent layer is 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 9 for more details. Each state {1, 2, ..., $t - 1$, t} is connected almost like a chain.

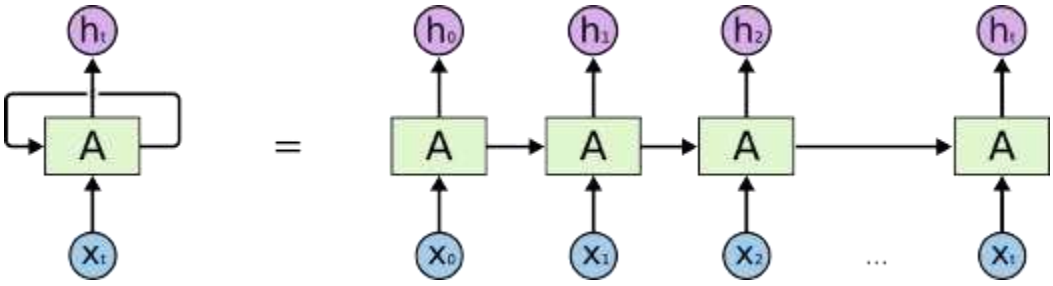


Figure 9. A Recurrent Layer [22]. The output of the cell at time t, from inputs x_t is h_t.

The main issue with this recurrent layer is how far back the network can remember. 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 better at remembering and do not get the vanishing gradient problem that RNN layers are subject to. LSTMs contain a cell state that 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 latter and is similar to the links in the RNN chain (Figure 10). The gates contain sigmoid functions, which can evaluate 0 and 1 and determine how much of a component will be let through.

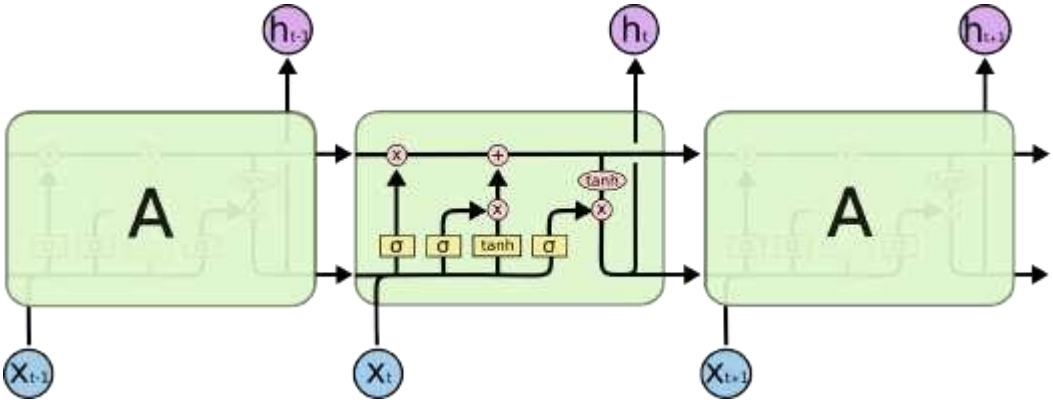


Figure 10. A LSTM (Long-Short Term Memory) Layer [22].

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 acts like a perceptron, multiplying inputs by weight, adding a bias, and then passing the result through the activation (Sigmoid) function (Equation 6).

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{6}$$

Once again, h_{t-1} is the layer output 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 7) and a *tanh* section that creates the new vector $\tilde{C}_t$ to add to the current state C_t, Equation 8.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{7}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{8}$$

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 9.

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \tag{9}$$

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 10, 11.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{10}$$

$$h_t = o_t \times tanh(C_t) \tag{11}$$

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 modeling 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 optimized to perform well at their respective tasks. This report keeps Mask-RCNN as default, but the prediction model architecture is bespoke and thus must be optimized for its use case.

Bayesian Optimisation of Layer Dimensions

To get the best network architecture and hyper-parameters, such as learning rate, it is possible to try combinations and record which performs the best naively. However, when optimizing networks efficiently, it is better to provide a range of possible working parameters and optimize them in an informed and, more importantly, automated manner. Bayesian optimization allows such a method and can quickly check the parameter space for a good combination to minimize network loss. Bayesian optimization uses an acquisition function that evaluates a Gaussian process. Evaluating a proxy function approximating the true function is cheaper than running the neural network. Hence, it is a very fast and effective method of optimization.

Gaussian Processes

A typical regression sample aims to determine the parameters of a function (or family of functions) best describe some data. A Gaussian process involves assuming 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 best approximate the relationship between these points via a covariance kernel $k(\ ,\ )$ and a mean function $m(\ )$. A Gaussian distribution is described by Equation 12 and a kernel function (Matern) by Equation 13.

$$p(x|\mu, \Sigma) = (2\pi)^{\frac{D}{2}} |\Sigma|^{-\frac{1}{2}} e^{(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu))} \tag{12}$$

$$k_{Mat,3/2}(x_i, x_j) = \sigma_f^2 \left(1 + \frac{\sqrt{3}||x_i - x_j||}{l}\right) e^{\left(-\frac{\sqrt{3}||x_i - x_j||}{l}\right)} \tag{13}$$

where l is a hyper-parameter that describes the smoothness of the data and σ_f the amplitude of the latent function. Figure 11 gives an example of a Matern kernel.

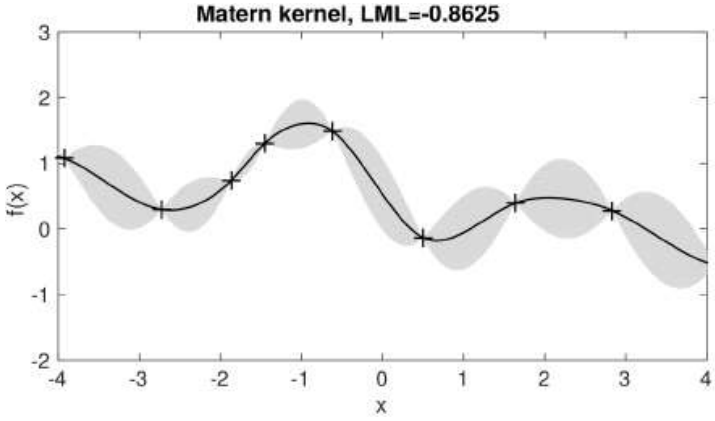


Figure 11. Gaussian Process: Matern Kernel [23].

Bayesian Optimisation

Bayesian Optimisation is a process that can maximize 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 is collected to initialize 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 [24], is presented in Equation 14.

$$\alpha(\mathbf{x}) = -(\mu(\mathbf{x}) - (k)\sigma(\mathbf{x})) \ , \ k > 0 \tag{14}$$

The lower confidence bound balances the exploration of global minima by the covariance function and weighting k, which increases at each iteration by exploiting local minima with the mean function. The Bayesian optimization 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 */

BACKGROUND, SLAM

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

Sensors

The sensors used in this project are an RGBD camera and an IMU. These are available in the Intel Realsense Camera (ZR300, Figure 12). This camera has a USB cable that provides power and sends information. An extension cable allows a laptop to be attached to the camera from a backpack whilst cycling. Potential problems with this setup are due to the juddering of the bike on the road; this could cause tricky footage to analyze and data loss when the hard drive is written to disk. The latter issue can easily be solved by writing to an SSD while cycling.

Figure 12. Intel Realsense Camera (ZR300).

Camera Distortion

To fully understand the bike's lo- cation, the environment can be considered from two frames of reference: the world and the 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 13).

Figure 13. No Distortion (left) [25], Positive/negative radial distortion (middle/right).

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 15)

$$r^2 = x_1'^2 + x_2'^2 \tag{15}$$

$$\mathbf{x}'' = \mathbf{d}(\mathbf{x}') = \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} \begin{bmatrix} x_1' \\ x_2' \end{bmatrix} \tag{16}$$

where x′ 1, x′ 2 is the 3d point mapped to the unit plane in z, $k_1 - k_6$ are radial distortion parameters, p1, p2 are tangential distortion parameters. The model parameters can be determined during camera calibration.

Camera to World Frame

Calculating the 3D Ray

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 14. 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 18 shows how to account for this. Hence to find a ray describing a 3d point, the image must first be scaled (Equation 17).

$$\mathbf{x}'' = \mathbf{k}^{-1}(\mathbf{u}) = \begin{bmatrix} \frac{1}{f_1} & 0 \\ 0 & \frac{1}{f_2} \end{bmatrix} \left(\mathbf{u} - \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \right) \tag{17}$$

where f_1, f_2 are x/y focal lengths in pixels, and c_1, c_2 is the principal point (image center) in pixels. The point can be projected to its undistorted location using the reverse of Equation 18.

$$\mathbf{x}' = \mathbf{d}^{-1}(\mathbf{x}'') \tag{18}$$

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

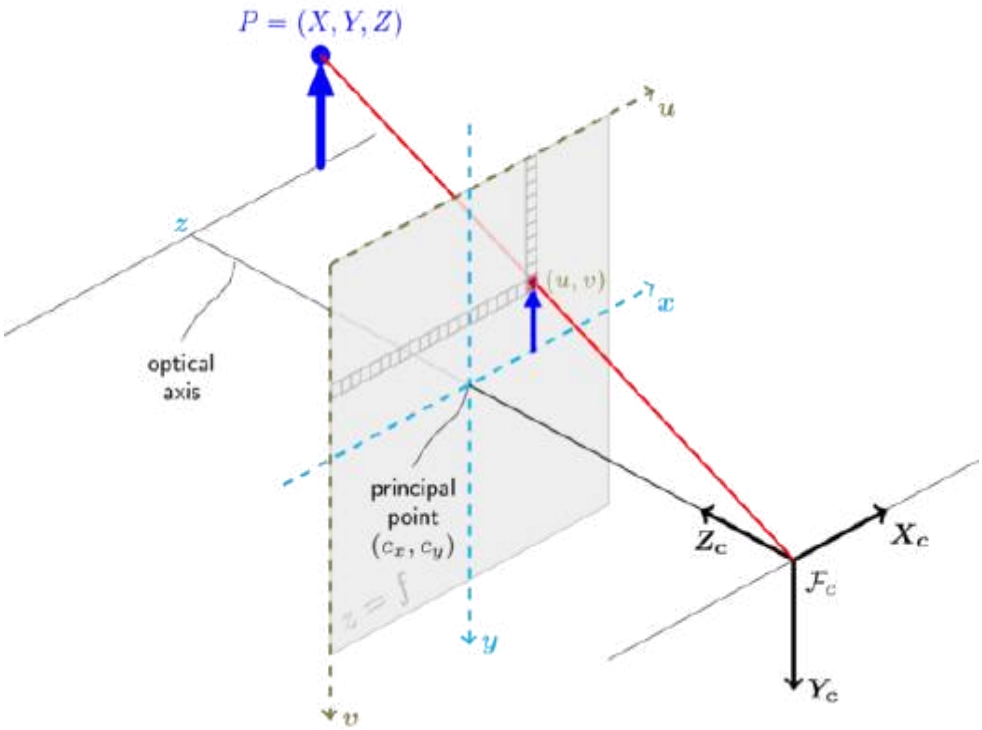


Figure 14. Pinhole camera model.

