computers

MDPI

Article

A Computer Vision Encyclopedia-Based Framework with Illustrative UAV Applications

Terence Morley *, Tim Morris and Martin Turner

Department of Computer Science, The University of Manchester, Oxford Rd, Manchester M13 9PL, UK; tim.morris@manchester.ac.uk (T.M.); martin.turner@manchester.ac.uk (M.T.)
* Correspondence: terence.morley@manchester.ac.uk

Abstract: This paper presents the structure of an encyclopedia-based framework (EbF) in which to develop computer vision systems that incorporate the principles of agile development with focussed knowledge-enhancing information. The novelty of the EbF is that it specifies both the use of drop-in modules, to enable the speedy implementation and modification of systems by the operator, and it incorporates knowledge of the input image-capture devices and presentation preferences. This means that the system includes automated parameter selection and operator advice and guidance. Central to this knowledge-enhanced framework is an encyclopedia that is used to store all information pertaining to the current system operation and can be used by all of the imaging modules and computational runtime components. This ensures that they can adapt to changes within the system or its environment. We demonstrate the implementation of this system over three use cases in computer vision for unmanned aerial vehicles (UAV) showing how it is easy to control and set up by novice operators utilising simple computational wrapper scripts.

Keywords: framework; encyclopedia; deblocking; super-resolution; SfM; drone

Citation: Morley, T.; Morris, T.; Turner, M. A Computer Vision Encyclopedia-Based Framework with Illustrative UAV Applications. *Computers* **2021**, *10*, 29. https://doi.org/10.3390/computers10030029

Academic Editors: Panagiotis D. Ritsos, Kai Xu, Alfie Abdul-Rahman and Rita Borgo

Received: 8 January 2021
Accepted: 24 February 2021
Published: 4 March 2021

1. Introduction

Unmanned aerial vehicles (UAV) are being increasingly used as input directly for specific imaging applications [1–3]. In this paper, we refer to small, low-altitude, commercial off-the-shelf (COTS) UAVs as these have the advantage of mass availability and (relatively) low cost and are already being exploited by major organisations and in many and various imaging applications.

One can imagine an operator flying a UAV and monitoring the surveillance video on the remote control, but this is an inefficient use of manpower or is impractical as a solution when the system scales. What is desired is a system of multiple UAVs that fly autonomously and transmit their observations back to a monitoring station that has image processing capabilities and is manned with the minimum of personnel. To visualise a possible system, imagine a search and rescue team locating a lost climber in a mountainous area [4]. A set of UAVs (with programmed way-points) are launched to record video of different areas and transmit the data back to the base-station. The base-station software automatically monitors the video feeds using an artificial neural network human-recognition imaging system running across available compute nodes. When a high-probability human detection is made, the operator is notified who will then be able to monitor that particular video sequence with more attention. In order to do this, the operator would benefit from functions such as the ability to freeze or slow the stream, correlate features in 3D depth, remove noise or blurring and zoom into an area of the video with either interpolation or super-resolution. The availability of these functions and the extent of their complexity will be dictated by the available computing power.

In order to create a system such as this efficiently, while remaining flexible and configurable, an automated framework is needed that can use 3D image processing software libraries on a set of video streams. The framework also needs to allow for the connection

of specific computational resources as and when needed, while keeping the operator in control and in-the-loop.

The authors are currently performing research for a UK company operating in the maritime sector. Their particular problem has a similar scenario, in that UAVs will be searching for sea-borne craft, particularly in coastal areas. In addition to this *immediate*, hands-on process, there is the need for *after-the-fact* analysis of video. In the event that the real-time UAV monitoring system failed to identify a specific maritime object, it would be of future benefit to be able to re-analyse the collected video streams with different image processing algorithms involving different computational resources. The framework needs to be able to be re-purposed for non-real-time computational use.

We have alluded to a number of modes of operation and processing techniques that should be implemented within the system, and many more can be easily imagined. In the development of a system, a systems integrator might be called in to write software to perform specific functions who may supply custom, standalone hardware. This would require additional expense on staff training, maintenance, upgrades and so on [5,6]. The Royal Navy has identified a need for a data-centric processing framework [7,8] to enable (and require) a supplier to provide smaller sub-systems that will interoperate with other systems (from the same supplier and others).

In this paper, which builds on [9], we take this a step further and propose a framework incorporating an encyclopedia of related knowledge where processing elements can be created as individual modules that interoperate. So, rather than paying for even a smaller inter-operable system, a company can buy software modules that will cross-interoperate. This may require some system integration by the end-customer but would bring the added benefit of agility to their system. As will be shown, we envisage the operation of this framework to be similar to a smart mobile phone operating system and infrastructure model where users can select the appropriate app without needing to know anything specific about particular system integration.

2. Framework

The guiding principles in the Encyclopedia-based Framework (EbF) design are that it is modular, adaptable, scalable and adheres to the ideas of agile software development [10]. Rather than modules being incorporated into a monolithic system, we require that they operate more as building blocks that can be brought together and work together. This is similar to a smart mobile phone operating system and its apps. Apps are often written with a single purpose and when the user selects a function, the current app can pass data to another to perform the required function. In the Android operating system, this mechanism is called an *intent* [11]; in Linux, inter-process pipes or sockets might be used.

