Cameron Shipman

Registration number 100322315

2023

An Anti-Aircraft Shooting Simulator for the Langham Dome

Supervised by Dr Ben Milner

University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

This project explores the realm of Computer Vision for video recognition tasks. The objective is to develop a system which can recognise a ’hit’ or ’miss’ during a session at a WWII anti-aircraft gunner simulation at the Langham Dome military museum. The targets and gunner’s sights are projected onto a screen using different colours, on top of real footage of aircraft.
The basic requirement of the project is to locate these target(s) and the gunner’s sight reticle and produce a reading of whether the two features are in immediate proximity of each other, i.e. a ’hit’. This is accomplishes using a number of object detection techniques including simple colour masking and cross-correlation.
» Results
» Conclusion

Acknowledgements

This section is used to acknowledge support and would typically include your supervisor, other staff, family and friends, etc. that supported you throughout the course of your project.

Contents

1 Introduction

Computer Vision is a rapidly advancing field of image processing technology that encompasses various techniques, including object detection. Object detection itself is a wide subject and employs a variety of techniques including machine learning.

Langham Dome was a WWII-era anti-aircraft gunner training facility based at RAF Langham near the north Norfolk coast. It was originally built in 1942 as a training facility for anti-aircraft gunners which used cutting edge technology that provides the basis for augmented reality technologies we have today. It is one of 6 remaining 'Dome Teachers' (initially 43 built), that were utilised by the RAF to rapidly better the accuracy of trainees during WW2, crucially helping the defence of Britain at the time. (Langham Dome Group, 2014)

From the centre of the dome, moving pictures were projected onto its interior walls to simulate an enemy aircraft in motion. Next to projector apparatus sat a dummy gun which AA trainees used to shoot ahead of the plane so the bullets would hit the plan by the time they reached its distance. This target was indicated by a yellow dot on the film, however, the sights of the dummy gun had a yellow filter so that only the instructor (and other observers) could see the this dot. As the gun was fired, a copy of its sight was also projected onto the interior walls of the dome so the instructor could monitor the trainee's performance.

The dome is now owned by the North Norfolk Historic Buildings Trust and is run by a group of volunteers. It was renovated in 2014 and now serves as a military museum recounting the dome's history and importance during WWII (Langham Dome Group, 2014). Its main attraction is an interactive display which comprises a computer projector and the replica anti-aircraft gun. It works much the same as the original training apparatus bar the digital video projection for the training video.
The volunteers would like to add to the display by introducing a feature to the system so that a gunner's accuracy can be automatically measured and scored accordingly.

1.1 Aim

The basis of this project is to develop and produce a device that will determine if the participant has scored a hit, by detecting where the target (yellow dot) is, where the 'gunner' is aiming (white reticle) and applying some scoring strategy. A user interface will provide control of the system and present the score.

Requirements:

The system must be able to:

- Detect one or more yellow dots. The result will be a list of locations of the dot centres.
- Detect a single reticle. The result will be the location of the reticle centre.
- Eliminate (or at least minimise) false positives and negatives.
- Be executed on the hardware which shall live capture the training video.

The system should be able to:

- Provide a score based on the distance between the reticle centre and the nearest yellow dot centre.
- Present the score to the user/participant. If the system is fast enough the information could be presented in real-time - visually, audibly or both. If not, then a score of hits may be presented at the end of each session.

The system could able to:

- Provide a user interface that allows the user to control the system.
- Involve an option for the gun's trigger to directly inform the system of when it is pressed, via a contact switch connected to a GPIO pin for example.