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:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}' \times depth(\mathbf{x}') \\ depth(\mathbf{x}') \end{bmatrix} \tag{20}$$

Assuming a static camera location, this would be all needed to map the space in front of the camera. Yet, when this camera undergoes motion, the reference point and alignment of the camera frame shift

in relation to the global 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.

Projecting to World Frame

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;

$$\begin{aligned} q &= q_w + q_x We + q_y j + q_z k, \\ i^2 &= j^2 = k^2 = ijk = -1 \end{aligned} \tag{21}$$

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:

$$\begin{aligned} T_{WC} &= T_{WS} \cdot T_{SC} \\ p_W &= T_{WC} \cdot p_C \end{aligned} \tag{22}$$

A transformation between frames is represented as T_{BA} where B and A represent reference frames, so T_{WC} converts between the world frame and camera frame. It is composed of a position and orientation transformation:

$$T_{WC} = \begin{bmatrix} C_{WC} & r_{WC} \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{23}$$

where $T_{WC} \in R^{4\times4}$, the position translation matrix $r_{WC} \in R^{3\times1}$, and the orientation matrix $C_{WC} \in R^{3\times3}$. 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

$$\begin{aligned} C_{CW} &= C_{WC}^{-1} = C_{WC}^{T}. \\ T_{CW} &= T_{WC}^{-1} = \begin{bmatrix} C_{WC}^{T} & -C_{WC}^{T} r_{WC} \\ 0_{1\times 3} & 1 \end{bmatrix} \end{aligned} \tag{24}$$

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

Camera to World Summary

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 for the reverse, namely:

Algorithm 2: Algorithm to transform camera point to world frame

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 $\in R^{4\times4}$

Output: N points in the world frame

$worldPoints \in \mathbb{R}^{3,N}$

$x =$
$scale\_point(imageCentre, focalMatrix, cameraPoints)$
/* Equation (17) */
$x = undistort(cameraModel, x)$ /* Equation (??) */
$T_{wc} = matrix\_multiply(T_{ws}, T_{sc})$
$x = add\_depth(depth, x)$ /* Equation (20), and Vstack [depth,1] to bottom of x */
$worldPoints = transform\_to\_world(T_{wc}, x)$
/* Equation (22) */

Bicycle States

A complete representation of the bike's state is needed to find the bike's location and orientation in each frame. At each taken image (k), the bike can be assumed to have a set state X_R^k. Contained within X_R are a set of pose states, X_T, and speed/bias states X_{sb}, Equation 25. Landmarks are contained within the set X_L.

$$\begin{aligned} \mathcal{X}_T &:= [{}_W r_S^T, q_{WS}^T]^T \\ \mathcal{X}_{sb} &:= [{}_s v^T, b_g^T, b_a^T]^T \\ \mathcal{X}_R &:= [{}_W r_S^T, q_{WS}^T, {}_s v^T, b_g^T, b_a^T]^T \end{aligned} \tag{25}$$

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, b_g & b_a the accelerometer and gyroscope biases.

The SLAM Problem

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

$$\tilde{\mathbf{z}} = \mathbf{b}_c + s\mathbf{M}_z + \mathbf{b} + \mathbf{n} + \mathbf{o} \tag{26}$$

Algorithm 3: Algorithm to transform world point to camera frame

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 (24) */

$x = transform\_to\_camera(T_{cw}, worldPoints)$ /* Equation (22) */

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

$x = remove\_depth(depth, x)$ /* Reverse Equation (20) */

$cameraPoints = scale\_point(imageCentre, focalMatrix, x)$ /* Equation (17) */

where **z** is the correct measurement, $\mathbf{b}_c$ is a long-term constant bias, *s* is a scaling factor, **M** - misalignment, **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 its origin. This makes intuitive sense if, say, there are only IMU measurements available. At each step, more uncertainty is introduced from the noise and biases (shown in Equation 26), and the bike's position becomes increasingly uncertain.

For some time, it was also presumed to be the case even when external measurements of the environment were taken. This, however, is not the case as proved by Durrant-Whyte [27]. Due to the substantial interdependence among estimates of diverse landmarks and positions on a map, this interconnectivity tends to amplify with successive observations (Figure 16). As an example of this (Figure 16: Right) displays an 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 mi's location may be quite uncertain, the relative location between two landmarks $m_i - m_j$ is well-known.

In the case of the bicycle set-up, there is an IMU and a camera. These provide the basic inputs to the OKVIS software [26]. Okvis can determine the bike's location by optimizing the total error, combining both the camera and IMU error concerning the bicycle state. This optimization process is discussed in the following sections. Firstly, it is important to detect landmarks from one frame to another, which is discussed in the next section.

Frame to Frame Matching

Keypoint Detection

The Okvis [26] paper uses a corner detection method to determine key points and then matches these key points using a binary descriptor (BRISK [29]). 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 utilized. A simplistic way to think of this is how one would determine 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 + Δx, y + Δy). An area of rapid gradient change would have very different intensity values at (x, y) and surrounding points. Hence, a corner will maximize the sum of squared differences between points.

$$f(x,y) = \sum_{x_k, y_k \in W} (I(x_k, y_k) - I(x_k + \Delta x, y_k + \Delta y))^2 \tag{27}$$

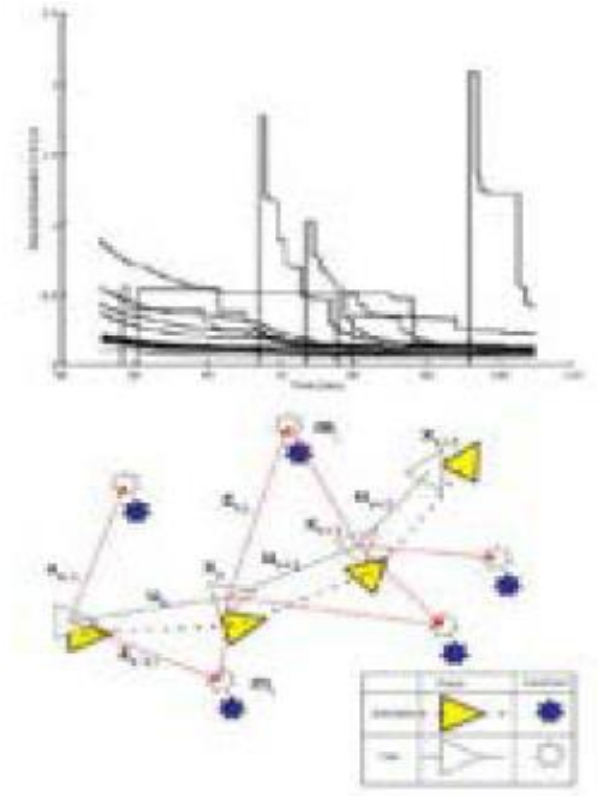


Figure 16. Left: Convergence in landmark uncertainty. Over time, standard deviations reduce monotonically to a lower bound [28]. Right: Error between estimated and true landmarks is common.

The Taylor expansion can further distill this to:

$$f(x,y) = \sum_{(x,y) \in W} (I_x(x,y)\Delta x + I_y(x,y)\Delta y)^2 \tag{28}$$

or in matrix form:

$$M = \begin{bmatrix} \sum_{(x,y)\in W} I_x^2 & \sum_{(x,y)\in W} I_x I_y \\ \sum_{(x,y)\in W} I_x I_y & \sum_{(x,y)\in W} I_y^2 \end{bmatrix} = R^T \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R \tag{29}$$

$$f(x,y) = (\Delta x \ \Delta y) M \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{30}$$

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 17.

$$R = det(M) - ktrace(M)^2 = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{31}$$

Keypoint Matching

Binary descriptors use a sampling pattern to create pairs of points (lines). The algorithm then compares the two points (p1, p2); if p1 > p2, a 1 is recorded; otherwise, it is 0. This creates a binary string that, along with orientation representation, describes a key point. When matching key points between images, a simple, fast ΣXOR reveals key point similarity from which a threshold can be used to determine matches.

The BRISK sampling pattern (Figure 18: 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 noise-prone than traditional binary detectors, e.g., BRIEF.

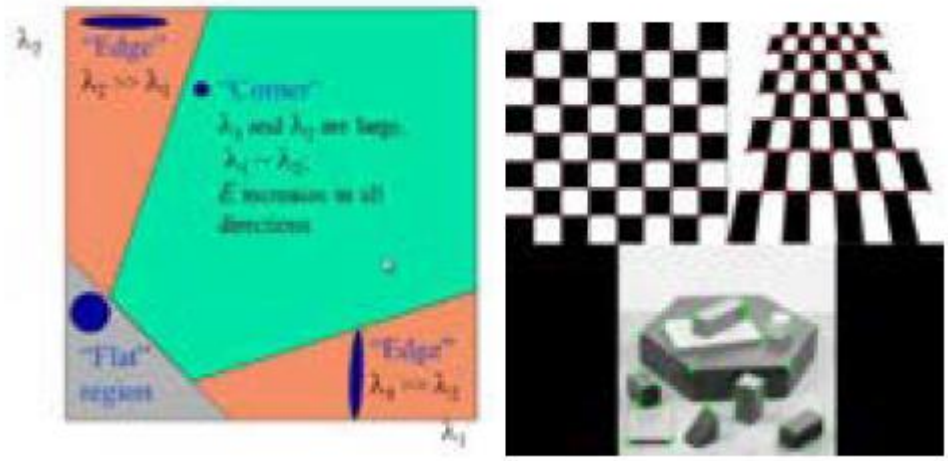


Figure 17. Left: Structure Tensor Eigenvalues, Right: Harris corner detection result [30].

