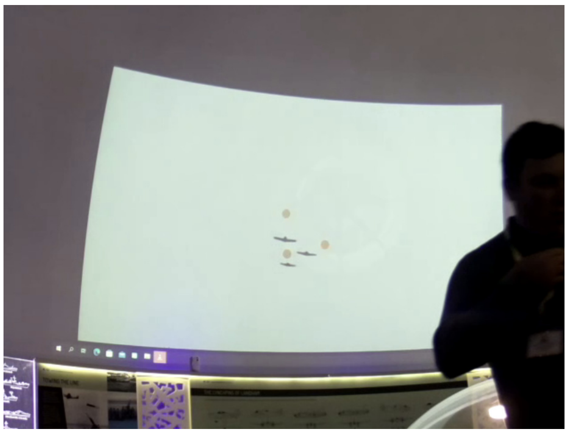
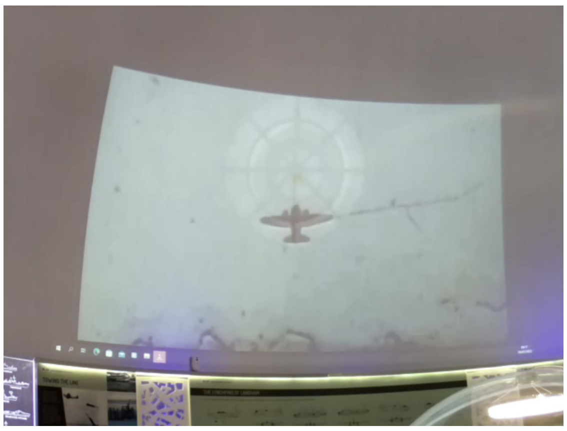
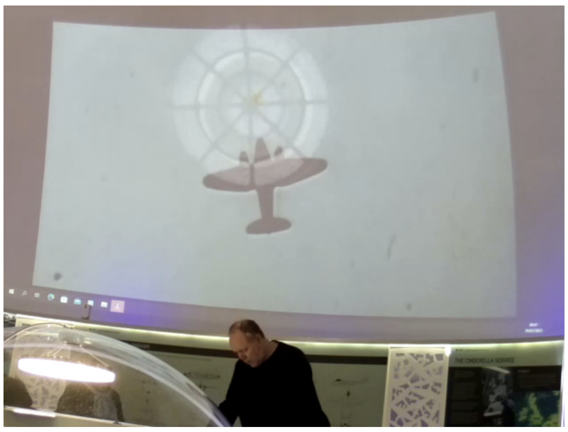
## 3.4. Example Images and Edge Cases


(a) OffLReticle 00009.png


(b) OnLReticle 000024.png


(c) OnLReticle 00007.png

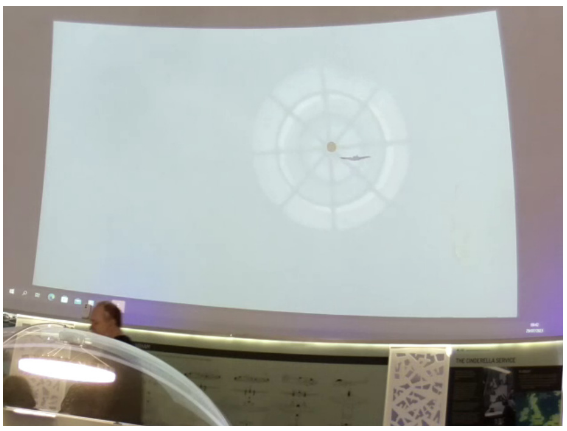

(d) OnRReticle 000020.png

Figure 4: Example stills from dataset

A range of stills is seen in Figure 4, with some having extremely visible features whilst others seem almost impossible to see. For instance, Figure 4a shows off very clear yellow dots, but a very hard to distinguish reticule, where Figure 4c has the opposite. This is due to the ambient lighting, with the lights being off in the former, making the flashlight appear much less distinctive against the projection but emphasising the contrast of the yellow dots slightly more. Figure 4b has lots of noise making it appear darker, whereas Figure 4d offers the clearest version of both elements. There's no difference in visibility from the varying angles.

## 4. Yellow Dot Detection

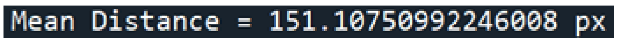
The first essential part of the system is the detection of the yellow dots themselves. They form a key part of the scoring in the game, with the requirement of being accurate and quick to detect. Consisting of two methods and a final comparison of the two, this section explains the work completed on yellow dot detection.

## 4.1. Evaluation Metrics

In order to see progress as code improves, metrics need to be created to determine the effectiveness of the yellow dot detection. The two concepts to measure were:

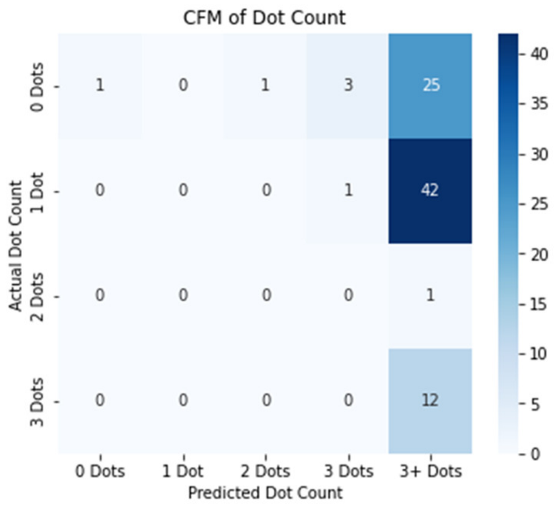
A) How accurate is the detection of the yellow dot?
B) Are the correct number of yellow dots detected?