2 Background

2.1 Computer Vision in Industry

Not only is 'there a lot of research being done in the computer vision field' (IBM, 2019), it also plays an important role in the real world. Its ability to analyse and understand visual data has unlocked new possibilities for automation, efficiency, and innovation. In healthcare, computer vision has revolutionized medical imaging, enabling precise diagnosis and treatment planning. It promotes the early detection of diseases, assists in surgical procedures, and enhances the accuracy of radiological interpretations. (Intel, a)

In the industry of transportation, computer vision has played a big part in advancing autonomous vehicles. By interpreting data from cameras and other sensors, computer vision algorithms enable vehicles to perceive and comprehend their surroundings. This technology enables tasks such as lane detection, object recognition, pedestrian tracking, and traffic sign recognition, making autonomous driving safer and more reliable. As well as in cars, computer vision is employed in transportation systems for traffic monitoring and surveillance, contributing to reduced congestion and improved road safety. (Intel, d)

The impact of this technology extends to many other sectors such as:

- **Retail** - in the enabling of cashier-less stores, 'hyperpersonalized' (Intel, b) shopping experiences, and inventory management.
- **Security** - the facial recognition technology powered by computer vision enhances security systems, enabling biometric identification and access control.
- **Entertainment** - instrumental in augmented reality (AR) experiences, allowing users to interact with virtual objects in real-world environments.
- **Manufacturing** - improving quality control and automation of certain processes through advanced robotics. Its use in sorting items optimises production efficiency. (Intel, c)

And many others; it remains apparent computer vision's key role in today's world is vast and multifaceted, with applications spanning across industries. Its continued advancements and integration with other technologies like machine learning and deep learning promise further potential, making it an appropriate field for the basis of the project's implementation.

2.2 Michael Reeves

It's not just the commercial industry that utilises and benefits from computer vision. For the typical consumer, the technology is easily accessible for various situations they may choose to apply it to. Michael Reeves, a popular YouTuber known for his inventive and humorous engineering projects, has demonstrated various insights into the application of computer vision. He has explored the potential of computer vision in creating interactive and responsive systems. One notable project of his involves using computer vision techniques to modify and command a BostonDynamics Spot® robot to dispense beer into a cup after detecting it (Reeves, 2021).

By experimenting with computer vision algorithms and integrating them into his projects, Michael Reeves showcases the capabilities of the technology in detecting and tracking objects or people, which can be leveraged for a range of applications beyond traditional uses. He brings a unique perspective by employing it in his often unconventional ideas, inspiring others to explore the potential of computer vision in solving real-world challenges.

2.3 OpenCV

Michael Reeves isn't the only person applying computer vision for own personal use (and for other's entertainment). There is an entire vibrant and collaborative community of developers, researchers and enthusiasts who contribute to the development and advancement of computer vision technologies. A significant driving force of this community is the OpenCV project, originally started at Intel in 1998 (OpenCV, 2020).

OpenCV (Open Source Computer Vision) is an open-source library that provides a comprehensive set of tools and functions for image and video processing, object detection and tracking, machine learning, and more. The community-driven nature of OpenCV has led to the creation of numerous resources that benefit both beginners and experienced practitioners. Online forums and discussion platforms allow developers to seek help, ask questions, and share insights. The official OpenCV documentation provides extensive guides, tutorials, and examples to assist learning and development.

From such research into computer vision techniques, OpenCV clearly presents itself as the most suitable for the detection of objects projected on the screen. Its approachable and widely used nature indicates to prove it as a useful tool for the necessary image processing which will be an integral element in this project.

3 Methodology

The key to the success of this project is to choose the correct object detection solution for the target and reticle, which may be require different methods. Considerations include:

- Accuracy of solution - avoiding false positives and negatives.
- Speed of solution - how quickly a result can be provided?
- Overlap of unrelated objects, such as an aircraft obscuring a large part of the reticle.

Another important aspect is the quality of the input data. The captured video needs to be clean (no glitches, random pixel colours, missing frames) and sufficiently high definition, with minimum brightness variation and saturation towards white. If the colour spectrum drifts towards white (or any other colour for that matter, e.g. black), that narrows the differences between the objects (in the colour space), that would otherwise have easily distinguishable colour signatures.

3.1 Toolset

3.1.1 Image Processing Library

It was obvious from the initial stages of the research that OpenCV was going to feature heavily in this project. It is a powerful comprehensive library of tools that include methods for neural network detection, cross correlation, contour detection, drawing, colour space conversion and a whole lot more. OpenCV is open source and therefore appropriate for use here.