Figure 18. Left: Brisk keypoint matching example, Right: Brisk Sampling Pattern [29].

Batch Non-linear Least Squares

Okvis formulates an error equation combining the IMU and camera re-projection errors. The state of the bike can then be determined by minimizing this error function. Given the error function for the camera and IMU in the form:

$$J(\mathbf{x}) := \frac{1}{2}\sum_{n=1}^{N} \mathbf{e}_n(\mathbf{x})^T \mathbf{W}_n \mathbf{e}_n(\mathbf{x}) \tag{32}$$

where $e_n(\cdot)$ is an error term that is weighted by W_n and would consist, in this case, of the camera and IMU error. The only way to minimize this error is through observations, which are also presumed to have an associated noise term.

$$\mathbf{z}_n = g_n(\mathbf{x}) + \mathbf{v}_n, \tag{33}$$

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

$$\mathbf{e}_n(\mathbf{x}) = z_n - g_n(\mathbf{x}) \tag{34}$$

$$error = observation - predicted\ \ observation \tag{35}$$

Since observation noise between the IMU and camera v_n are independent, the covariance matrix is diagonal. Hence $Wn = E[\hat{\mathbf{e}_n}\mathbf{e}_n^T]^{-1} = Q^{-1}$. Least squares aim to find the state vector x that minimizes the error function. Hence, the optimization problem is posed as follows:

$$\mathbf{x} = argmin(J(\mathbf{x})) \tag{36}$$

If the error term $e_n(x)$ was linear w.r.t x the solution would be as simple as setting the $\delta J(x)^T/\delta x$ to zero and solving the respective simultaneous equations. However, this is often not the case, so the equation's solution 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 solving for cyclist state from respective error functions. For further information, the reader is directed towards the Okvis paper [26], and other reports of a more in-depth nature in this field.

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 to understand the relevant object environment.

Kernelised Correlation Filter

In the recent paper by Henriques et al. [31] 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:

$$P = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix} \tag{37}$$

Therefore the product of $Px = [x_n, x_1, x_2, ..., x_{n-1}]$ and any shift can be modelled using Pux. Because of its cyclic nature, the signal x repeats when u = n, and so the full set of shifted signals is obtained with $P^u x|u = 0, ..., n - 1$. An example of this is shown in the 1D case by Figure 19.

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

$$X = C(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 & \dots & x_{n-1} & x_n \\ x_n & x_1 & \dots & x_{n-2} & x_{n-1} \\ x_{n-1} & x_n & \dots & x_{n-3} & x_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_2 & x_3 & \dots & x_n & x_1 \end{bmatrix} \tag{38}$$

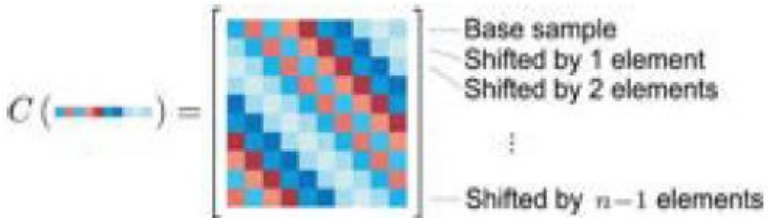


Figure 19. Illustration of circulant matrix [31].

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

$$F(\mathbf{z}) = \sqrt{n}F\mathbf{z} \tag{39}$$

$$\hat{\mathbf{x}} = F(\mathbf{x}) \tag{40}$$

$$X = Fdiag(\hat{\mathbf{x}})F^H \tag{41}$$

This is possible because the DFT is a linear operation.

Kernels

Now that the KCF has several 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 42 creates an n × n kernel matrix $K_{i,j} = k(x_i, x_j)$

$$k(\mathbf{x}, \mathbf{x}') = exp(\frac{1}{\sigma^2}||\mathbf{x} - \mathbf{x}'||^2) \tag{42}$$

and therefore, the function becomes:

$$f(\mathbf{z}) = \mathbf{w}^T\mathbf{z} = \sum_{i=1}^{n} \alpha_i k(\mathbf{z}, \mathbf{x}_i) \tag{43}$$

with the solution of Kernelised Ridge Regression given by:

$$\alpha = (K + \lambda I)^{-1}\mathbf{y} \tag{44}$$

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 diagonalization of Equation 44 to Equation 45.

$$\hat{\alpha} = \frac{\hat{\mathbf{y}}}{\hat{\mathbf{k}}^{\mathbf{xx}} + \lambda} \tag{45}$$

where $\hat{\mathbf{k}}^{\mathbf{xx}}$ is the first row of the kernel matrix $K = C(\hat{\mathbf{k}}^{\mathbf{xx}})$ and aˆdenotes the DFT of a vector. This means that only an n×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 × n sized kernel matrix.

Detection

The basic premise is that the α term in Equation 45 is now initialized to a learned set of coefficients that correctly describe the object. Therefore, the regression function f(z) must be evaluated at several candidate patches to detect the object of interest. Circulant shifts can also describe these patches. Defining the kernel matrix Kz 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 V-A1 allows only the first row of the kernel matrix to be relevant, and hence:

$$K^z = C(\mathbf{k^{xz}}) \tag{46}$$

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

$$f(z) = (K^z)^T \alpha \tag{47}$$

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 be implemented can be found in the relevant paper [31].

CONTRIBUTION

Software Solution

The system consists of several modules, each responsible for a specific aspect; see Figure 20 for more details. These will now be summarised below, and 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 environment representation, the cyclist model for path prediction, and 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.

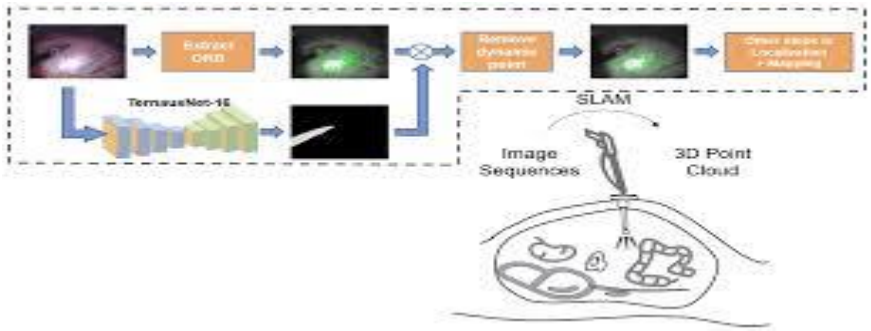


Figure 20. System design.

Tracking Module

The tracking module takes the segmented image masks from Mask-RCNN, the passed-on depth image, and Okvis pose output to determine a detected object's location. 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 key points, 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 an intersection between the cyclist and objects in the future. The idea is to notify both the object (possibly a person or car), and the cyclist that they are about to collide. Hopefully, both parties can take preventive measures to avoid each other once aware of the unfolding situation.

Limitations

Sadly, the system 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.

DEPTH MEASUREMENT OUTLIER REMOVAL

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. Several points are not detected, so large quantities of the depth map are 0. Therefore, the median depth is selected only from coordinates at which *depth* > 0. Figure 21 shows examples of a depth camera image taken in the Imperial Library.

As seen from Figure 21 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. When the depth measurements are often incorrect, an infeasibly large value is returned > 50*m*. This is a real problem for tracking objects, not necessarily after successful tracking, as the value can be discarded as an outlier but on initialization or after only a few data points. The depth is incorrect, but that problem is propagated in all directions when transformed into the world frame. An example of this phenomenon is displayed in Figure 22, where a bottle is tracked from frame to frame, which also demonstrates the ability of the tracking method to be non- cycling domain-specific.

This can occur in several instances, and far away objects are most prone as little area is 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, the latter object must be larger than the first. Hence, using the camera to the world model, if the depth is incorrectly large, the object's height in the world frame will be far larger than is known to be possible, i.e., a 10m tall human.

Given that human sizes roughly vary from a 4ft child to a 7ft adult, poor depth predictions convert people 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 1.5*m*. This logic can be extended to other objects like cars and bikes. Therefore, assuming a bounding box of [x_{tl}, y_{tl}, x_{br}, y_{br}], where *tl* denotes the top-left corner and *br* is the bottom right, the matrix P can be formed.

$$\mathbf{P} = \begin{bmatrix} x_{tl} & x_{tl} \\ y_{tl} & y_{br} \end{bmatrix} \tag{48}$$

Figure 21. Depth camera output.