Two separate metrics were created, see Figure 5, where one metric focused on distance, calculated as the Euclidean distance between the closest predicted dot and the actual dot, and one as a confusion matrix showing predicted vs actual counts of yellow dots. The Euclidean distances were averaged over all tested images, resulting in a final average minimum distance between the closest predicted and true yellow dots.



```
Mean Distance = 151.10750992246008 px
```

Average Distance

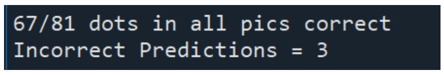


|  | 0 Dots | 1 Dot | 2 Dots | 3 Dots | 3+ Dots |
|---|---|---|---|---|---|
| 0 Dots | 1 | 0 | 1 | 3 | 25 |
| 1 Dot | 0 | 0 | 0 | 1 | 42 |
| 2 Dots | 0 | 0 | 0 | 0 | 1 |
| 3 Dots | 0 | 0 | 0 | 0 | 12 |

Confusion Matrix

Figure 5: Confusion matrix plot and average distance showing example performances

A further two metrics, see Figure 6, that show how many of the yellow dots were detected as a percentage, and one that shows a total of how many misclassifications occurred, were added later. The goal was to minimise misclassifications, maximise classification accuracy, and keep the mean distance between predicted and actual positions as low as possible.



```
67/81 dots in all pics correct
Incorrect Predictions = 3
```

Figure 6: Two further metrics: Dots + Incorrect predictions statistic example

Timings of the runtime were also recorded in seconds, with the expectation of fast runtime on the Pi. To see these in a real output of the Pi during testing, see 38.

## 4.2. Foundations for Dot Detection

Before creating the detection algorithm, preliminary efforts can make the process more efficient. This involves research into how the dots are visible in different colour spaces, how to detect contours in an image that could each be a yellow dot, and some basic screen masking to limit the count of contours. This section focuses on how these elements were worked on, and how relevant literature persuaded choices made.

### 4.2.1. Colour Spaces

Different colour spaces emphasise different aspects of an image, allowing certain features, such as brightness, hue, or saturation, to be more clearly distinguished Gonzalez and Woods (2018). Popular colour spaces were tested, including BGR (RGB), HSV, CMYK and HSL - with HSV and BGR ultimately coming out the best due to the values of the yellow dot pixels. This meant the best identifiers for a yellow dot were using the values of Red **R**, Green **G**, Blue **B**, and then colour saturation **H**, the intensity of the colour **S**, and the brightness **V**. HSV is particularly useful because it separates chromatic content from intensity, making yellow detection more robust under the varying lighting conditions Sharma (2003). For an explanation of how these colour spaces were chosen, through an investigation into these varying pixel values across yellow dots in an image, see B.

### 4.2.2. Detecting Shapes Using Contours and Pixel Values

Another essential step is to identify the contours of potential shapes in the image, where contours represent the boundaries of shapes in an image. Contour detection in general acts as an efficient method for shape based filtering and feature extraction Suzuki and Abe (1985), with OpenCV's contour approximation being very well suited to real time tasks OpenCV team (2025). Each contour detected can be put through a check to see how likely it is to be a yellow dot, in this case nicknamed a "IsADot" function subsubsection 4.3.1, and then kept if relevant.

For this system, the process can be simplified into scanning through each pixel in the image, and keeping only those that are somewhat a yellow colour. All other pixels are removed, and what's remaining is a mask consisting of a lot of contours that all have a possibility of being a yellow dot.

To do this in practice, a high and low boundary of BGR or HSV values is used in the form of two arrays, and the image is scanned for all pixels that are within this range. If within the values, the pixel is saved, if not, then it is removed. The effect of this process can be seen in Figure 7. The original boundary was decided by placing an image into an editing program and finding the BGR value of the centre pixel of the dot. Then 10px was given on either side of this to account for slight changes in colour throughout the videos, and it formed a very rough boundary for yellow dots. This colour range needed refining, which can be seen in subsubsection 4.3.1

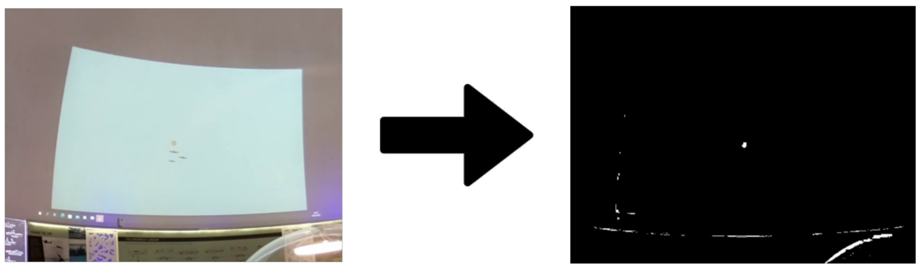


Figure 7: Example mask created from finding pixels in a strict BGR range:

`lowBGR = [172.33, 190.41, 182.20]`

`highBGR = [215.38, 218.97, 208.98]`

focusing on the colour yellow. Black consists of removed pixels, whilst white consists of pixels between this range and thus are somewhat yellow

This creates the contours which, using OpenCV's contour approximation, can each be inputted into the IsADot function to determine if that shape is likely to be a dot.

#### 4.2.3. Screen Masking

The problem with detecting contours through a range is that there can be hundreds of pixels detected, usually in clusters of irrelevant places. This means if each individual one is to be checked, it can be computationally expensive, and it's a requirement for this to run extremely quickly. Looking at Figure 7, a lot of the contours are visible around the bottom, which aren't relevant to the screen at all. As an initial attempt to reduce the number of contours, a further mask is determined to work out the size of the screen and only keep contours in that section. Limiting the area to a smaller location like this is known as finding the region of interest (ROI), which significantly improves speed and accuracy in real time vision systems Szeliski (2010).