Figure 1 shows a block diagram of a system using the EbF. Modules are implemented with Unix-type pipes with Python wrappers, so that they can be chained together to achieve a specific purpose. The modules can be changed by the user as well as by the encyclopedia to produce the best results with the current processing capabilities (with both also being able to alter parameters).

If we consider a module that removes noise from a video stream and produces an output video stream, we can see how intelligence can be added to a system by the use of the encyclopedia. All modules interrogate the encyclopedia to find input parameters defined by the device that created the video, as well as about the computational capabilities available. These might be the noise characteristics of the specific camera on a particular UAV (discovered when the UAV was added to the fleet). The parameters can be dynamically updated by modules monitoring the video streams to detect noise such as dead pixels or dirt on a camera lens. This evolving information in the encyclopedia is then available for multiple purposes.

Software-wrapped library code means that a user can build up their own application pipeline in an intuitive way with minimal scripting. As an example—the user starts with a module that receives the video stream from a UAV and another that displays it in a window.

The stream-receiver module will retrieve the camera model and distortion parameters from the encyclopedia so that an undistorted image is displayed. The user then decides that the imagery is too noisy so selects a noise-reduction module into the pipeline. The video stream display module might have the ability to allow a user to select a zoom area. In this case, the module can offer the choice of re-sampling the imagery or via use of a super-resolution module.

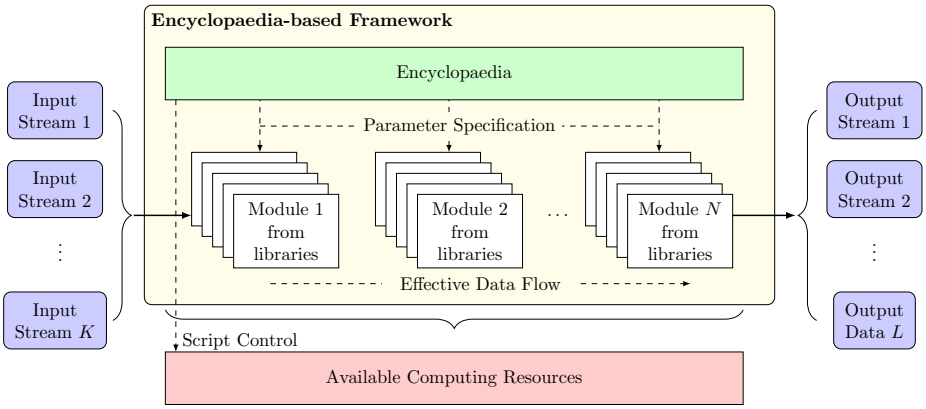


Figure 1. Block diagram of the Encyclopedia-based Framework. The diagram shows its flexible, scalable and configurable nature: pipelines can be assembled using any combination of modules from the libraries. This is directly controlled by the encyclopedia to achieve the user's configuration. There is data flow between the modules and encyclopedia and between the modules themselves (but this is not included in the diagram for clarity).

The EbF gives the user the ability to configure the system on-the-fly. UAVs often have a fixed video stream: fixed resolution, frame-rate and compression level, but it is up to the user how the information is presented to them. The user may require a high-quality, super-resolution view of the video but is happy with a lower frame-rate.

We now provide results and the set-up of three use cases (using Python and OpenCV [12]), which describe how the EbF can take advantage of the available computing resources.

3. Use Cases

This section briefly discusses three use cases for the EbF as shown in Figure 2 and, in each case, the contents of the encyclopedia will be highlighted. The encyclopedia contains parameters for various functions in the algorithm; meta parameters to calculate parameters dependent on the particular circumstances; and input data which may come from input sensors or from the output of other pipelines running within the framework.

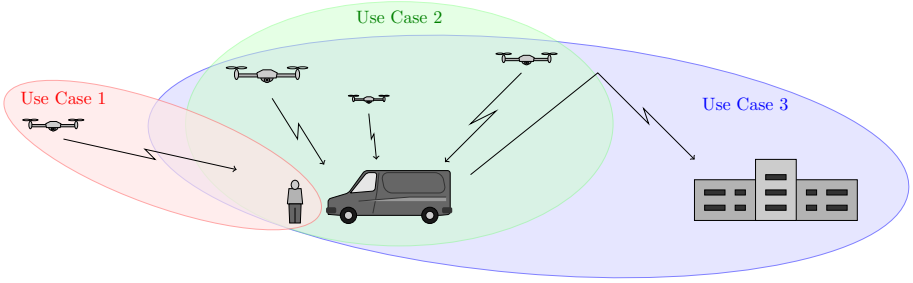


Figure 2. Example use cases. **Use Case 1:** A user with a controller that has the ability to process the video stream. **Use Case 2:** Using more powerful processing power in a base-station (in this case, the computing facilities are located within a vehicle). **Use Case 3:** A system with high throughput computing (HTC) facilities for parallelisable, time-consuming algorithms.

