# ­Live Streaming 360° Vision Sensors

# Abstract

*This investigation aims to research and assess performance of various methods to implement 360° Vision streaming capabilities on Land Vehicles as part of DST’s Advanced Vehicle Systems STC in Land Division. Included is the documented attempts to use the FLIR Ladybug5+ 360° camera and to integrate it with the Clearpath Robotics Jackal UGV. The document then makes a number of recommendations with regards to future work and research regarding the Ladybug5+.*

# Introduction

The Advanced Vehicle Systems STC is part of the DST Land Division, and is responsible for developing next generation vehicle systems for all land personnel. One of the emergent technologies with a range of applications is the remote operation space is Virtual Reality, which provides immersive 360° field of vision to the user. This technology in conjunction with video streaming is highly desirable for the operation of land vehicles as it can provide superior situational awareness without compromising the user’s safety. To this end, the research conducted in this paper will investigate the early stages of implementing 360° Vision capabilities on a UGV platform, with further applications on similar AVS projects such as LOAVES. The primary hardware for this investigation is the FLIR Ladybug 5 Spherical Camera and Clearpath Robotics Jackal UGV, which served as a host platform for the camera systems.

As a proof of concept demonstration, it was desired to be able to show video streaming from the Jackal UGV to a custom HMD application created by a colleague. In order to achieve this the final camera system was required to stream .jpg image frames in real time to the host server, which would handle the process of displaying them on the HMD. How the images were acquired and processed were subject to the results of this research.

# Methodology

**Literature Review/Initial Testing**

The investigation began with the review and analysis of all relevant documentation relating to the hardware being used. This included the reference material for the intended hardware selected by the project supervisor, as well as general background reading. This information became the foundation of all subsequent decisions regarding the hardware compatibility. By examining maximum ratings with regards to power delivery and requirements, it was found the Jackal could operate and power the Ladybug 5 without any additional modifications (Clearpath Robotics, 2014) (FLIR Systems, 2017). The only hardware would be an open lead to dc jack to connect the Jackal’s screw terminals to the GPIO Vin and GND pins. Following the review, it was determined the manufacture of the mounting hardware necessary to secure the Ladybug to the Jackal would be outsourced to SES.

The software provided for use with the Ladybug 5 included a proprietary API and SDK, as well as LadybugCapPro, a capture and processing application for Windows 10. Following the install the camera was tested using LadybugCapPro. Image quality was excellent and the processing options made it easy to save in JPEG and bitmap formats with minimal difficulty. Video capture however required additional work, as the software only recorded to a proprietary ‘ladybug stream’ file. This could then be converted into a number of popular video formats such as .h264 and mp4, however took some time to process. At this point it became clear that whilst the camera was certainly capable of producing the image quality desired for this project, LadybugCapPro would not be suitable for the end application. This meant the research focus would need to shift to application development with the Ladybug API (FLIR Systems, 2017).

Before commencing any design work, it was also required to review the relevant defence standards. In this case, the AS-GVA (Department of Defence, 2017) and DS 00-82 (Defence Standard 00-82) were the relevant documents. DS 00-82 requires that for image transfer over a wireless network, a UDP server must be used with either JPEG or JPEG2000 image formats. Given the restriction from the Ladybug API, JPEG was selected as the image format due to it integrating well with the Live Stream Oculus HMD project.

When examining the documentation for the Ladybug SDK and API it became quickly apparent that there would be some major difficulties implementing them in a custom application. The majority of documentation for the Ladybug 5 only referred to the API in passing, with brief examples on what example code was bundled with the SDK. An official help index did exist for the API, however outside simple descriptions of class names and functions, there was little to guide the development process. As a result, much of the knowledge derived about the Ladybug API was found by inspecting and experimenting with example code.

**C# Implementation**

An install of the Ladybug SDK was completed on a machine running Windows 10 to start observing the Ladybug in use and examine example code. Unfortunately, this install was completed correctly and failed to run all provided C++ examples. FLIR support was contacted however the issue was not resolved. The only functional code were two C# examples: *LadybugCSharpEx* and *LadybugProcessStream\_CSharp*.

Initial work comprised of analysing the example code to understand where in the code the image data could best be extracted for use in an image stream. Of the two, *LadybugCSharpEx* was the most suitable for basic experimentation as it outlined the key requirements to run the Ladybug5+ via C#. The method chosen was to call a method of the *LadybugImage* object member variable *pData*, which points to the first element in the array storing all image data. This data can be appended into a string to then dump in a log file. To do this, a string builder was used within *LadybugCSharpEx* to append all image data in real time before saving the file. Following this implementation, the code was then integrated with UDP server code from the Live Stream Oculus HMD project being conducted concurrently with this research.

The knowledge gained from modifying *LadybugCSharpEx* was then used as the foundation of basic application that attempted emulate the C++ example *LadybugSimpleGrab* in C#. All development was completed using DOTNET Core.

**Linux Implementation**

Following extensive attempts to solve the API issues when running the camera on Windows, one of the few avenues left was to pursue a fresh install on Linux. The install requires a complete Ubuntu 16.04 or similar installation. Then the Ladybug install must be downloaded from Point Grey Research. This requires an account with FLIR and specifying the correct OS to find the appropriate file. To install the Ladybug SDK, simply install the .deb file like a regular package. The next step is to adjust the image file size limit within Linux. This process is detailed in the SDK Readme, and requires editing the /etc/default/grub file to support larger image files over USB. Following this install all example code in C++ and C# should be able to compile using the included makefiles.