Completing the initial steps of Cam2World (Section IV-B) until Equation 20, and adding the 1 required for the proceeding transform, the process yields:

$$\mathbf{K} = \begin{bmatrix} k_{(1,1)}d & k_{(1,2)}d \\ k_{(2,1)}d & k_{(2,2)}d \\ d & d \\ 1 & 1 \end{bmatrix} \tag{49}$$

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(x′) from the depth image. For the next steps, the transformation matrix T_{WC} calculates the world position.

$$\mathbf{T_{WC}} = \begin{bmatrix} \dots r_1 \dots \\ \dots r_2 \dots \\ \dots r_3 \dots \\ \dots r_4 \dots \end{bmatrix} \tag{50}$$

where r_i denotes row i of T_{WC}, and r4 = [03×1, 1]. Therefore, replicating a simple matrix multiplication in Equation 51, the row r_3 is the only row from T_{WC} that is needed for the object height.

$$\mathbf{WorldPoints} = \begin{bmatrix} x_{w,tl} & x_{w,bl} \\ y_{w,tl} & y_{w,bl} \\ z_{w,tl} & z_{w,bl} \\ - & - \end{bmatrix} = \mathbf{T}_{WC}\mathbf{K} = \begin{bmatrix} \dots r_1 \dots \\ \dots r_2 \dots \\ \dots r_3 \dots \\ \dots r_4 \dots \end{bmatrix} \tag{51}$$

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

Figure 22. 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.

$$z_{w,tl} - z_{w,bl} = r_3\mathbf{K}[:,1] - r_3\mathbf{K}[:,2] = r_3(\mathbf{K}[:,1] - \mathbf{K}[:,2]) \tag{52}$$

where K[:, 1] defines the first column of K, and defining vector A as:

$$Ad = \mathbf{K}[:,1] - \mathbf{K}[:,2] = \begin{bmatrix} k_{(1,1)}d - k_{(1,2)}d \\ k_{(2,1)}d - k_{(2,2)}d \\ 0 \\ 0 \end{bmatrix} = d \begin{bmatrix} k_{(1,1)} - k_{(1,2)} \\ k_{(2,1)} - k_{(2,2)} \\ 0 \\ 0 \end{bmatrix} \tag{53}$$

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

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

TRACKING MODULE

RCNN generates unordered bounding boxes and masks for each frame, requiring a method for seamless object tracking across consecutive frames. This is a hard challenge due to object occlusion, variation in shadow/lighting conditions, and rapid in-frame movement. An obvious method is the intersection over union (IOU) of the bounding boxes on a frameby-frame basis. This is good in theory. However, it cannot compensate for rapid movement (of the object and the camera angle), occlusion, and object overlapping. Therefore, a set of methods has been developed to overcome this problem, providing a robust tracking system.

The method must track objects in the frame, determine which objects are no longer in the frame and find any new objects that enter the frame. Information about each object must be stored to determine the velocity and direction of travel/ if a collision is likely between the cyclist and the object in question. Figure 23 displays an example of a working system.

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 V-A. 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 IX-B, IX-C, and IX-D. 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-RCNN 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 co- variance 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 combined with the other tracking modules in Section IX-F.

BRISK - Selector

As described earlier in Section-IV-G is a binary descriptor that can match key points across two photographs. This can also be used in the tracking section to match key points across RoIs; although not thoroughly accurate, it offers a method of matching RoIs regardless of location. This should help redetect objects after periods of occlusion, invariant of distance, which other methods cannot do. The process to match the RoIs is simple. First, key points 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 RoWe determines how well they match.

Class Matching

An obvious additional requirement is the matching of classes between frames, as a person's RoWe should not match the RoWe of a car in the next frame. Hence, the class is stored for access by the next frame for each RoI. This can reduce the expensive computation time of other metrics but provide an easy RoWe comparison tool. It can be noted that although Mask RCNN is not perfect and often miss-classifies objects, it is reliable enough to be assumed as correct.

Kalman Filter RoWe Matching

A Kalman filter, developed by Rudolf E. Kalman, has two main parts: a predictive step and its update, which uses measurements of the object's 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 RoWe on the input image. It was quickly discovered that the model was not sufficiently robust as it did not consider 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 was far more successful.

Kalman Predictive Step: Assuming a linear system with a Gaussian process, the state can be modeled as:

$$\hat{X}_k = F_k \hat{X}_{k-1} + w_{k-1} \tag{55}$$

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

$$\hat{X}_k = [Cx, Cy, Cz, w, h, Cx_{vel}, Cy_{vel}, Cz_{vel}, w_{vel}, h_{vel}] \tag{56}$$

where (*Cx*,*Cy*) is the center point of the bounding box of the object,Cz is the median distance of the masked RoI, w the width, and h the height. Cxvel describes the centre point velocity in the x direction, and Cy_{vel}, Cz_{vel}, w_{vel}, hvel 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:

$$\begin{aligned} \gamma_{k+1} &= \gamma_k + \gamma_{vel} \times \Delta t \\ \gamma_{vel,k+1} &= \gamma_{vel,k} \end{aligned} \tag{57}$$

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

$$F_k = \begin{bmatrix} I_5 & \Delta t \times I_5 \\ 0_{5\times5} & I_5 \end{bmatrix} \tag{58}$$

The final term w_k = 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 human would walk (in m) from one frame to the next, given an average walk of 1.5m/s. These parameters disth for a human, $dist_b$ for other cyclists and $dist_v$ for a motor vehicle are listed below:

$$\begin{aligned} dist_h &= 1.5m/s \\ dist_b &= 10m/s \\ dist_v &= 15m/s \end{aligned} \tag{59}$$

The variance in the process noise would also be expected to increase depending on the type of object concerned, and thus, they are initialized 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 are 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 initialize the Kalman filter.

$$\begin{aligned} v_h &= [1.5, 1.5, 1.5, 7, 7, 0.15, 0.15, 0.15, 0.7, 0.7] \\ v_b &= [10, 10, 10, 7, 7, 1, 1, 1, 0.7, 0.7] \\ v_v &= [15, 15, 15, 7, 7, 1.5, 1.5, 1.5, 0.7, 0.7] \\ Q_k &= diag(v) \end{aligned} \tag{60}$$

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:

$$Z_k = H_k X_k + v_k \tag{61}$$

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

$$H_k = \begin{bmatrix} I_5 & 0_{5\times5} \end{bmatrix} \tag{62}$$

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

$$\begin{aligned} r &= [0.2, 0.2, 0.2, 20, 20] \\ R_k &= diag(v) \end{aligned} \tag{63}$$

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 object type and respective presumed velocities.

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

$$\begin{cases} \hat{X}_k(-) = F_k \hat{X}_{k-1}(+) \\ P_k(-) = F_k P_{k-1}(+) F_k^T + Q_k \end{cases} \tag{64}$$

where $\hat{X}_k(-)$ is a priorWe state estimate and $P_k(-)$ is the priorWe estimate error at step k. It is essentially the previous state × 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:

$$\begin{cases} \tilde{y}_k = Z_k - H_k \hat{X}_k(-) \\ S_k = R_k + H_k P_k(-) H_k^T \\ K_k = P_k(-) H_k^T S_k^{-1} \\ \hat{X}_k(+) = \hat{X}_k(-) + K_k \tilde{y}_k \\ P_k(+) = (I - K_k H_k) P_k(-) (I - K_k H_k)^T + K_k R_k K_k^T \\ \tilde{y}_{k|k} = Z_k - H_k \hat{X}_k(+) \end{cases} \tag{65}$$

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

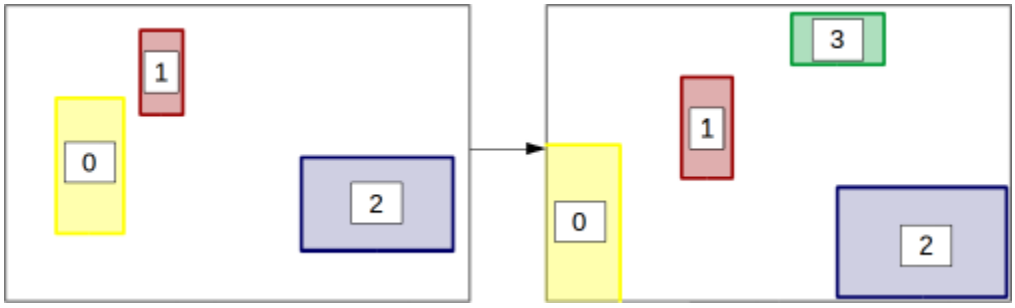


Figure 23. RoWe Old to new frame example.

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 RoWe 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 $\tilde{X}_k(-)$. Skipping the formulation, the log-marginal likelihood is:

$$l = log\ p(\mathbf{z}) \tag{66}$$

$$\begin{cases} l^{(-1)} = 0 \\ l^{(k)} = l^{(k-1)} - \frac{1}{2}(\tilde{y}_k^T S_k^{-1} \tilde{y}_k + \log|S_k| + d_y \log 2\pi) \end{cases} \tag{67}$$

where d_y is the dimension of the measurement matrix. The part dependent on the new RoWe is $\tilde{y}_k^T S_k^{-1} \tilde{y}_k$ as S_k, from Equation 65 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:

$$M = \tilde{y}_k^T S_k^{-1} \tilde{y}_k \tag{68}$$

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

$$Match\ Metric = \begin{cases} 0 & if\ K_{threshold} - M \leq 0 \\ K_{threshold} - M & otherwise \end{cases} \tag{69}$$

RoWe Lives

Each region of interest may be tracked from one frame to the next. However, if that object is occluded, it may not be spotted in the next frame. This could lead to information loss regarding that object's state unless accounted for. Therefore, a life system is in place whereby an object has several 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.

Combining Matching Methods

For each ROI_{old} and each ROI_{new}, the match metrics for each method (KCF,BRISK,Kalman) are calculated. This forms a matrix matches $\in R^{new \times old \times methods}$ 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 completing the matching process, several extra steps must occur before continuing to the next frame. For example, any old RoIs 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 RoWe
        objects

// Update newROIs to include undetected RoIs
   from old frame.
foreach old ∉ old_matched do     /* For every old
 roWe not found in new frame */
    older = old_frame[old]        /* Get olderROWe
     object */
    if older.lives > 1 then     /* If the RoWe 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 RoWe 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 newROWe objects

```
// Add older RoWe details to matched new ROoI
foreach old, new ∈ old_matched, new_matched do
 /* For every matched old/new RoWe pair */
    older = oldROIs[old]  /* Get olderROWe object
     */
    newer = newROIs[new]      /* Get newerROWe
     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
     RoWe with older RoWe information */
     /* (Kalman Filter, lives, colour, id, model) */
end
```

Finally, any new RoIs not in the old frame must be assigned an individual ID to be recognized over the next frames, Algorithm 6.