This uses the same method as subsubsection 4.2.2, using boundaries focusing on the colour white instead of yellow. It has a few extra processing tasks that essentially smooth corners, but it then finds the biggest contour and masks that as the screen. Further checks were initially implemented, including a check to see how many sides it had and if it was somewhat rectangular, but with the footage including people walking in front of the camera, sometimes this caused it to think it had more than 4 sides and not mask properly.

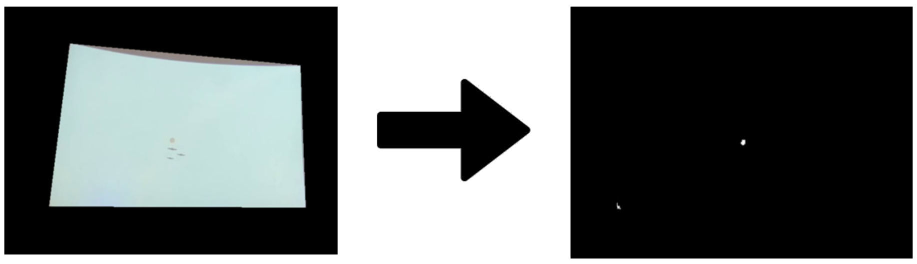


Figure 8: Example mask from Figure 7 but with the use of an additional screen mask to reduce the count of contours

## 4.3. Method 1: Statistical Probabilistic Approach

The first approach of detecting yellow dots from the still images consisted of utilising a range of statistical analyses to determine if the detected shape was in fact a yellow dot. The question posed was *what specifically identified the yellow dots as yellow dots*?

- **Circularity** - Each dot was assumed to be approximately circular, so the roundness of the shape was a key feature.
- **Colour** - The dots were expected to fall within a certain range of yellows, making colour another important factor.
- **Size** - Since the camera remained fixed, the size of the yellow dot was consistent and could serve as another indicator.

For each of these, the probability that a shape was a yellow dot could be estimated using statistical distributions, for instance a shape that is slightly yellow but very small might score highly in colour based classification, but receive a low probability score based on its size.

This entire method can be simplified into this flowchart:

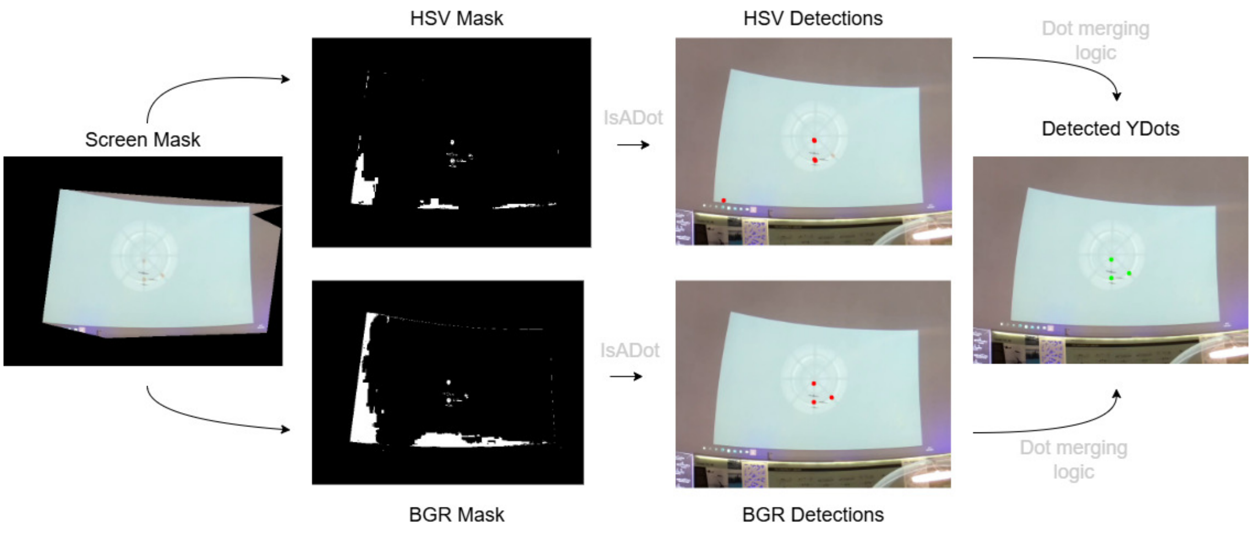


Figure 9: Flowchart illustrating the statistical method used for yellow dot detection. Red nodes represent predicted dots from individual colour spaces (HSV and BGR), while green nodes indicate the final merged dots after applying the dot merging logic - with an incorrect HSV detection successfully discarded through this merging process.

The key steps are:

1. A screen mask, subsubsection 4.2.3, is applied to remove irrelevant parts of the image.
2. A mask is applied with varying thresholds for both HSV and BGR colour spaces and contours are created for both colour spaces (subsubsection 4.2.2).
3. Each contour from both colour spaces are passed into the IsADot function, subsubsection 4.3.1, which checks how likely the contour is to be a yellow dot and returns true if the likelihood is above a set threshold.
4. All these predicted dots from the HSV and BGR colour spaces are combined using "dot

merging logic", subsubsection 4.3.2, to decide which are most likely real dots and which are not.

For a summary of results, see E.1

### 4.3.1. Creating the IsADot Function With Real Data

Building on the preliminary steps, the next phase involved using real dataset images to refine the detection of yellow dots across all frames. Currently, the only determining factor of whether a contour was a yellow dot or not, was the estimated colour boundary using a singular image. The following steps outline some further methods of determining whether a contour is a yellow dot, which combined together form the IsADot function.