3.1. Use Case 1

Figure 2 (Use Case 1) shows a system with a single user with a single UAV controlled by a mobile phone app. The video stream is highly compressed and therefore exhibits JPEG/MPEG type artefacts.

In this case, the user may be content with a lower frame-rate than that streamed from the UAV for flying and an even lower frame-rate may be sufficient when they want to view something in the scene. We propose an algorithm to reduce the blockiness and other noise present in the streamed video and describe how it is incorporated into the framework.

Our deblocking algorithm, detailed in Algorithm 1, takes previous frames in the video stream and uses them to improve the quality of the current frame. There are a number of parameters that are involved in this algorithm and their choice depends on the current conditions, for example the physical properties of the devices in the system, the environmental conditions and the type of scene being imaged.

Algorithm 1 Compression blockiness reduction.

1: **procedure** DEBLOCKER($fname$, $scale$)

Require: N image file names in list $fname$ and image scaling factor in $scale$

2: **for** $i \in [-N+1, 0]$ **do**
3: $C_i \leftarrow$ READCOLOURIMAGE($fname_i$)
4: $G_i \leftarrow$ CONVERTTOGREYIMAGE(C_i)
5: $f_i \leftarrow$ GETFEATURES(G_i)
6: **end for**
7: **for** $i \in [-N+1, 0]$ **do**
8: **if** useBoundarySmoothing **then**
9: $C_i \leftarrow$ SINGLEIMAGEBLOCKREDUCTION(C_i)
10: $C_i \leftarrow$ UNSHARPMASK(C_i)
11: **end if**
12: $m_i \leftarrow$ FINDFEATUREMATCHES(f_i, f_0)
13: $\mathbf{H}_i \leftarrow$ FINDHOMOGRAPHY(m_i)
14: $W_i \leftarrow$ WARPIMAGE(C_i, $\mathbf{H}_i$, $scale$)
15: $T \leftarrow T + W_i$ ▷ T is a floating-point accumulation image
16: MASK $\leftarrow$ WHITEIMAGE(sizeof(C_i))
17: $D \leftarrow D +$ WARPIMAGE(MASK, $\mathbf{H}_i$, $scale$) ▷ D is count for each pixel added to
18: **end for**
19: $R \leftarrow T/D$ ▷ R is the resulting output image
20: **return** R
21: **end procedure**

The function call on line 9 of Algorithm 1 is based on the method described in [13] and smooths the DCT block borders by applying two 1D Gaussian smoothing kernels to the low-quality JPEG images as shown in Figure 3, where the grey pixels are the border pixels on either side of a DCT block boundary.

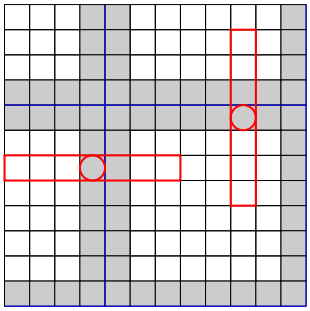


Figure 3. Smoothing the borders of discrete cosine transform (DCT) blocks. The blue lines show the boundaries between the 8×8 blocks in an image. The shaded pixels are the boundary pixels to be smoothed. The red outlines represent the vertical and horizontal 1D filter kernels.

Using ORB (Oriented FAST and Rotated BRIEF) [14] as the feature detector in Algorithm 1, we have the encyclopedia contents shown in Figure 4. The particular ORB parameters for this experiment were found by use of a parameter sweep on high-throughput computing facilities (HTC), see Section 3.3 for details of the system.

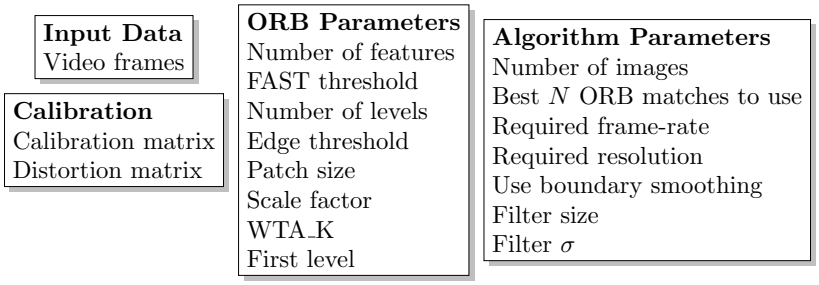


Figure 4. Example encyclopedia contents for Use Case 1. This shows the inclusion of input data, parameters and meta parameters.

It is important to note that these parameters may change for different flights or during a single flight. Before a flight, the encyclopedia can be initiated with parameters from a similar environment and similar conditions. Or, if storage capacity allows, multiple sets of parameters can be stored with the framework choosing the most appropriate one. During a flight, modules monitoring the video streams and sensors can adjust the parameters to suit the conditions.

The blocking artefacts, due to the discrete cosine transform (DCT) compression, are reduced by combining multiple registered frames with pixels weighted according to quality level, to create an improved output, at a higher resolution if required. Use of the algorithm for super-resolution may require more processing power than is available on the mobile phone, and this knowledge is available in the encyclopedia so that it can decide whether or not it can achieve the desired frame-rate at the requested resolution.