// Note that if there are no objects in old frame
new_frame.id = zeros
maxId = new frame.id.max() + 1 ; /* Get current

Algorithm 6: Algorithm to create new IDs for the 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 newROWe objects

```
maximum ID + 1 */
 foreach new ∉ new_matched do          /* For every
not-matched new RoI */
    newer = new_frame[new] ;      /* Get newerROWe
   object */
    newer.id = maxId ;          /* Assign new ID */
    maxId += 1 ;      /* Increment the maximum ID */
 end
```

MOVEMENT PREDICTION

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

Datasets

The great advantage of the system is that it already has the core components required to create unlimited training data for movement prediction. All required is to cycle around, record the images, and pose with Okvis. This can then be passed into the model for training. One caveat is that due to the poor depth camera readings outside, Okvis sometimes struggles to maintain the correct position and can drastically miss-predict the bicycle's speed. This means that several 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.

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 24. The main problem with models is the ability to generalize; hence, training on a limited, unbalanced dataset is tricky. So, for the prediction, it is assumed that someone can record their daily commute and train the model on that route. So, the validation and test sets consist of the same route on a different day with the obvious varied lighting, traffic conditions, and traffic lights that come with the average daily commute variation.

Objects Prediction

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

$$F_{k+n\_seconds} = \begin{bmatrix} I_5 & n\_seconds \times I_5 \\ 0_{5\times5} & I_5 \end{bmatrix} \tag{70}$$

where n_seconds is the number of seconds in the future to predict the position at.

$$\hat{X}_{k+n\_seconds}(-) = F_{k+n\_seconds}\hat{X}_{k-1}(+) \tag{71}$$

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 so it is easy enough just to repeat Equation 72 (n_seconds/Δt) times.

$$P_{k+\Delta t}(-) = F_k P_{k-\Delta t}(+) F_k^T + Q_k \tag{72}$$

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

$$radius = \sqrt{P_{k+n\_seconds}(-)} \tag{73}$$

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.

Bike Model

The bike is deemed the most important model, and therefore, predictions should consider 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 for predictions using image inputs.

Okvis Sequence Inputs

A recurrent neural network is an obvious choice since the model must learn a sequence. 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 generalize 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 learns the difference in camera frame coordinates from the 0th frame in the sequence. However, things such as orientation can make a large difference, if the bike is at a big angle in the world frame, it will turn 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.

$$X_{frame_n} = \left[[I_{1\times3} \quad 0] T_{WC_0}^{-1} T_{WC_n} \quad , \quad C_{ez} \right] \tag{74}$$

$$C_{ez} = (T_{WS}T_{SC})^{-1} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix}^T \tag{75}$$

$$y = r_{WC_{n+1}} \tag{76}$$

The model comprises the layers on the right of Figure 25. Given a limited input space X = ($len_{sequence}$, 16), 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 y = [x, y, z], representing the displacement from the 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 a mean squared error and is formulated as follows:

$$mse = \sum_{i=0}^{n}\sum_{j=0}^{m}(\mathrm{y}_{i,j} - \hat{\mathrm{y}}_{i,j})^2 \tag{77}$$

where i is the sequence number, j is the variable prediction and therefore m = 3 for [x, y, z]. The model is 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:

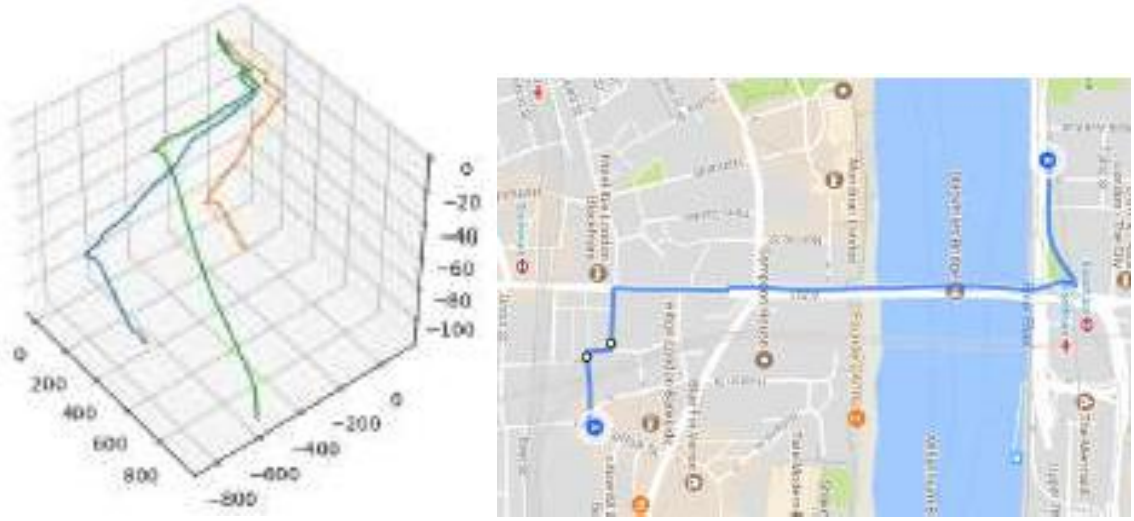


Figure 24. Okvis Paths: Left, Google Map: Right.

Figure 25. 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 III-B. The loss function for the RCNN model is the same as Equation 77, as this is a regression problem once again. The images are fed into the network, and the difference is predicted much like before. The only variation is the data input and the flattening layer before the LSTM. A convolutional layer outputs several 3D tensors (w, h, d), which is incompatible with the LSTM or dense matrices.

Hence, a flattening layer converts the tensors into a shape (sequence length,w×h×d) for input into the final network layers. Additionally, extra pre-processing is required; since images are scaled from 0 to 255, it is hard to learn. Hence, the images are scaled to (−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 for the model to fit into GPU memory. This is particularly prevalent when classifying a sequence.

Optimising Learning Rate and Decay

Adam optimizer was used for the model, it is a common solver for deep learning problems that requires several parameters, which are all set as the default recommended from the paper [32] $\beta 1 = 0.9$, $\beta 2 = 0.999$, $\epsilon = 10{-}8$ 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 is required to select the right values. A grid search method was chosen, and loss characteristics at epoch 200 were compared to get a reasonable approximation for the correct learning rate. The image and pose model prefers higher learning rates around the 10^{-4}, 10^{-3} region. Learning rate decay has little influence on the results of the images network, which has an optimal value of:

From experience and logic, taking an optimal value with decay for longer network runs is preferable. As the optimization is performed over only 200 epochs often, the lower learning rates with decay perform better in the long run. Hence, the final model uses the parameter choice ranked no.2 in Table 3. A part to note is that the loss function is a mean squared error, so 0.11 loss is roughly equivalent to 0.3m. Interestingly, the pose model has a higher optimal loss than the images model, likely due to the number of epochs and model complexity in using the same route for the test set (different days but same route). Therefore, the pose model is simple and can train much faster than the image model, so it should be trained for longer for a fair comparison. The image model has more layers and thus may over-fit the route whilst the pose model may perform better on unknown routes. The highest ranked parameters in Table 4 are taken forward to the next stage of the process, optimizing the layer dimensions.

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

Pose Model: As the pose model was once again optimized with a small number of epochs, it has a slightly higher loss than the model of the image. The results from optimizing the LSTM layers are as follows:

The second best and best results are narrowly different, so the second best and simpler model from Table 4 was then taken forward as the final dimensions for the network and is used in the complete model.

Images Model: The image model is optimized 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 25). Optimizing 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 optimized.

Interestingly, the image 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 33.

Final Optimised Models:

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. However, the final layers of these networks 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 27; it has two input tensors of the images plus pose data and three output variables, the predicted [x, y, z].

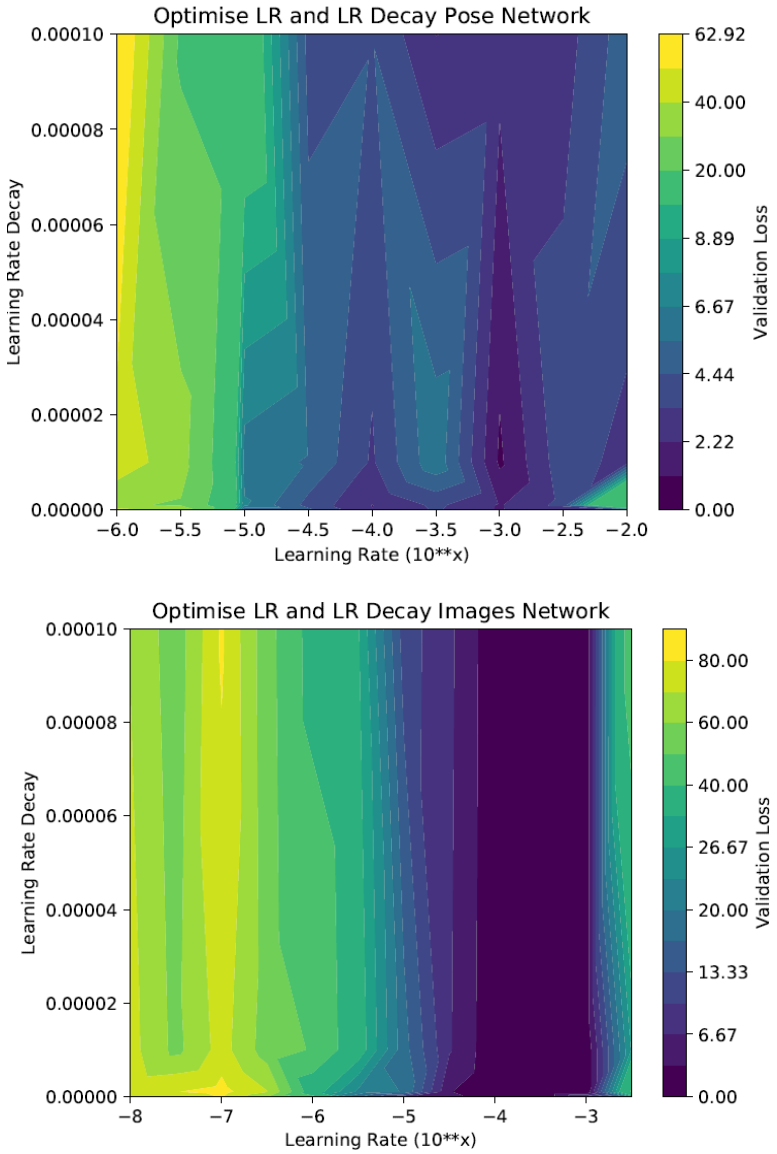


Figure 26. Pose Lr and Lr Decay Grid: Left, Images Network Optimisation: Right.