3.1.2 Programming Language

Python seems to be well suited to machine learning and data analysis. OpenCV comprehensively supports Python and so for object detection, it should be a good choice. It is also ideally suited to the later mentioned Raspberry Pi pre-installed with Raspberry OS which is a Debian Linux based operating system and includes a recent distribution of Python.

3.2 Yellow Dot Detection

The yellow dot (target) is a good example of when colour detection techniques can be used. The yellow is significantly different to the other objects in the scenario and should therefore be easy to separate out. The proposed method, therefore, is to create a mask from the image, from pixels that are coloured in the appropriate yellow band. I.e. The mask will be set to 1 for pixels that are in the required (yellow) colour range, and to 0 otherwise. A bounding technique can then be used to find the dot(s). Any boxes that are outside the expected size range will be discarded. OpenCV provides functions to do both the masking (`inRange`) and the bounding box finding (`findContours`).

To evaluate the results, a set of images with the yellow dots identified in a set of label/annotation files could be provided. The above algorithm will be then run on all these files and check that the correct number of yellow dots are identified, and the distance between the expected centres and detected centres are below a (low) threshold. This could be plotted on a graph with the y-axis showing the distance between the centres and the x-axis showing the image number, although the best way to do this when there are false positives and multiple yellow dots may require some experiment. This mechanism could also be used to automatically test the program whenever changes are made.
» Todo: Evaluation and include graph

3.3 Reticle Detection

The colour of the reticle is very close to the screen background and is also obscured by aircraft and the yellow dot. So using a simple colour masking technique is unlikely to be appropriate. The first approach would be to use a sliding window technique and cross-correlate a template based on a relatively clean cropped image of the reticle. Remarkably, OpenCV is able to provide both of this functionality in a single function: `matchTemplate()`. This function enables the choice of a correlation algorithm. In particular normalised and non-normalised cross-correlation and correlation-coefficient, seem the most appropriate.
» Todo: List correlaton options
Experimentation will determine the best one in this scenario. Some of the algorithms require a mask, and this may prove useful for the overlapped case. One way to evaluate the algorithms would be to try the different correlation algorithms, on a set of images with the reticle labelled manually in a separate label (or annotation) file, similar to the

yellow dot evaluation (in fact they could be the same set of files), and score them. So for each image, run each algorithm and measure the distance between the actual centre (from the label file) and the one detected by the algorithm. Then this data could be plotted on a graph, a plot for each algorithm. The y-axis would show the distance between the actual and detected centres (or 0,0 if none detected) and the x-axis would show the test image number. That should help determine which algorithm is better. This mechanism could also be used for automatically testing the program whenever changes are made.

» Todo: Evaluation and include graph

If correlation fails to detect the reticle or produces false positives then a machine learning approach may need to be considered. OpenCV provides ways to train a neural network or the weights from a pre-trained model could be used, for example YOLOv7 includes a large community-created dataset. » Todo: Explain YOLOv7

3.4 Scoring

A basic scoring strategy would be to simply compare the distance against a threshold. If the distance is below a certain threshold then a hit can be recorded.

An extension of this would be to provide a 'quality' of hit, by how close to the centre of the yellow dot the reticle centre is, rather than a binary hit or miss.

A scheme is required to determine how many hits can be made during a session or during a single press of the trigger.

One possibility would be to identify when the trigger is pressed and released - when depressed the reticle is displayed, so this can be inferred from just the video feed. It could then be set up so that only one hit can be made during a single press. Otherwise, a hit will end up being counted for every frame that the hit criteria are met (i.e. distance < threshold).

Perhaps a better way would be to count the number of times the hit criteria are met between times when they are not met. For example, when the reticle passes close enough to the yellow dot - a single hit is recorded. A flag is raised so that no more hits are recorded while the reticle remains on target. When the reticle passes too far away from the yellow dot, the flag is then lowered so hits can continue to be recorded. When the reticle next passes close enough, another single hit is recorded. And so on. So the participant can move the gun about and a hit is scored each time the reticle passes over the yellow dot. The flag would be lowered when the reticle disappears, indicating the participant has released the trigger. If they pull it again, and it happens to be pointing at a yellow dot, a hit will be recorded.