Figure 5 shows the results from the EbF Python scripts layer that outputs enhanced frames at a lower frame rate as specified by the operator (5 fps, for example, rather than 30 fps).

An intensity slice through the car registration plate is plotted in Figure 6 for Algorithm 1 with and without the DCT block boundary smoothing. It can be seen that in both cases, the intensity level in the process output follows the original much more closely than the JPEG quality-10 image. This manifests itself as sharper digits on the registration plate with higher contrast.

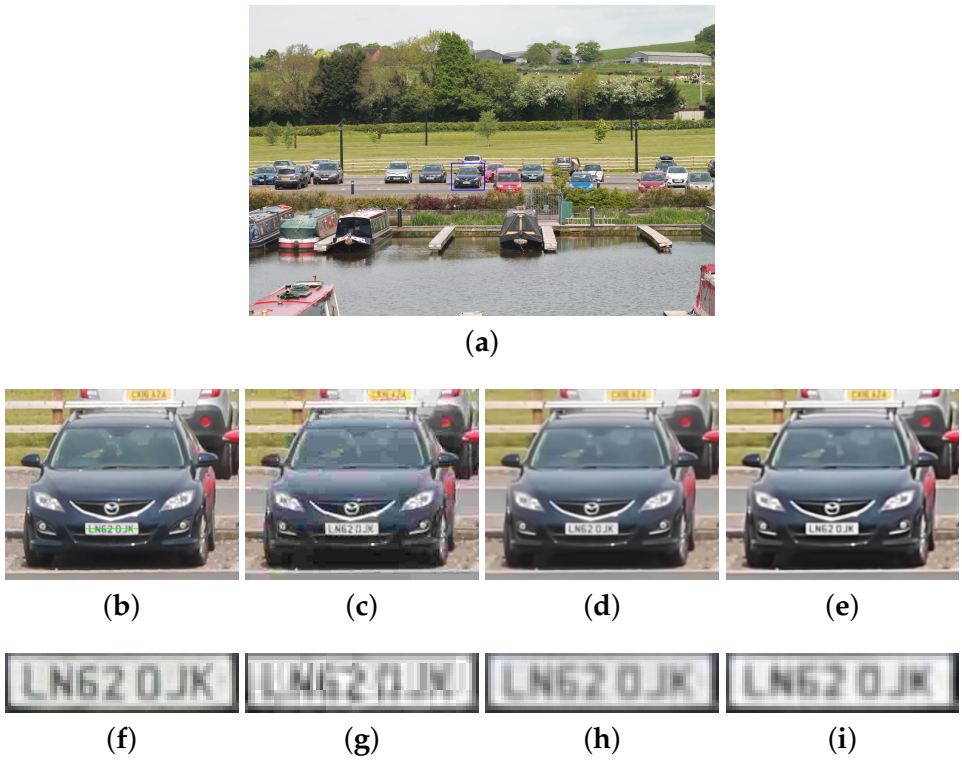


Figure 5. Using Algorithm 1 for multi-frame deblocking. (**a**) Original high-quality reference image (4272 × 2848) showing a region of interest marked by a rectangle. (**b**) Magnified region of interest (304 × 248) showing a line through the number plate from which a comparison is made. (**c**) Zoom of reference image saved at a JPEG quality level of 10. (**d**) Output from Algorithm 1 using 30 input images (without smoothing on DCT block borders). (**e**) Output from Algorithm 1 using 30 input images (with smoothing on DCT block borders). (**f**–**i**) Close-ups of the number plates.