Training Regime

The network is large, 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 25. 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 finetune the weights. For the final training, the frame gap was increased to 1s; 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.

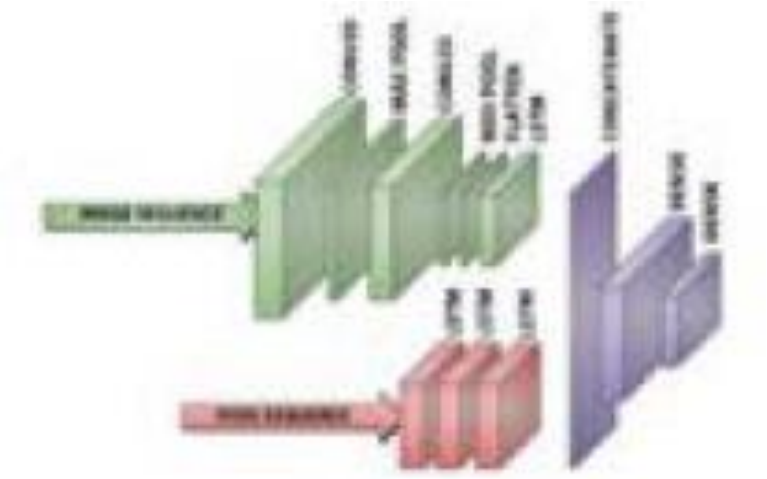


Figure 27. Complete network architecture.

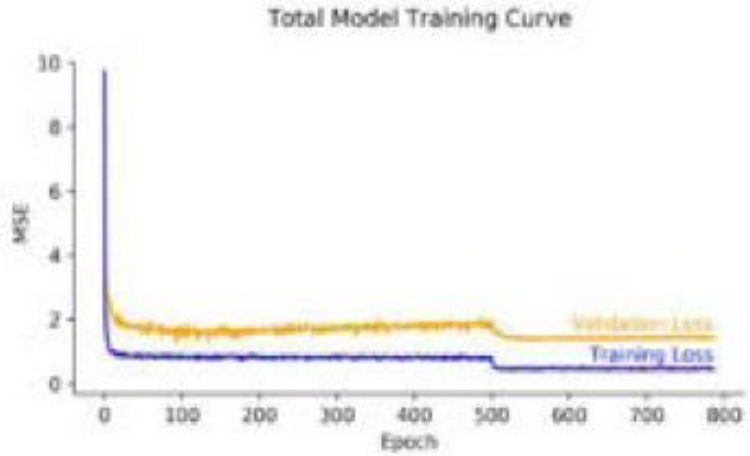


Figure 28. Complete network training.

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 prediction time frame. In this case, a timeframe of c.1-2s is reasonable as the Mask-RCNN model cannot detect small objects further away with adequate accuracy, and a bike can move at around 10m/s. There is a narrow advantage to using the proposed architecture at this time frame, so the decision would be hardware-specific. This is due to the 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, joining the image-segmentation and prediction networks may be beneficial. 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.

Collision Detection Module

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

$$dist = \sqrt{((x_c - x_o)^2 + (y_c - y_o)^2} \tag{78}$$

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

$$\begin{cases} dist < R_{bike} + R_{obj} & \textbf{Check IOU} \\ dist > R_{bike} + R_{obi} & \textbf{Continue} \end{cases} \tag{79}$$

$$R_k = \sqrt{var_k} \tag{80}$$

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 the union of the two circles.

Points of intersection: The first step is to calculate where the circles overlap. This is detailed in Figure 29. An intersection point can form a triangle in each circle between the y-axis, x-axis and a line of length Ro, Rc passing through the circle's origin.

$$\begin{aligned} x^2 + y^2 &= A_r^2 \\ (d^2 - x^2) + y^2 &= B_r^2 \end{aligned} \tag{81}$$

Solving for x and y yields:

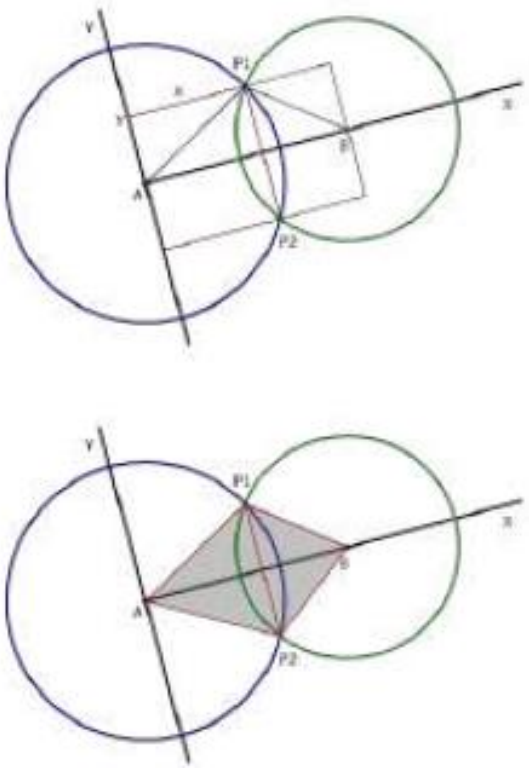


Figure 29. Intersecting Points on Circle [33], Right: Area of Two Intersecting Circles [34].

$$x = \frac{A_r^2 - B_r^2 + d^2}{2d}$$
$$y_{1,2} = \pm\sqrt{A_r^2 - x^2} \tag{82}$$

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

$$\overrightarrow{e_1} = \frac{\overrightarrow{B} - \overrightarrow{A}}{|\overrightarrow{B} - \overrightarrow{A}|} = \frac{1}{d}\begin{bmatrix} B_x - A_x \\ B_y - A_y \end{bmatrix} \tag{83}$$

$$\overrightarrow{e_2} = rot(90)\overrightarrow{e_1} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\overrightarrow{e_1} \tag{84}$$

$$\overrightarrow{P_{1,2}} = \overrightarrow{A} + x \cdot \overrightarrow{e_1} \pm y \cdot \overrightarrow{e_2} \tag{85}$$

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.

$$dist < |B_r - A_r| = \textbf{Circle Inception} \tag{86}$$

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

$$\theta_A = 2sin^{-1}\left(\frac{y}{A_r}\right)$$
$$\theta_B = 2sin^{-1}\left(\frac{y}{B_r}\right) \tag{87}$$

From this the area of the two triangles are:

$$area_{t,1} = y\sqrt{A_r^2 - y^2}$$
$$area_{t,2} = y\sqrt{B_r^2 - y^2} \tag{88}$$

Finally, the area of the intersection is calculated using:

$$area = A_r^2 sin^{-1}(\frac{y}{A_r}) + B_r^2 sin^{-1}(\frac{y}{B_r}) + y\left(\sqrt{A_r^2 - y^2} + \sqrt{B_r^2 - y^2}\right) \tag{89}$$

Intersection Over Union: To better understand the likelihood, the circle approximation is used. Although not exact as Gaussians are a bell shape, this 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 probable crash. It can be calculated from previous results:

$$IOU = \frac{area_{intersection}}{area_{c1} + area_{c2} - area_{intersection}} \tag{90}$$

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 object's velocity is predicted, informing the collision detection module if the object in question is moving.

EXPERIMENTAL RESULTS AND SYSTEM EVALUATION

Mask-RCNN

There have been several interesting outcomes in applying Mask-RCNN to the specific urban environment. Overall, the system is robust, often detecting images well, so there are rarely gaps when an object is detected in one frame and not the next. Several incorrect classifications often provide a worry; for example, the classification of road paint can cause a problem for the system (Figure 30). This often happens with road markings of bicycles that are relatively convincing examples; this is perhaps solvable by using these examples as part of the null class during training and could be investigated in future work.

Figure 30. Mask-RCNN failure.

Cyclist Model

The cyclist model is effective, as proven in Section X-C; however, it currently requires a new model with respective inputs and convolutional layers. This is slow and memory intensive; the best solution would be to integrate the model into Mask-RCNN and share some of the convolutional layers.

However, it is a significant modification as Mask-RCNN does not use data sequences, so a novel architecture would need to be investigated.

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

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. However, the extent of the problem is often drastic and can cause Okvis to perform poor pose estimations and hence be unable to determine the bicycle location. This problem could be eliminated using a stereo camera system or integrating and relying on GPs in bright lighting conditions.

Crash Sequence Evaluation

To determine the system's effectiveness, it is important to know that it will produce numerous false positives and irritate users and 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, reducing the danger in experimenting. The other sequence contains a parked car, which is detected but also determined not to be moving, hence of no danger to the rider.

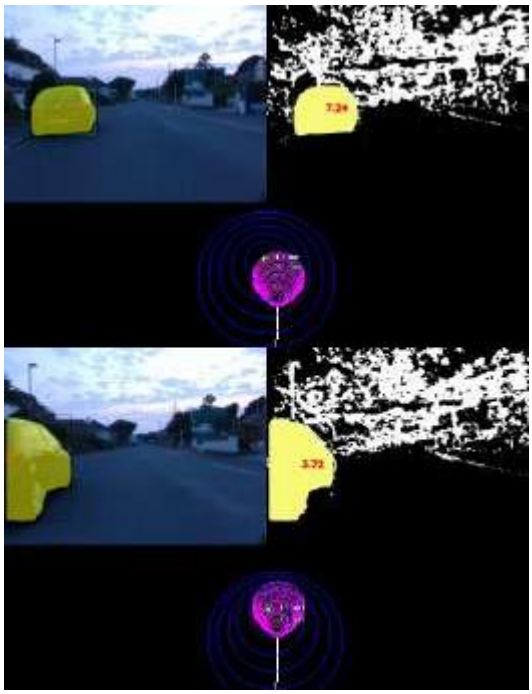


Figure 31. Parked car evaluation.