#### 4.3.1.1. Extracting HSV and BGR Values From Footage

The most effective way to define accurate colour boundaries for detecting yellow dots was to extract the exact colour values of each dot across all images and visualise them as a distribution. This allowed for a clear understanding of how the yellow values were spread, making it easier to adjust the boundaries to contain the full range of all valid detections. To do this, code was created that:

1. Goes over each image
2. Takes 3px radius mask around each true dot centre in the image
3. Finds average colour of all pixels in mask

This resulted in the following graphs:

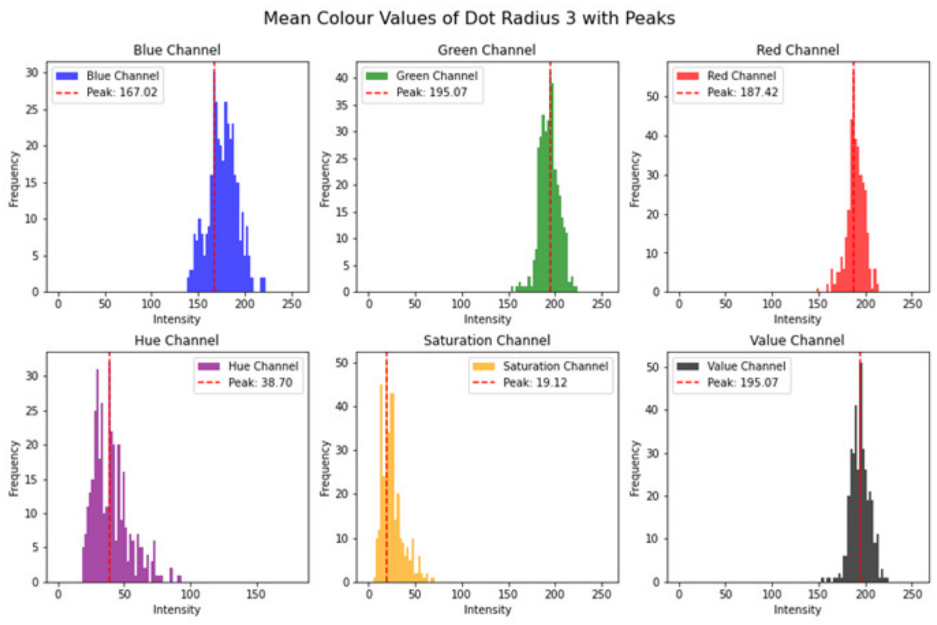


Figure 10: Output of all colour values across every yellow dot in the footage

The distributions form a bell curve, suggesting that a normal distribution function can be used to determine the probability of a colour. By working out the standard deviation and mean, the normal distribution function returns the likelihood that a given pixel's colour value corresponds to a yellow dot - it predicts the probability of a shape being a yellow dot based on its colour. Additionally, the standard deviation can help define the HSV and BGR range boundaries discussed in subsubsection 4.2.2, where a range of ± 3 standard deviations captures approximately 99.7% of the data.

#### 4.3.1.2. Extracting Area Values From Data

The next constant listed in subsection 4.3 was the area of the dot, which focuses on the size of the detected shapes found in the boundaries just defined. To do this, each detected shape is put through a check to see how close it is to the true centre, and if within a distance, it is categorised as a successful detection. This way, all false detections are ignored, and all true detection (covering over the true centre) had their areas recorded.

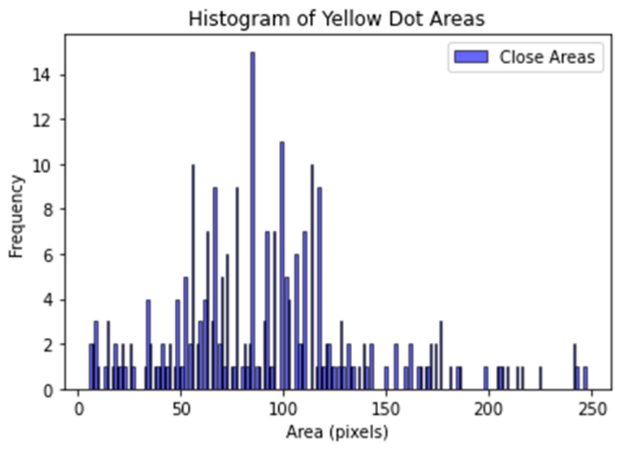


Figure 11: A histogram of all yellow dot areas

When put into a graph, the distribution appeared somewhat random, meaning a slightly more advanced Kernel Density Estimation (KDE) function Virtanen et al. (2020) was used to produce the same effect as colour detection. In this case, the KDE function was used to smooth and approximate the distribution of the dot areas, saving the data and returning probability values. However, when this approach caused the timing to be slightly longer, a normal distribution was attempted and provided similar results in terms of accuracy and missed detections. For the sake of the runtime, this normal distribution is what's used in the final code.

#### 4.3.1.3. Additional Features and Refinements

On top of the checks already determining what is and isn't a yellow dot, some extra checks were implemented:

- **Circularity**
  The other constant mentioned in subsection 4.3, is the circularity of the dot - found with the equation $\frac{4\pi \times \text{Area}}{\text{Perimeter}^2}$. By checking the shape is somewhat circular, determined by if the value from the mentioned equation is above 0.4 as discussed in Sirinukunwattana et al. (2015), there's another quick check for removing false shapes.
- **Negative Probability**
  Further area values were found based on all wrong areas found in paragraph 4.3.1.2. This

meant any areas that were not close to the true detection (thus not covering the true centre) had their areas recorded. This created two exponential graphs of 'incorrect areas', one for each colour space, to be used alongside the current probability of a shape found being a yellow dot.

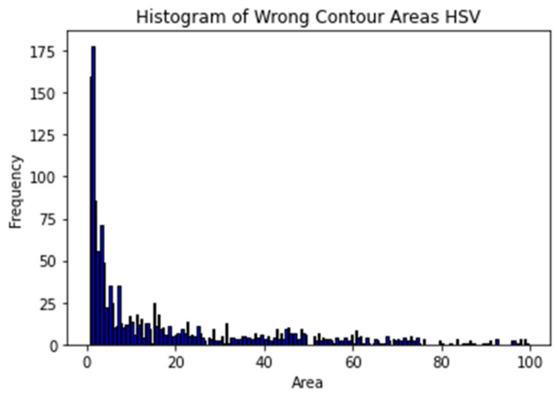


Figure 12: Example histogram to show all wrong areas recorded of yellow dots in HSV colour space

- **Weighting**
  Exponential weighting enabled the positive probabilities to be prioritised over the negative, which helped lessen the effect of a considerable number more readings from latter. The best values were found to be 0.7 to the positive and 0.3 to the negative by comparing accuracy results.