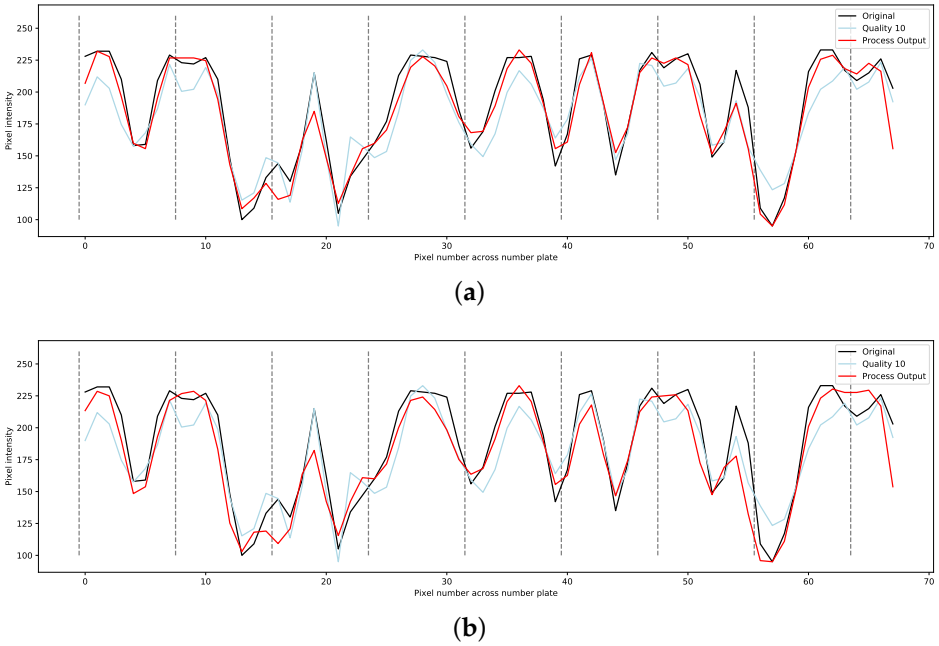


Figure 6. Pixel intensity slice through the number plate from the high-quality version, the JPEG quality-10 version and the output from our method. (**a**) Using Algorithm 1 on 30 input images (without smoothing on DCT block borders). (**b**) Using Algorithm 1 on 30 input images (with smoothing on DCT block borders). The vertical dashed lines show the DCT block boundaries.

As well as being used for reduction of blocking effects, Algorithm 1 can also reduce other types of noise. For example, it can operate on higher quality imagery that is affected by noise. Figure 7a shows a close-up of one of the original images with 40% of the 8×8 blocks corrupted (by keeping only the dc-component of the DCT). Figure 7c shows one of the images corrupted by Gaussian noise with $\sigma = 100$. Figure 7b,d show the results of applying the algorithm to these noisy images. It can be seen in both cases that the algorithm reduces the noise so that the number plates become readable.

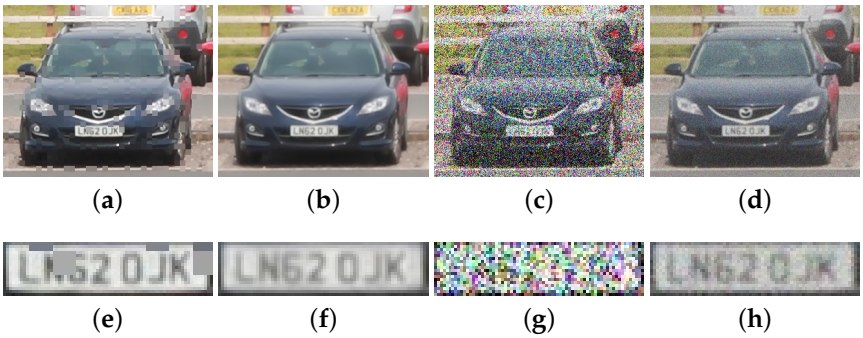


Figure 7. Using Algorithm 1 for multi-frame noise reduction. (**a**) One of a set of 30 images with corruption in a random 40% of the 8×8 DCT blocks. (**b**) The result of applying the algorithm to the set of images with block corruption. (**c**) One of the images corrupted by Gaussian noise at $\sigma = 100$. (**d**) The result of applying the algorithm to the set of images with Gaussian Noise. Each image is a 304×248 crop from the full 4272×2848 image. (**e**–**h**) Close-ups of the number plates.

3.2. Use Case 2

Figure 2 (Use Case 2) shows a fleet of UAVs streaming video to a 3D computer graphics and image processing workstation within a van or ground-station.

This application can survey the surrounding area with a number of UAVs programmed with a set of way-points. To minimise manpower, all of the video streams are autonomously linked back for pre-processing (noise reduction, deblocking, etc.) and a decision-making module streams to the operator.

This brings the human into the loop who can then zoom into an area to look more carefully (see [15]). As has been discussed, compression artefacts, low resolution and motion blurring may all make this difficult, so modules to perform the cleaning of the stream and super-resolution are integrated within the EbF.

Our framework can solve this problem with specific modules that can perform the individual tasks. But there are a number of features that make our framework more useful.

To enable the neural network modules to accurately recognise objects, it is necessary to correct the imagery (a wide-angle lens would distort the shape of objects) and as the framework stores calibration parameters for each UAV in its encyclopedia, this can automatically specify the relevant parameter values. Commercial off-the-shelf UAVs will have different specifications—sensor resolution, frame-rate, focal lengths, colour cast, and so on, so all of this information is stored in the encyclopedia.

The system can be scaled up by adding more UAVs. This is a matter of declaring the UAVs and their specifications in the encyclopedia. These UAVs then become immediately available to the EbF and are ready for the operator to use without the need to shut down the system.

Figure 8 shows the encyclopedia contents for this use case.

Figure 9 shows Algorithm 1 being used to perform super-resolution. The original thirty high-quality images were scaled down to contain a quarter of the number of pixels (from 4272×2848 to 2136×1424) for processing. The low-resolution reference image is scaled up $2\times$ using cubic interpolation as a comparison (Figure 9a). The algorithm then performs super-resolution to $2\times$ and $4\times$ in Figure 9b,c respectively.