3.5 User Interface

The user interface is intended to be accessed via a touchscreen integrated as part of the device so allow for better placement near to where the gunner sits. An on-screen keyboard or a physically connected keyboard should be used by the user in order to be able to control the software and see the score results.

- The user interface will provide a simple start and stop option to control detection/scoring software.
- A display is required to show the current score, with settings that control how it is displayed. For example size of the screen, font, colour, affects.
- As a possible enhancement, a speaker could be attached, which could have explosion sounds (for example) played through it when a hit is made. The user interface would allow control over the volume (including a mute), and the sound file that is played, as well as any other related settings.

For best use-case, the system should be implemented as a GUI for optimal usability which is ideal for a touchscreen display. Although it is intended in the project, producing a GUI within the current timescale may prove too ambitious, especially when Tkinter, the GUI toolkit for Python, can become time consuming to format correctly. For now, the program will be developed with a CLI until the functionality is complete. If time permits, development of a GUI will commence.

3.6 Hardware

For the application, a Raspberry Pi (or other small form factor computer) is suitable as it is a very versatile platform and can work in conjunction with multiple components to complete the task. A camera will also be required in order to capture the screen (i.e. the inside wall of the dome) onto which the yellow dots and the reticle are projected.

Should it be possible to take physical reading from the trigger by connecting to it, the hardware will additionally comprise of some minor electronics that will be connected to the GPIO pins of the Raspberry Pi.

4 Implementation and Evaluation

The implementation of this project requires the creation of an algorithm that can read video and reliably detect the desired objects. It then needs to be able to calculate the distance between the reticle and its contact with the yellow dot(s). The distance will be recorded over time which will be used to calculate an 'accuracy score' for the gunner's session.
» Todo: Add description for hardware and GUI (CLI)

4.1 Software and Code

The main function of the code is to capture video, process it and generate some result.

4.1.1 Video Capture

In the absence of real time capture video, the main loop works through a series of frames read from a video file containing pre-capture 'Dome Shooter' footage. So the first thing the program needs to do is to read the video file.

```
video_file = './langham_dome_0987.mp4'
cap = cv2.VideoCapture(video_file)
while(cap.isOpened()):
ret, frame_bgr = cap.read()
if ret:
frame_number += 1
else:
print(f'Number of frames processed:
       {frame_number} / {num_frames}')
break
# Do stuff with frame_bgr, if ret is True
```

The variable cap contains the video capture object, but has not read any images when constructed. The `cap.read()` method does that. The call to `cap.isOpened()` should be redundant because the content of ret will signify when the loop should end, but is called anyway for robustness.

For each pass of the loop, `frame_bgr` contains a frame of video as a 3 dimensional NumPy array (from the NumPy Library). A NumPy array is a powerful class for manipulating arrays (as matrices among other things). The dimensions are rows (height) x columns (width) x colours (BGR). As with (probably) all graphical applications the top left pixel is (0,0).

Note that OpenCV reads images as BGR rather than RGB (if no colour space conversion is required that is) for historical reasons.

The rest of the loop will process `frame_bgr` to detect the yellow dot and the reticle. In the case of running real-time video capture, obtaining the frame would be different, but the rest of the loop would remain the same.

4.1.2 Window of Interest

To avoid processing more than necessary to, a crude size reduction method was employed. This simply cropped each frame to a rectangle that was sufficiently large enough to contain the screen in all its various positions.

```
screen_area_bgr = frame_bgr[200:950, 300:1550]
```

A new frame image, `screen_area_bgr`, is extracted from `frame_bgr`. The values were obtained empirically.

4.1.3 Yellow Dot Detection

The colour selection was made in two passes.