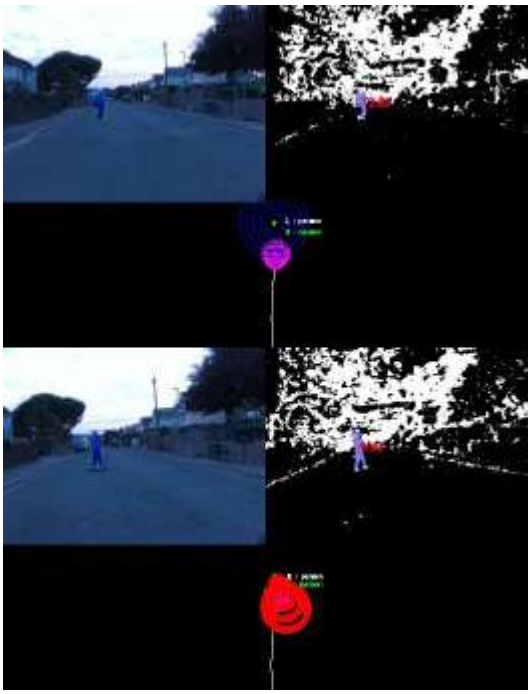


Figure 32. Left: Pedestrian First Detected, Right: System detects potential collision.

It is hard to get a sequence into the report, but a comparison to the staged crash shows that the car was detected but not deemed a threat to the cyclist. A common problem with the system is depth perception - this can often cause some objects to appear to move slightly as the depth measurements past 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 center is the predicted position, and the radius is 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 X-D. It was simply because it is hard to find a willing volunteer to be cycled at enough speed and proximity to warrant a warning, so slower speeds and higher sensitivity were used 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 (Figures 31 and 32).

CONCLUSION

Speed

It is evident that this product is much needed in the arsenal of a cycling commuter, but it cannot be implemented in its current state. 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 to increase accuracy to the current level. Also, a pure implementation in a faster language such as *c* + + could help. 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.

Object Detection

At the time of writing, the only real-time object detection method that could apply to cycling is tiny-yolo [10]. This bounding box approach 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 VIII.

Although not perfect, combined with a 3D appreciation of the environment, it could be effective and, most importantly, fast enough for use.

SLAM

Although Okvis is effective, the poor quality depth cam- era images render it less reliable outside than in indoor environments. This would typically not be acceptable on a final system, so perhaps incorporating GPS measurements and Okvis would be beneficial. Okvis is great for areas without GPS, such as tunnels, where lighting is also reduced. Okvis can run from a single camera, so removing 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.

CONTRIBUTIONS

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

Section V-A1 discusses how to modify the KCF tracker (Section V-A) for use in the multi-class case. This is later used with Brisk, class, and Kalman matching for object identification between each frame Section IX-F.

The final contributions of this report involve the tracking and prediction models for both objects (Section X-B) and cyclists (Section X-C). The object model modified 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 the 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 methods required to integrate them into the final system. It is not a completed system but shall continue to be developed after this report is submitted. The project's goals 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 in real-time (even on dedicated hardware). However, the system is designed modularly so that when advances in respective fields occur, it can be modified accordingly.

ETHICS

Modification for Military Purposes

Although strictly for civilian applications, the code presented in this report has 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 and would require significant modification for use in such an environment 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 in real-time and could be a potential source of misuse.

Other Ethics Issues

The code developed is intended to offer a basis for an early warning system for cyclists in urban environments. Hence, some 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 a lack of attention and potential accidents when the system cannot detect a

collision. That being said, it is evident that there is a need for such a system for cyclists, so the benefits would clearly outweigh the disadvantages in such a case (Figure 33).

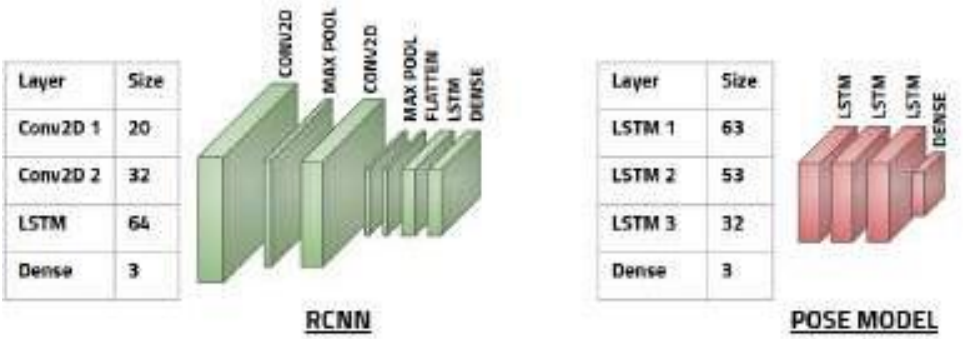


Figure 33. Final Images Model: Left, Pose Model: Right.

REFERENCES

1. D. F. Transport, "Table ras30001: Reported road casualties by road user type and severity, great britain, 2016," Dft, Report, 2016. pages 1
2. "Ras50001: Contributory factors in reported accidents by severity, great britain, 2016," Dft, Report, 2016. pages 1
3. "Focus on cycling in 'reported road casualties great britain 2013," Dft, Report, 2013. pages 1
4. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," vol. 25, 01 2012. pages 2
5. A. Karpathy. (2017) Networks. [Online]. Available:http://cs231n. github.io/convolutional-networks/pages 2
6. Li, H., Guo, X., Ouyang, B. D., & Wang, X. (2018). Neural network encapsulation. In Proceedings of the European conference on computer vision (ECCV) (pp. 252-267).
7. Paik, I., Kwak, T., & Kim, I. (2019, October). Capsule networks need an improved routing algorithm. In Asian Conference on Machine Learning (pp. 489-502). PMLR.
8. S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *CoRR*, vol. abs/1710.09829, 2017. [Online]. Available: http://arxiv.org/abs/1710.09829pages 2
9. 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. [Online]. Available: http://arxiv.org/abs/1311.2524pages 2, 3
10. 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. [Online]. Available:http://arxiv.org/abs/1506. 02640pages 2, 23
11. 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 3
12. P. F. "Felzenszwalb and D. P. Huttenlocher, ""efficient graph-based image segmentation"," *"International Journal of Computer Vision"*, "2004". pages 3
13. R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available:http://arxiv.org/abs/1504.08083pages 3, 4
14. 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. [Online]. Available:http://arxiv.org/abs/1406.4729 pages 3
15. 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. [Online]. Available:http://arxiv.org/abs/1506. 01497pages 3, 5
16. J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014. [Online]. Available: http://arxiv.org/abs/1411.4038pages 4, 5

17. 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. [Online]. Available:http://arxiv.org/abs/1604.01685pages 4
18. 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*, ser. ICML'13. JMLR.org, 2013, pp. III– 1058–III–1066. [Online]. Available:http://dl.acm.org/citation.cfm?id= 3042817.3043055pages 4
19. 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. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.htmlpages 5
20. C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. pages 5
21. K. He, G. Gkioxari, P. Doll a´r, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870pages 5
22. C. Olah, "Understanding lstms blog." [Online]. Available:http://colah.github.io/posts/2015-08-Understanding-LSTMs/pages 5, 6
23. M. Deisenroth, "Gaussian processes and bayesian optimisation," 2018.pages 7
24. N. Srinivas, A. Krause, S. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," 2010. pages 6
25. O. docs. (2012) Images. [Online]. Available:https://docs.opencv.org/2.4/modules/calib3d/doc/camera calibration and 3d reconstruction.html pages 8
26. S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimiza- tion," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015. pages 9, 10, 11
27. H. F. Durrant-Whyte, "Uncertain geometry in robotics," vol. 4, pp. 23– 31, 03 1988. pages 10
28. T. Bailey and H. Durrant-Whyte, "Simultaneous localization and map- ping (slam): Part i," 10 2006. pages 10
29. S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *2011 International Conference on Computer Vision*, Nov 2011, pp. 2548–2555. pages 10, 11
30. O. docs. (2012) Images. [Online]. Available:http://opencv-python-tutroals.readthedocs.io/en/latest/py tutorials/ py feature2d/py features harris/py features harris.htmlpages 11
31. 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, no. 3, pp. 583–596,March 2015. pages 11, 12
32. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980pages 19
33. R. Eisele. (2016) Circle intersection. [Online]. Available:https://www. xarg.org/2016/07/calculate-the-intersection-points-of-two-circles/pages 21
34. (2016) Circle intersection. [Online]. Available:https://www.xarg. org/2016/07/calculate-the-intersection-area-of-two-circles/pages 21

APPENDIX

RoWe Matching Algorithm

// Multiply along the methods axis to produce one

Algorithm 7: Algorithm to match old and new frames

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 RoWe 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
   (oldROWe → newROI) for each
   method, oldROI, newROI
foreach oldROWe ∈ old_frame do    /* For every
 RoWe in last frame */
    foreach newROWe ∈ new_frame do    /* For
     every RoWe in current frame */
        foreach method ∈
         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, :, :]
```