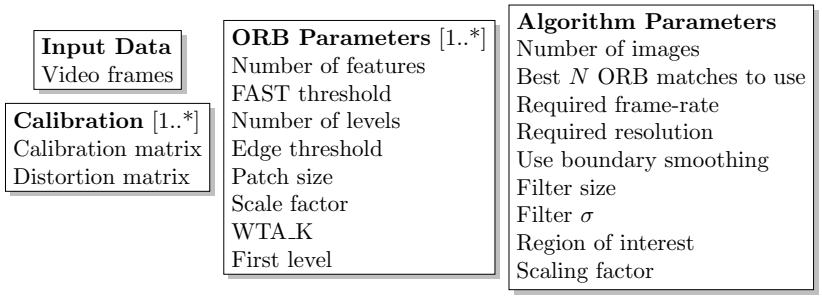


Figure 8. Example encyclopedia contents for Use Case 2. Multiple instances of the calibration and ORB parameters sets have been indicated for each camera, though it is debatable whether different ORB sets are required. * one-to-many.

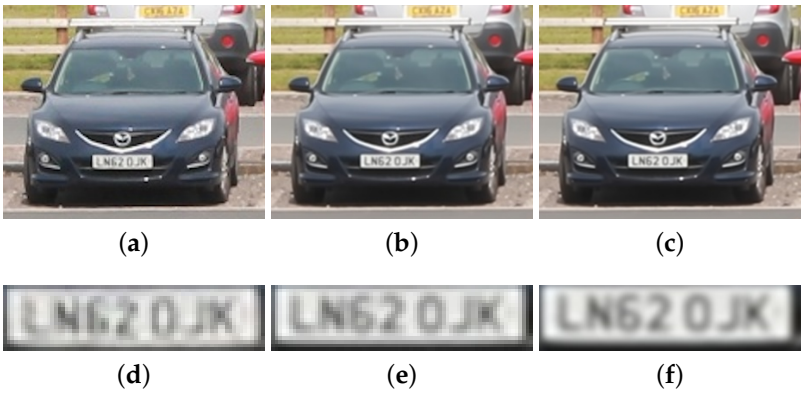


Figure 9. Super-resolution using Algorithm 1 with the thirty original high-quality images scaled to half in both directions. (**a**) The low-resolution reference image scaled up 2× with cubic interpolation for comparison (image size 304 × 248). (**b**) Output from the algorithm with 2× super-resolution (image size 304 × 248). (**c**) Output from the algorithm with 4× super-resolution (image size 608 × 496). (**d**–**f**) Close-ups of the number plates.

To test the ability of a neural network automatic number plate recognition system (ANPR) to read the number plates in the super-resolution image, an online demo system was used [16]. This system is designed to find number plates with particular digit heights in different resolution images. For this reason, the 4× super-resolution image was required. As can be seen in Figure 10, the number plate was correctly recognised.

Figure 10. Automatic number plate recognition (ANPR) using [16]. The recognition software expects the number plate digits to be a certain size so the 4× image in Figure 9c was required for successful recognition. Simply up-scaling the images in Figure 9a,b is not sufficient to enable recognition.

3.3. Use Case 3

To evaluate the framework's use of remote image processing computation, we utilised the University of Manchester high-throughput computing environment (built on HTCondor) [17]. This system has approximately 1,200 CPU cores that are available at any time as well as around 1000–3000 extra CPU cores being available out of hours (in the shape of PCs in the teaching clusters) and has the ability to launch jobs in the AWS cloud.

The freely available HTCondor software was developed by The University of Wisconsin-Madison [18] and is perfectly suited for streaming and simultaneously processing thousands of image processing module commands. The multiple video sequences are cut into one second long pieces so that thousands of seconds of video can be processed.

In order to submit a job to the Condor Pool, a simple text file is required, as shown in Figure 11. This file is generated by the framework and stored in the encyclopedia. The main parts of the submission file are the executable, arguments and input files followed by one or more jobs to queue for processing. The output files are automatically returned by HTCondor. During processing, a log file in the source directory is updated with the progress and an email can be issued to the operator on completion.

```
Universe = vanilla

Requirements = (Target.Opsys == "LINUX" && Target.Arch == "X8_64")
Request_Memory = 1000
Rank=kflops

Should_Transfer_Files = IF_NEEDED
When_To_Transfer_Output = ON_EXIT

Executable = /opt/anaconda2/envs/python37/bin/python
Transfer_Executable = False
Environment = "PYTHONHOME=/opt/anaconda2/envs/python37 PYTHONPATH=python37_packages"
Transfer_Input_Files = orb_param_sweep.py, gcps.cfg, im1.jpg, im2.jpg, ../python37_packages
Notification = Never

Arguments = orb_param_sweep.py gcps.cfg 500 100 20
Log = 500_100_20.log
Output = 500_100_20.out
Error = 500_100_20.error
Queue

Arguments = orb_param_sweep.py gcps.cfg 500 100 30
Log = 500_100_30.log
Output = 500_100_30.out
Error = 500_100_30.error
Queue

# <Blocks repeat for more jobs with different input parameters>
```

Figure 11. Script to submit jobs to HTCondor for an ORB parameter sweep. Notice that the last block of five lines is repeated to run multiple jobs with different parameters. The file is generated by the framework to accommodate hundreds or thousands of parallel jobs.