#### 4.3.1.4. Final Probability Equations of IsADot

The final equation for detecting yellow dots in each colour space can be explained with the following:

$$P(\text{YDot} \mid C) = P(\text{Circularity} \mid C) \cdot P(\text{Colour} \mid C) \cdot P(\text{Area} \mid C) \tag{1}$$

$$P(\text{Not YDot} \mid C) = P(\text{Area (False)} \mid C) \tag{2}$$

Where $C$ is a single candidate contour, and:

- $P(\mathit{Circularity} \mid C)$: Likelihood the shape is a yellow dot based on its circularity.
- $P(\mathit{Colour} \mid C)$: The combined probability that each of the colour channels (HSV or BGR) matches the expected values, each worked out using their respective normal distributions.
- $P(\mathit{Area} \mid C)$: The probability based on the shape's area, calculated using normal distributions of areas of known yellow dots.
- $P(\mathit{Area\ (False)} \mid C)$: The probability based on the shapes area in comparison to non dot areas, using an alternative exponential model fit to incorrect areas.

Which in totality means in order for a shape to be a yellow dot, the following must be true:

$$\frac{P(\mathit{YDot})^{0.7}}{P(\mathit{YDot})^{0.7} + P(\mathit{Not\ YDot})^{0.3}} > 0.00001 \tag{3}$$

Equation 3: The final equation for the probability of a shape being a yellow dot in either BGR or HSV colour space.

Where the boundary *0.00001* was found by comparing the value outputted from the above equation of any shapes over the true centre (the correctly identified shape) across multiple stills. This boundary is the same in both HSV and BGR colour spaces.

### 4.3.2. Dot Merging Logic

"Dot merging logic" refers to the combining of detections from two different colour spaces, HSV and BGR, to increase the accuracy of identifying true yellow dots. When acting individually, each colour space may detect different contours as likely yellow dots, meaning two seperate possible dot lists are created. This logic helps filter these results, resolving overlaps and ensuring only the most probable dots are kept. This acts as an essential step for reducing false positives. The logic can be explained in the flow diagram Figure 13, or the explanation following.

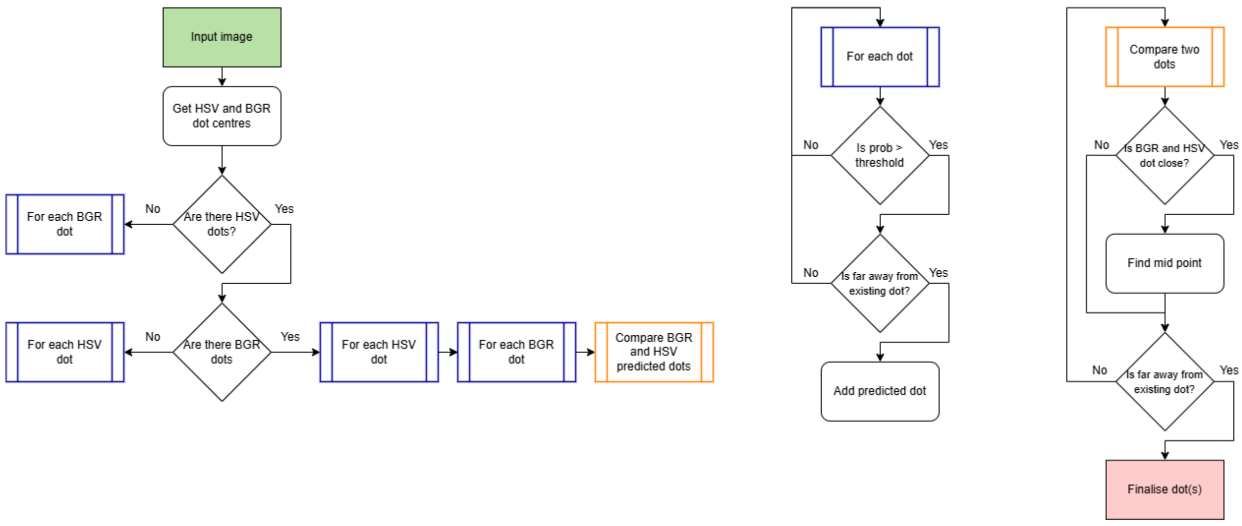


Figure 13: A flow diagram of the dot merging logic, where each dot is compared from the two colour spaces through probability thresholding, distance filtering, and midpoint averaging for overlapping detections. Green blocks act as inputs, red as outputs.

- If dots are found only in BGR colour space (HSV finds zero dots), it checks whether any of the BGR detected dots hit a confidence threshold of 0.95. If so, it adds this dot to a list of confirmed dots, unless it's already close (within 20px) to a previously added dot to avoid duplicates.
- If both HSV and BGR detect dots, the system first evaluates the HSV detected dots. Each dot that meets the 0.95 threshold, and is not too close to an existing detection, is added to the final list. The same process is repeated for BGR-detected dots.
- To further improve accuracy, if an HSV and a BGR detection are close to each other (within 20 pixels), their midpoint is calculated. This average position is then added to the final list, provided it isn't too close to a dot already included.

This method ensures dots found are most likely to be a true yellow dot, whilst avoiding the possibility of duplicates and possible problems if there are no dots found in one of the colour spaces.

#### 4.3.3. Results And Evaluation

The following results, see E.1, are based on the setup described in subsection 3.2 where the final 10 of each folder were used as testing data, and use the metrics discussed in subsection 4.1.