1. A number of frames were saved to individual image files, and then examined manually using a drawing app (e.g. Paint). Once a set of colours were found, the darkest and lightest were chosen.

2. Then the video was converted to a mask based on this colour range and the mask observed. Adjustments were made to the colour range until the yellow dot was reliably detected for all frames it was present in.

Saving an image to a file is trivial using OpenCV:

```
output_file = f'screen_area_{frame_number}.tiff'
cv2.imwrite(output_file, screen_area_bgr)
```

This only needed performing once.

Originally this process was done in BGR, which was reasonably reliable but not good enough. So the HSV (hue, saturation, value) colour space was tried and this produced much better results, although false positives are still seen.

```
        screen_area_hsv = cv2.cvtColor(screen_area_bgr,
                                       cv2.COLOR_BGR2HSV)
```

Next the colour range was used to produce a binary colour mask, which was displayed alongside the true video. This clearly showed how well the detection was performing. Note that the HSV needs to be able to fit into an 8-bit representation. So for hue this means the normal 360 value is halved, to be in the range 0 to just under 180. For saturation and value, they fit into the range 0 - 255, whereas normally they are a percentage or normalised fraction.

```
lower_yellow_dot = np.array([20,50,160], dtype = "uint16")
upper_yellow_dot = np.array([45,150,220], dtype = "uint16")
yellow_dot_mask = cv2.inRange(screen_area_hsv,
                              lower_yellow_dot,
                              upper_yellow_dot)
cv2.imshow('yellow_dot_mask', yellow_dot_mask)
```

The last line shows the mask and is not normally present in the main loop.

The mask can be used to find the yellow dots, by running a contour finding algorithm. Instead of designing our own, multiple fast algorithms already exist, one of which is currently included in OpenCV (Suzuki et al., 1985).

```
contours, _ = cv2.findContours(yellow_dot_mask,
cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
for c in contours:
```

```
x, y, w, h = cv2.boundingRect(c)
if 10 <= w <= 30 and 10 <= h <= 30 and (w/h > 0.8 or h/w > 0.8):
cv2.circle(screen_area_bgr, (x+int(w/2), y+int(h/2)), 1,
           (0,0,255), 1)
cv2.rectangle(screen_area_bgr, (x-5, y-5),
              (x + w + 5, y + h + 5),(0,0,255), 1)
```

» Todo: Describe RETR_TREE and CHAIN_APPROX_SIMPLE and reasons for choosing them.

To remove (some of) the false positives the results are filtered as shown. The w and h must be in the range 10-30 pixels. In addition, they need to be within 80% of each other, to yield only (roughly) square shapes. The dots should be circular, but after detection, they may not be. For development, the bounding boxes and centre of the detections are drawn in red and overlaid on the video feed.
» Todo: Include figures of testing boxes

4.1.4 Reticle Detection

Cross correlation against a template image is used to detect the reticle. A suitable reticle template was extracted from a frame, by cropping the image and saving it to a file. Like the yellow dot colour range, this was also done empirically.
» Todo: Put template image here

OpenCV provides a function to use correlation to match a template to an image.

```
match_result = cv2.matchTemplate(image,
                                 template_image,
                                 cv2.TM_CCOEFF_NORMED)
```

What is returned is a 2 dimensional nparray containing the correlation results for each position of the sliding window. The above snippet shows that the normalised correlation coefficient algorithm was selected. This algorithm does not require a mask to go with the template image.

To obtain the best result we find the location with the highest result:

```
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(match_result)
```

Note that `min_loc` and `max_loc` refer to the top left position of sliding window, so we need to adjust that to be the centre of the window.

The threshold was obtained by saving all the `max_val` values to an array and then presenting them as a graph at the end of the video.

```
def PrintMaxValGraph():
    plt.hist(max_vals, bins=10)
    plt.show()
```

`PrintMaxValGraph()` is called when the main loop is completed.
» Todo: Try YOLOv7

4.1.5 User Interface

» Todo

4.1.6 Program Structure

`process_video.py`:
Main program containing main loop.