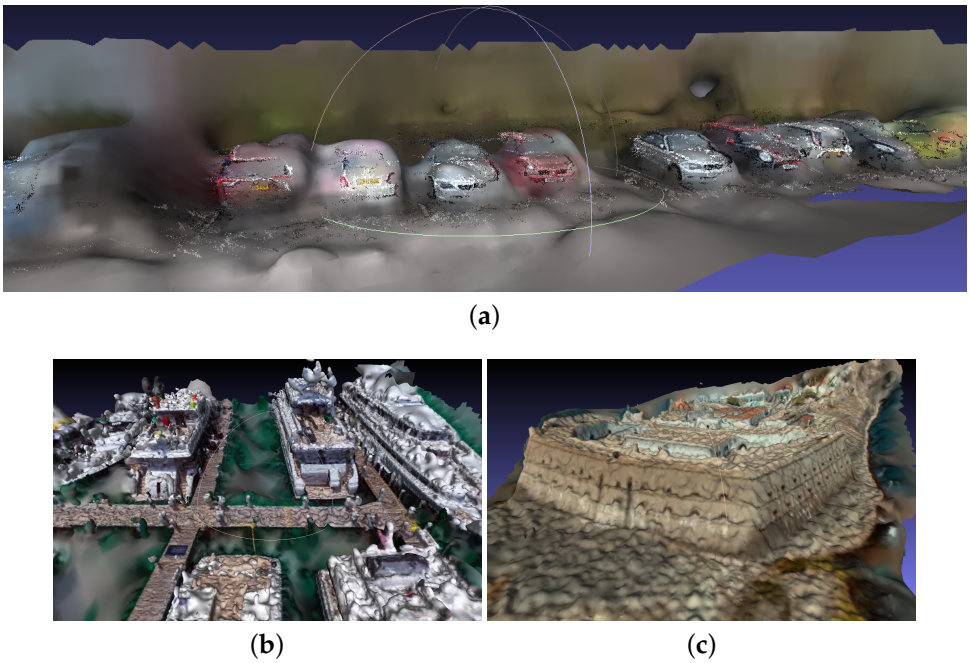
The framework has knowledge of the different levels of computing power available, as well as the latency rates acceptable by the operator; the encyclopedia also stores the Python scripts to call each compute instance (which may be fixed or generated by the framework).

In this case study, we address a problem that requires more processing power than the previous two, requiring high throughput computing facilities, see Figure 2 (Use Case 3). However, the system is no longer constrained by the available local hardware.

This use case covers the non-real-time and more intensive analysis of video data after some event. For example, five UAVs performing surveillance for two hours at 10 fps equates to 360,000 frames. Overnight (12 h) with 1500 nodes on the HTC, detailed processing can be carried out on these streams requiring 3 min per frame.

To demonstrate this using Structure from Motion (SfM), 3D models are created of a scene from all of the video frames using VisualSfM [19,20]. (HTCondor can run binary executables as well as Python scripts).

VisualSfM is flexible in its operation and allows for divide and conquer techniques to be used. The video streams are cut into manageable pieces and feature matches are found between frames on separate compute nodes. The matches can then be brought together to construct the 3D model. This method is described in [21]. The resulting model (rendered in Meshlab [22]) is shown in Figure 12.



(**a**)

(**b**) (**c**)

Figure 12. Creation of 3D Structure from Motion (SfM) models. (**a**) Model of a car park generated from 324 frames at 1920 × 1080 pixels. This image shows the generated 3D surface overlaid with coloured points from the SfM model. (**b**) Model of a harbour generated from 130 frames at 3840 × 2160 pixels. (**c**) Model of a castle generated from 58 frames at 3840 × 2160 pixels. The images in (**b**,**c**) show only the 3D surface model; the overlay of coloured SfM points was not necessary because the higher resolution images made possible a more detailed model from which to create a surface.

4. Discussion and Conclusions

An important feature of this framework is the modularity. Suppliers often sell large systems that are designed to work in a fixed way and work on the supplier's own particular choice of hardware. The client becomes tied to the supplier for training, the upkeep of the system and for any future modifications. They are designed to meet the requirements of the client but often fail to satisfy every need. Indeed, the client often does not know their needs at the time of system specification so agile development is usually cited as a solution to this problem.

Using the EbF, a company can be in control of the development of their own systems and respond quickly to their own changing needs. Because the modules are self-contained, they can be developed and tested outside the client's system and each module can be purchased from a different supplier to reduce costs.

We describe the use of our framework with three implemented case studies on systems with different levels of computational power. With the pace of improvement in computing hardware, the processing can be smoothly moved down to lower levels in the system. We believe that our framework will continue to accommodate this advancement by its intelligent, adaptable operation.

Author Contributions: Conceptualization, T.M. (Terence Morley), T.M. (Tim Morris) and M.T.; funding acquisition, T.M. (Tim Morris); investigation, T.M. (Terence Morley); methodology, T.M. (Terence Morley); software, T.M. (Terence Morley); supervision, T.M. (Tim Morris) and M.T.; writing—original draft, T.M. (Terence Morley); Writing—review & editing, T.M. (Tim Morris) and M.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Engineering and Physical Sciences Research Council, UK (PhD studentship) and BAE Systems (Maritime—Naval Ships).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Authors can confirm that all relevant data are included in the article.