- **Mean distance of 14.10px**, which when compared to the radius of a yellow dot being around 4 px, means it's off by about 3/4 yellow dots. This performance isn't ideal but, although it certainly could be improved, isn't unusable in the system.
- **66/81 dots correct** means it has a success rate of **81.5%** across all dots in the tested pictures. This doesn't quite meet the wanted percentage listed in the test cases in A.2, but comes very close.
- **788 missed predictions** causes a little concern as this seems a very high number, and its effect should be checked in the real-life application.
- The confusion matrix showed extreme overfitting, with almost all results having a higher count of predicted compared to the actual. It does show that some zero dot count images identify less false predictions in general, although still more than the zero.

#### 4.3.4. Conclusion

This method utilising probabilities gives a solid framework for detecting yellow dots based on different shape properties like colour, area and circularity - backed by empirical distributions found by analysing real data. Through these models, the system can successfully distinguish yellow dots from noise or other irrelevant shapes with an 81.5% accuracy rate with a reasonable mean distance of 14.10px.

However, the high false positive rate and signs of overfitting, shown in the confusion matrix, highlight areas for further improvement. Despite these limitations, the method's reliability in detecting the dots, despite the wrong predictions, make it a strong candidate for use in the full system.

### 4.4. Method 2: Patch-Based CNN Detection Model

With the false detections being so high, research into how to improve results provided a CNN as an alternative method that could help detect dots. This in practice involves training a tuned deep learning model on a set of square image cutouts or "patches", each labelled as either containing a yellow dot or not. The model learns to distinguish between the two, and when presented with new, unseen patches, it can classify them accordingly.

In the same way, the previous method had an IsADot function that outputted a value representative of its confidence of it being a yellow dot, the CNN takes in a patch and outputs a confidence score between 0 and 1, representing the likelihood that the patch contains a yellow dot. In this way, the CNN effectively serves as an automatic IsADot function.

### 4.4.1. Patch Creation

The first step was using the training data (the first 40 of each folder as described in subsection 3.2) to create patches to be fed into the model as training. A folder in which to place these patches had two children, 'YDot' and 'NoYDot' which would be populated with patches containing yellow dots and then just background noise respectively.

True centres in the .lab files of each image were used to iteratively go through and extract a perfect patch, but this only created one YDot patch for each dot, which wasn't very many to train from. A further few were created using the centre of the areas found in paragraph 4.3.1.2, giving a few more patches that were slightly off centre which should help the model generalise better since it introduces noise into the training data. A lengthy step here was to check each patch individually too, as feeding in an incorrect patch as a yellow dot would confuse the model.

To fill NoYDot patches, the centre of every wrong area found in paragraph 4.3.1.2 was used, creating thousands of patches to train the model on what not to look for. This created quite an imbalance, and would effect the models training substantially if it were not for the balancing discussed in subsubsection 4.4.2. Again, patches were checked individually where possible, as containing a yellow dot would confuse the model. However, due the vast quantity, and future balacing efforts, it didn't pose as serious a risk as with YDot patches.

#### Patch Size Selection and Results

The sizes of these patches were not thought about at the start, but were definitely a key feature of this process that needed adapting further on. This is illustrated by Table 3 which shows results across varying patch sizes.

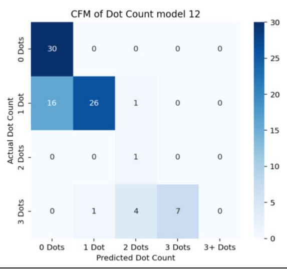
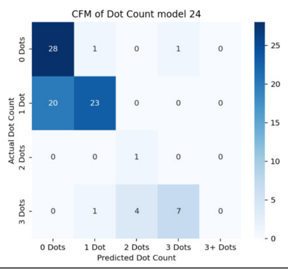
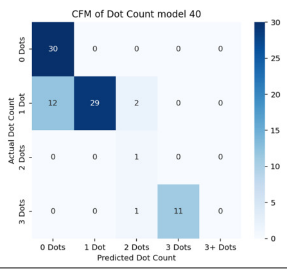
| **Metric** | **20×20px** | **30×30px** | **40×40px** |
|---|---|---|---|
| Mean Error (px) | 2.21 | 2.67 | 2.00 |
| Correct / 81 | 59 (72.8%) | 67 (82.7%) | 55 (67.9%) |
| Wrong Predictions | 2 | 2 | 4 |
| Confusion Matrix |  |  |  |

Table 3: Comparison of Dot Patch Sizes

At the start 20×20px extracts were used as it seemed a sensible size, but this caused problems as the dots filled up most of the space leaving very little for the model to learn from. Another size, 30×30px, provided better results with more correct dot predictions, suggesting the model learned best using edges and surrounding data - but the larger the patch, the longer the processing time. As a compromise, 40×40px patches were extracted to capture more surrounding data, which when tested, performed the worst with a lower accuracy, so were then cropped down to 30×30px in the code before processing. This approach proved to be the most efficient, as it allowed for a consistent patching process where larger patches could be reused and resized, which saved time and avoided the need to re-extract data - a patch can be reduced in size, but not extended to reveal information that wasn't captured initially.

#### 4.4.2. Model Creation

A CNN works by scanning over images with filters to detect patterns like edges, shapes or textures. These patterns are then combined through layers in an attempt to identify more complex features. This means in the creation of the model itself, the effectiveness of this pattern finding is down to the tuning of its parameters, where some important ones include:

- **Number of layers:** The more layers the model has, the more complex patterns it can learn, but this comes at the cost of longer training time and risk of overfitting.
- **Filter size and number:** Filters help detect different features, with their size helping determine the level of detail for the model to focus on. You can also increase the number of filters, increasing the number of features that can be learned.
- **Pooling layers:** These help reduce the image size whilst trying to keep all important information - helping the model work more efficiently.
- **Dropout and regularisation:** These help prevent overfitting, trying to make the model less reliant on a singular feature or connection and encouraging it to find new ones.
- **Learning rate:** This controls how fast the model updates when training, where varying the rate can help it find a better accuracy
- **Class balancing:** When there's an abundance of a singular type of sample, the model can become biased. Methods can be used to try and negate this effect, giving under represented classes bigger weights or creating new samples based on valid ones.