```
 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
```

Figure 35. Parked Car Sequence.

XVIII. CRASH SEQUENCE

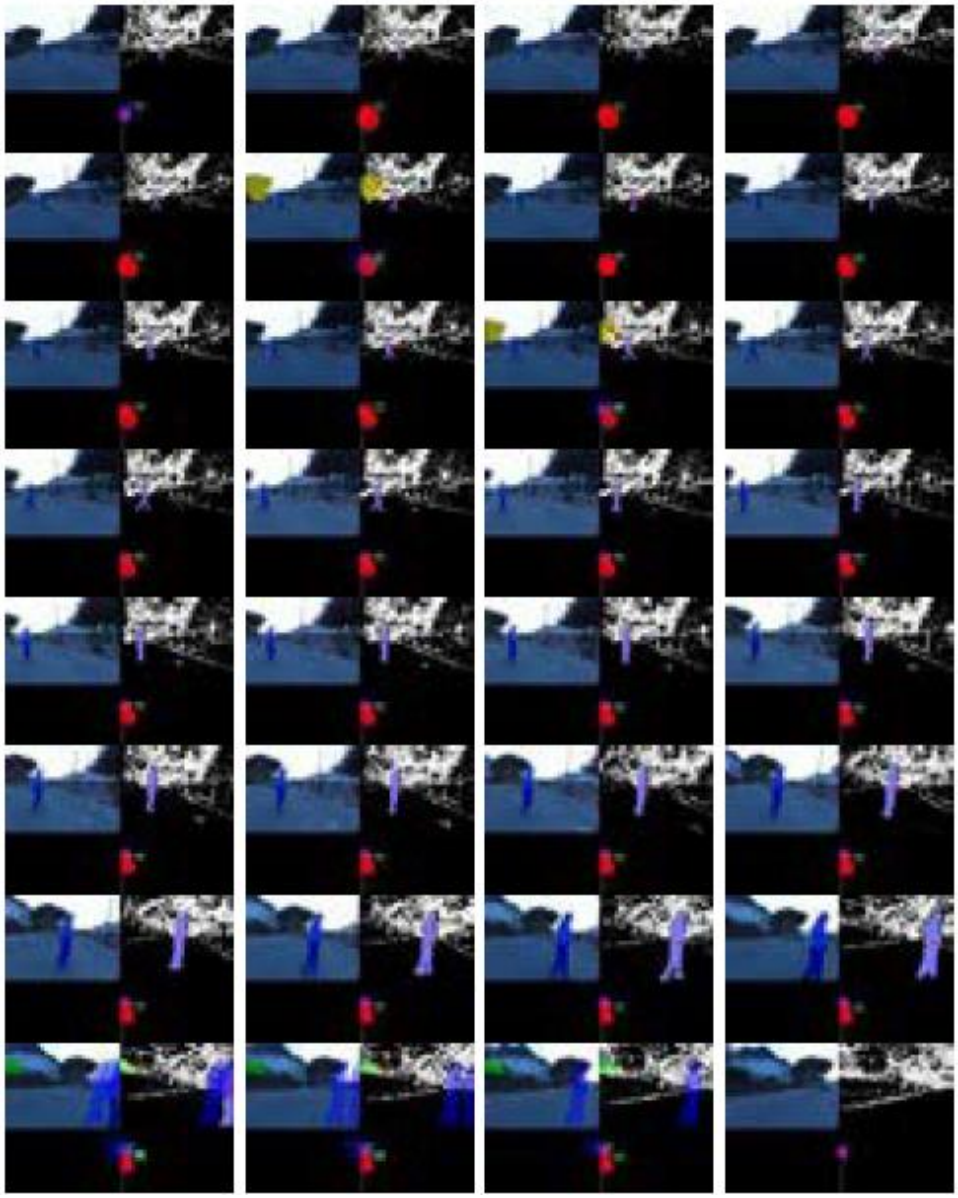


Figure 34. Staged crash sequence.

XIX. PARKED CAR SEQUENCE

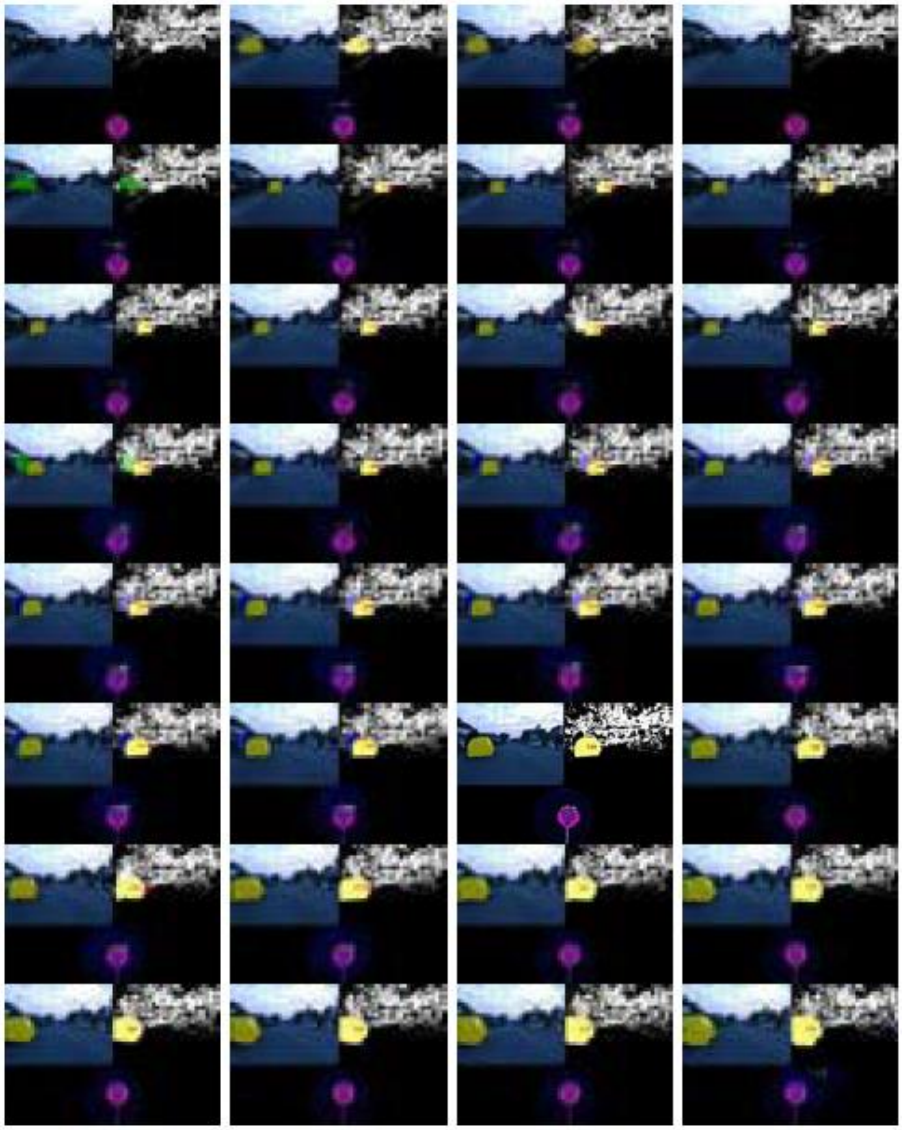


Figure 35. Parked Car Sequence.

Table 1. RCNN dimensions.

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 2. Pose model dimensions.

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

Table 3. Image learning rate vs learning rate decay.

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 4. Pose layer size Bayesian optimization.

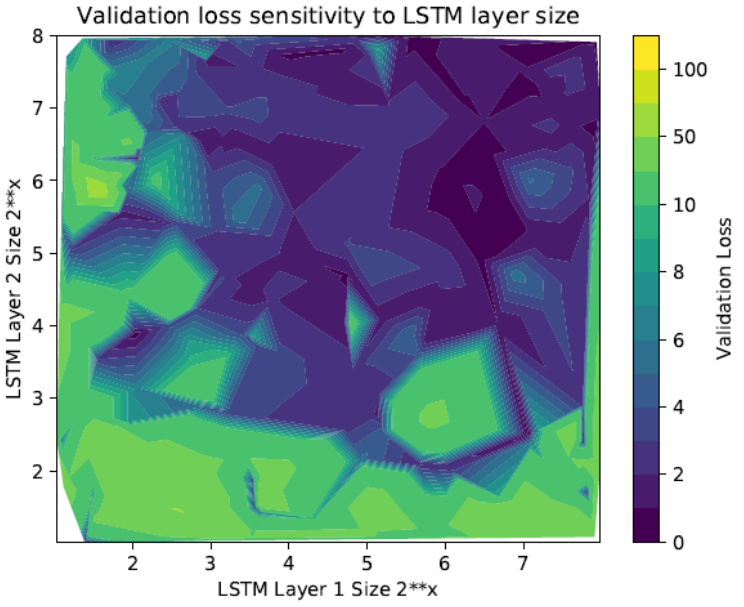
Rank	LSTM 1 Size	LSTM 2 Size	Validation Loss
1	26.44 = 87	26.25 = 77	0.42
2	26.04 = 63	$2^{5.7}$ = 53	0.43
3	22.03 = 4	$2^{3.9}$ = 15	0.48

Table 5. Images model Bayesian optimization.

Rank	Block 1 Size	Block 2 Size	Block 1 Layers	Block 2 Layers	Validation Loss
1	24.34 = 20	$2^{5.0}$ = 32	1	1	0.1
2	23.57 = 12	$2^{5.0}$ = 32	2	2	0.11
3	$2^{5.0}$ = 32	$2^{5.0}$ = 32	2	1	0.13

Table 6. Pose learning rate vs learning rate decay.

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



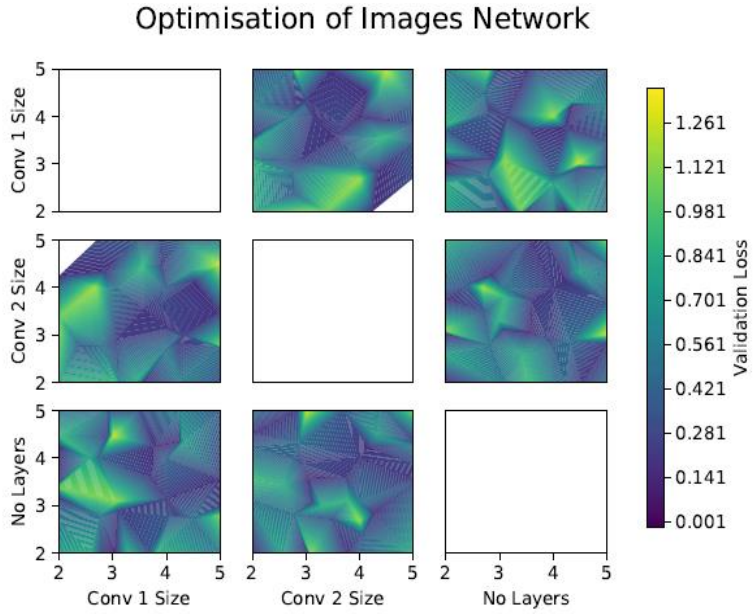


Figure 36. Pose Layer Optimisation: Left, Images Network Optimisation: Right.

XX. ETHICS CHECKLIST

Table ??

	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 Em- bryos/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?		✓