Acknowledgments: We would like to thank the Engineering and Physical Sciences Research Council (EPSRC) and BAE Systems for funding this research. Our thanks also go to the Research IT team at the University of Manchester for their help on using HTCondor and the upkeep of this valuable resource.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Shakhatreh, H.; Sawalmeh, A.H.; Al-Fuqaha, A.; Dou, Z.; Almaita, E.; Khalil, I.; Othman, N.S.; Khreishah, A.; Guizani, M. Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges. *IEEE Access* **2019**, *7*, 48572–48634. [CrossRef]
2. Fan, B.; Li, Y.; Zhang, R.; Fu, Q. Review on the Technological Development and Application of UAV Systems. *Chin. J. Electron.* **2020**, *29*, 199–207. [CrossRef]
3. Muchiri, N.; Kimathi, S. A review of applications and potential applications of UAV. In Proceedings of the Sustainable Research and Innovation Conference, Nairobi, Kenya, 4–6 May 2016; pp. 280–283.
4. Goodrich, M.A.; Morse, B.S.; Gerhardt, D.; Cooper, J.L.; Quigley, M.; Adams, J.A.; Humphrey, C. Supporting wilderness search and rescue using a camera-equipped mini UAV. *J. Field Robot.* **2008**, *25*, 89–110. [CrossRef]
5. Stoica, M.; Mircea, M.; Ghilic-Micu, B. Software Development: Agile vs. Traditional. *Inform. Econ.* **2013**, *17*, 64–76. [CrossRef]
6. Shaikh, S.; Abro, S. Comparison of Traditional & Agile Software Development Methodology: A Short Survey. *Int. J. Softw. Eng. Comput. Syst.* **2019**, *5*, 1–14.
7. Green, H. Creating the Navy of the Future. Available online: https://www.baesystems.com/en/cybersecurity/blog/creating-the-navy-of-the-future (accessed on 16 June 2020).
8. BAE Systems Naval Ships. The Source. 2020. Available online: https://admin.ktn-uk.co.uk/app/uploads/2020/07/CM193520.02.09_SDK_Infographic_Newsletter-template.pdf (accessed on 1 August 2020).
9. Morley, T.; Morris, T.; Turner, M. Encyclopaedia-based Framework for 3D Image Processing Applications. In *Computer Graphics and Visual Computing (CGVC)*; Ritsos, P.D., Xu, K., Eds.; The Eurographics Association: London, UK, 10–11 September 2020. [CrossRef]
10. Fowler, M.; Highsmith, J. The agile manifesto. *Softw. Dev.* **2001**, *9*, 28–35.
11. Brahler, S. Analysis of the android architecture. *Karlsr. Inst. Technol.* **2010**, *7*, 3–46.
12. Bradski, G. The OpenCV Library. *Dr. Dobb's J. Softw. Tools* **2000**, *25*, 120–125.
13. Park, H.W.; Lee, Y.L. A postprocessing method for reducing quantization effects in low bit-rate moving picture coding. *IEEE Trans. Circuits Syst. Video Technol.* **1999**, *9*, 161–171. [CrossRef]
14. Rublee, E.; Rabaud, V.; Konolige, K.; Bradski, G. ORB: An efficient alternative to SIFT or SURF. In Proceedings of the 2011 International Conference on Computer Vision, Barcelona, Spain, 6–13 November 2011; pp. 2564–2571.
15. Ordoukhanian, E.; Madni, A.M. Human-systems integration challenges in resilient multi-UAV operation. In Proceedings of the International Conference on Applied Human Factors and Ergonomics, Los Angeles, CA, USA, 17–21 July 2017; pp. 131–138.
16. ANPR Solutions. Automatic Number Plate Recognition (ANPR) Demo. Available online: https://www.anpronline.net/demo.html (accessed on 6 December 2020).
17. Cottam, I. The Evolution of "DropAndCompute"—Easing the Pain of Accessing High Throughput Computing (and Similar) Facilities. Available online: http://www.walkingrandomly.com/?p=3339 (accessed on 16 June 2020).
18. Thain, D.; Tannenbaum, T.; Livny, M. Distributed computing in practice: The Condor experience. *Concurr. Comput. Pract. Exp.* **2005**, *17*, 323–356. [CrossRef]
19. Wu, C. VisualSFM: A Visual Structure from Motion System. 2011. Available online: http://ccwu.me/vsfm/ (accessed on 16 June 2020).
20. Wu, C. Towards linear-time incremental structure from motion. In Proceedings of the 2013 International Conference on 3D Vision-3DV, Seattle, WA, USA, 29 June–1 July 2013; pp. 127–134.

21. Agarwal, S.; Furukawa, Y.; Snavely, N.; Simon, I.; Curless, B.; Seitz, S.M.; Szeliski, R. Building rome in a day. *Commun. ACM* **2011**, *54*, 105–112. [CrossRef]
22. Cignoni, P.; Callieri, M.; Corsini, M.; Dellepiane, M.; Ganovelli, F.; Ranzuglia, G. Meshlab: An open-source mesh processing tool. In Proceedings of the Eurographics Italian Chapter Conference, Salerno, Italy, 2–4 July 2008; Volume 2008, pp. 129–136.