An initial model was constructed using TensorFlow Abadi et al. (2015), with a very simple architecture to create a basis. This version was intentionally minimal, consisting of:

- A single convolutional layer with ReLU activation
- A max pooling layer
- A flattening layer
- A dense output layer with a sigmoid activation

This model lacked regularisation, which resulted in poor performance. However, it provided a useful starting point for tuning and improvement where slow improvements saw accuracies improve, ending with the following model:

- Two convolutional layers with ReLU activations, each followed by max pooling and dropout to reduce overfitting
- L2 regularisation on convolutional and dense layers to reduce complexity
- A fully connected dense layer with dropout to combat overfitting more
- A sigmoid output

Balancing was an important step with the emphasis on the difference in patch generation quantity, with there being far more incorrect dots than correct ones leading to a natural class imbalance. Without correction, the model would have likely learned to favour negative predictions. To address this, both *RandomOverSampler* Lemaître et al. (2017) (Increased the number of valid samples by randomly duplicating them to match the number of negatives) and *class weighting* Pedregosa et al. (2011) (Provided the loss function with weights that gave more importance to underrepresented classes) were used.

Some key improvements on the original model included the use of early stopping, which helped prevent overfitting by stopping training once performance stopped improving, and reducing the learning rate, which allowed the model to slowly focus more to the optimal weights. TensorFlow Lite (TFLite) Abadi et al. (2015) was utilised since research found it to be ideal to compress the model for faster performance on a Raspberry Pi. These changes significantly improved the model's performance and responsiveness.

Visualisations of training, validation loss and accuracy helped confirm that the model was learning effectively and not overfitting. Accuracy was also used as the primary performance metric, while test loss was used to see model confidence.

### 4.4.3. Results

Like before, the IsADot function required a threshold to determine if the yellow dot was or wasn't a yellow dot - in this case, a value between 0 and 1. In the statistical method, a rough estimate was used from visually looking at the shapes and their respective values. To improve this method, detection performance was compared over varying values and graphed to see incorrect predictions against correct ones.

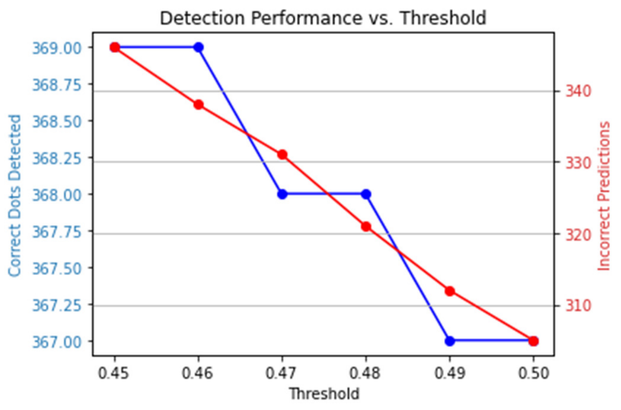


Figure 14: A plot to show the ideal threshold by comparing correct and incorrect dot detections

The point at which incorrect was at its lowest, and correct was at its highest, signified the optimal threshold. In this case, **0.48** appeared to be the best for the model.

The initial CNN, before implementing the improvements, gave strong results:

- **8.55px mean distance** is equivalent to 2 yellow dots away, which suggests very close accuracy across all images.

- **67/81 dots detected correctly** is equivalent to **82.7%** of all dots in testing being found. Again, this doesn't quite meet test cases from A.2, but is extremely close.
- **20 incorrect predictions** was very low, meaning across all images, only 20 times did the model predict dots where there weren't yellow dots.
- The most common error was predicting no dots where there was in fact one, suggesting the model is a lot more harsh than it possibly needed to be. If the threshold of 0.48 is edited, this would likely cause these dots to be detected, but would come at the cost of more false predictions.

When the model was improved, results differed slightly:

- **2.67px mean distance** is near pixel perfect, with this small of an error being more likely down to human error in placement of centres in .lab files.
- **67/81 dots detected correctly** is again equivalent to **82.7%** of all dots in testing being found. This is identical to the previous model.
- **2 incorrect predictions** is far less than the previous model.
- Again the most common error was with one dot being missed, with this actually occuring more often, but other mistakes of overfitting were lessened, like the mistake of it predicting two dots when there was only one happening only twice.

#### 4.4.4. Conclusion

These results are very impressive and suggest this method as a very strong contender for accurate yellow dot detection. Through careful patch selection, class balancing, and model tuning, the final model shows extremely strong performance with a mean distance error of just 2.67px and only two false detections across all test images. While the correct detection rate remained constant at 82.7%, the substantial drop in incorrect predictions, after the improvements, displays the improved precision and confidence of the model. To note is the model's harshness, majorly controlled by the chosen threshold, which suggests there could be additional gains by refining this value depending on what is more appropriate when actually being played (minimising false positives or maximising detection sensitivity). Overall, the use of a CNN, using patch based input, proved extremely useful for detecting yellow dots accurately.

## 4.5. Comparison of two methods

| **Metric** | **Statistical Method** | **Improved CNN** |
|---|---|---|
| Mean Distance from Ground Truth | 14.10 px | 2.67 px |
| Correct Detections (out of 81) | 66 (81.5%) | 67 (82.7%) |
| Incorrect Predictions (False Positives) | 788 | 2 |
| Confusion Matrix Trend | Overfitting | One dot missed |

Table 4: Comparison of performance between statistical method and improved CNN

**Reusable Dataset Pipeline**

The finetuning process is repeatable across both methods, enabling the algorithm to be adapted to new datasets as needed. This is essential as the current dataset being used isn't from the correct vantage point for the final system, and a new dataset will be required from the correct camera location.