`reticle_finder.py`:
Included by `process_finder.py`, provide function to find the reticle. Also has a 'main' section for running standalone, for testing. Contains functions:

- `ExtractTemplate()` - Called before main loop is started.
- `SearchForReticle()` - Called for each frame.
- `PrintMaxValGraph()` - Called after main loop has finished, temporarily, to find a threshold.

4.2 Experiments, Testing and Evaluation

Testing of the detection was simply done by observation. The bounding boxes and centres of the yellow dots and reticles are drawn over the video as it is replayed. A better testing method would be to run the detection on a number of frames selected from the

entire video, with the expected results in an accompanying file. This is somewhat time consuming and requires a tool to help with this.
» Todo: Add more to reticle set and create yellow dot data set

The most significant problem was the failure to detect all the reticles, especially when the aircraft overlapped a sizeable part of the reticle. In order to solve this, an attempt was made to use a neural network. YOLOv7 was selected as there seemed to be a reasonable amount of information online regarding its use. It also could work with larger images (640x640). Unfortunately, the trained model was not able to detect the reticle at all. This was probably because the (hand-crafted) dataset contained around 90 files, whereas thousands are needed to properly train a model.
Another problem was the false positives when detecting the yellow dots. This may well be down to video quality. Particularly in the last session, the 'shadow' after the aircraft seemed to contain similar colours to the yellow dots, resulting in the algorithm detection this.

5 Conclusion and Future Work

5.1 Yellow Dot Detection

This requires more work to remove the false positives. Some of this may simply be down to the poor quality video used. If not then perhaps cross-correlation with a template and/or mask could be used. That should be sufficient, because the aircraft and yellow dot do not overlap and the reticle is a less prominent colour and probably won't interfere with the detection. We would need to ensure that the template includes a ring of screen colour around the yellow dot, to ensure the shape is included in the correlation, otherwise a large area of yellow would give lots of results, when only the round yellow dots should be detected.

Gaussian blur was tried to improve matters, but did not seem to help significantly. » Todo: Try Morphological Transformations.

5.2 Reticle Detection

» Todo

5.3 Screen Stabilising and Cropping

Because the video used was from a hand-held phone, the screen does move about a bit, albeit not abruptly. A solution could be implemented to automatically stabilise the screen within the image so that it can be more accurately cropped. For two reasons:

- The size of the cropped area can be reduced, so less processing is required.
- The whole frame would contain just the screen area, meaning the object detection would be easier.

The solution could also be able to automatically determine the screen area and crop to that, but that would probably mean each frame would be a slightly different size and require padding. The padding could be the screen edge pixels repeated, so this may actually be a better solution, although scaling may be an issue. The object detection should handle any reasonable screen wobble though and in any installed version of this, the real-time camera would be fixed. So a fixed crop is most likely sufficient.

References

IBM (2019). What is computer vision? *https://www.ibm.com/topics/computer-vision*.

Intel. Advanced Medical Imaging Technology and Devices. *https://www.intel.com/content/www/us/en/healthcare-it/medical-imaging.html*.

Intel. Analytics in Retail. *https://www.intel.com/content/www/us/en/retail/analytics/overview.html*.

Intel. Machine Vision Technology. *https://www.intel.com/content/www/us/en/manufacturing/machine-vision.html*.

Intel. Smart Road Infrastructure. *https://www.intel.com/content/www/us/en/transportation/smart-road-infrastructure.html*.

Intel. What is computer vision? *https://www.intel.com/content/www/us/en/internet-of-things/computer-vision/overview.html*.

Langham Dome Group (2014). LanghamDome.org. *https://www.langhamdome.org/*.

OpenCV (2020). OpenCV's Platinum Membership - Intel. *https://opencv.org/opencv-platinum-membership/*.

Reeves, M. (2021). Teaching a Robot Dog to [Dispense] Beer. *https://www.youtube.com/watch?v=tqsy9Wtr1qE*.

Suzuki, S., Abe, K., et al. (1985). Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46.