**Accuracy of Threshold**

In the CNN approach, more care was taken in finding a threshold to determine what is and isn't a yellow dot in the IsADot function, using tested values on a graph to find the optimal. In comparison, the statistical method used only visual inspection, leading to suboptimal accuracy.

**Mean Distance Error**

The statistical method holds a reasonable 14.10px distance, which although not amazing, is far from unusable. However, it doesn't match the CNN near pixel perfect score of 2.67px.

**Correct Detections**

This is a more specific metric that measures exactly how accurate each method is at finding the required dots, with again the CNN approach beating the statistical method but only by one dot. Neither unfortunately meets the desired percentage listed in test cases, but this is likely due to the overall technique used and the quality of footage. For clarification on this, some example dots that the CNN and statistical method got wrong can be seen in Figure 15.

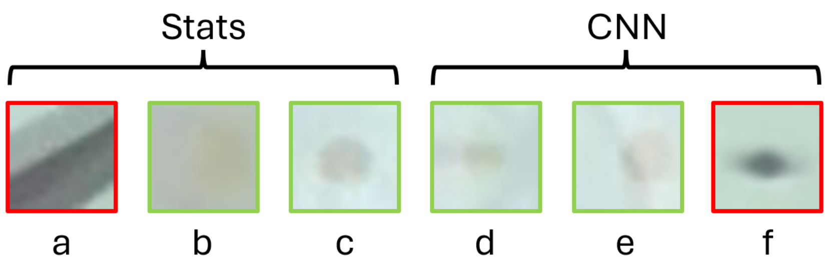


Figure 15: Wrongly identified dots, red showing examples where it was wrongly predicted as a dot, and green showing wrongly identified as noise instead of a dot

Figure 15a and Figure 15f show examples of patches including airplanes that the system got confused on, with this being the most common misclassification for it thinking it was a dot. This could suggest that further checks could be implemented that filter out these. On the other side, Figure 15b and Figure 15d show nearly invisible yellow dots, with Figure 15b actually being misclassified by the CNN too. Examples like these show no matter what method used, there are always some dots that are very difficult to classify correctly - suggesting another method to find these specific cases might be preferred.

This can be confirmed by checking which files have dots completely missed, and seeing if they're the same files across the methods. Figure 16 shows files that have had their dots missed completely by both methods (limited to three for ease of comparison but there are in total six).

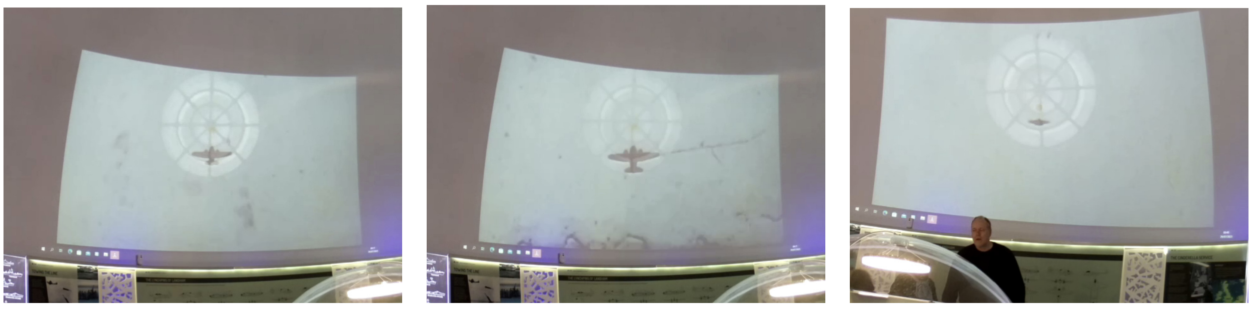

(a) OffLReticule/000041.png (b) OffLReticule/000042.png (c) OnRReticule/000049.png

Figure 16: Example images missed by both methods.

Both Figure 16a and Figure 16b can be seen to be a very noisy and are difficult distinguish the yellow dot. This is similar in Figure 16c where the yellow dot is significantly harder to visualise because of the positioning of the reticule. Looking at these under different colour spaces shares no benefit either, they are very difficult to distinguish even to the human eye. Having these consistently missed suggests they are very difficult under any method, and would require specific treatment.

No matter the reasons for the loss of accuracy percentages, both still show valuable scores, but CNN method remains the most accurate getting that one extra dot.

**Incorrect Predictions**

The comparison of incorrect predictions shows the CNN method achieving only 2 whilst the statistical method achieved hundreds more. This is the most common error of the statistical method, with all predicted assuming far more than there are in actuality. In comparison, the CNN approach's biggest downfall is missing a singular dot.

A key thing to note here is the emphasis on the relation to the reticule being a key decider in which dot to choose. The job of the detection is to find what it thinks are yellow dots, the scoring logic then chooses out of these likely yellow dots, the closest to the reticule centre, assuming that the user is in fact trying to aim for the dot. This means even if there were several incorrect predictions, as long as the closest dot to the reticule is the correct one, the score will be correct.

## 4.6. Suitability for Task

Comparing the two methods, it's clear the statistical method is outperformed by the CNN method, with all results of the latter being substantially better. However, higher accuracy does not necessarily mean it's better suited for the overall system. The key question is whether it's preferable to have more incorrect predictions, which could serve as a fallback in the absence of correct ones, or fewer incorrect predictions with no fallback when nothing is detected. From a gameplay perspective, when a player pulls the trigger, is it better for them to receive a score due to a false positive that suggests they were close enough, or to receive no score at all because the system failed to detect anything?

There's also the discussion of how many of these specifically effect the gameplay. While having