Of all the examples available for use within the SDK, *LadybugSimpleGrab* was selected as a starting point. This was due to it being capable of saving the desired image frames that could be used as a video feed for the HMD display being developed by the Live Stream Oculus HMD project. In its initial form, the example would initialise the camera, take a picture using all 6 cameras on the Ladybug 5+, then save them as a bitmap. This would result in 6 independent images corresponding to each camera on the device.

Modifications were made to this code to loop the image capturing process a desired number of times, to simulate the recording of video. The implementation was to simply place all image capturing code in a for loop specifying the desired number of frames. Due to being stored as an uncompressed bitmap, each image was extremely large at 120MB per file. This was adjusted by modifying the API specific function *LadybugSaveImage* parameters to save files in jpeg format.

By adding paths to the makefile, it was made possible to use additional libraries to implement POSIX threads and the necessary elements to enable UDP communication. These added libraries were then used to streamline the code by multithreading key processes.

**Networking Requirements Analysis**

Given the desire to utilise the Ladybug5+ as a video source in both wired and wireless networks, it was pertinent to confirm this capability by examining the network speed requirements. Average file size was measured from a sample of thirty frames taken using the Ladybug5+ with the modified application. The application was set to output in jpeg format without stitching any images together, meaning there were 6 jpeg files for each frame. Using the average file size for all jpeg images, it was possible to calculate the file size per each frame, and thus the required data throughput in order to send the frames at a given framerate. Results can be seen in Table 1.

**Application Performance Analysis**

Development and testing were conducted on a Linux workstation running Ubuntu 16.04 with an Intel Core i7-6700 CPU. A sample image stream was captured in the same manner as the Networking Requirements testing, with a sequence of 30 frames comprised of 6 images each. By examining the timestamps of the first and last file created in this way, it was possible to estimate the average framerate the application is currently capable of producing.

**Mounting Hardware**

The mounting hardware for the Ladybug5+ to the Jackal was outsourced to Scientific Engineering Services. The final design was an aluminium block bracket that mounted directly to the mounting holes on both devices.

# Results

The main product this project has produced used the method above to utilise the Ladybug5+ as an image capturing device, controllable via a Linux platform. The code will loop through an image capture process a desired number of times and save the images in a jpeg format. Plans for parallelising this code have been outlined and prepared for implementation, however cannot yet be completed due to restrictions imposed by the compiling method.

**Understanding the Ladybug API**

Modifying *LadybugCSharpEx* was successful in enabling the streaming of image data via a UDP server. This method was then taken to a custom application with the intent of running the same methods without any unrequired elements. Unfortunately, despite employing the exact process identified from before, the program would fail to execute properly, crashing after attempting to use the stringbuilder that enabled the previous version. It is unclear what caused this. Whilst this was an unsuccessful outcome, a number of key understandings about the Ladybug API were found and recorded. They are listed below

* The Ladybug API uses several objects to manage the wide range of variables needed to address the camera. They must be declared before using any other functions provided by the API. These include:
  1. *LadybugContext*: The container for which most other Ladybug functions must refer to
  2. *LadybugImage*: The image object that collects all image data and metadata
  3. *LadybugProcessedImage*: Similar to Ladybug Image but stores additional data once image has been processed. Required to save images to files, but unclear how to properly implement.
* In order to be able to address the camera, the device serial number must be known. During testing this was hardcoded as a variable in the main program. The serial can be obtained from LadybugCapPro from the Open Camera dialogue box. In the menu where a camera can be selected, the serial should be listed in the column next to it.
* Several functions within the API have a variable to control asynchronous operation. This could be very useful when optimising an application using these functions.

The final product of this work was a full initialisation procedure that would correctly start up the Ladybug5+ and ready it for image capture. It was also capable of capturing an image. Given the limited access to documentation and references, no reliable way was found to save any images captured by the device, rendering the line of research moot. Further attempts to restore the C++ portion of the Ladybug SDK to allow further work were made without success.

**Using the Ladybug API in Linux**

The installation of the Ladybug API on Linux will place all example code in */usr/src/ladybug/src/* directory of your computer. Here all code can be compiled and run using the makefile within each example folder. Code can be modified with appropriate sudo privileges, however additional libraries cannot be added without modifying the makefile. This is the only way to add custom libraries to a Ladybug program that has been found.

No attempts at building a C++ program from scratch using the Ladybug API has been successful at this stage. Standard compilers such as g++ are incapable of locating the required libraries that the API utilises. If a programmer had the skills to replicate the makefile compilation method that the examples code depends on then it should be produce custom applications to suit the project needs.

**Networking Analysis**

Analysis of this software design also provides insight into the system requirements when aiming to provide a certain level of performance in the final application. This is shown below. The average file size is taken from observations of the program output, with all subsequent values calculated from the camera properties.

|  |  |  |
| --- | --- | --- |
| **Video Speed (fps)** | **Throughput (MB/s)** | **Network Speed (Mb/s)** |
| 15 | 63 | 504 |
| 30 | 126 | 1008 |
| 60 | 252 | 2016 |

Table 1: Required Data Transmission rates to support Ladybug5+ at different frame rates

This data indicates that in order to provide video at a functional framerate, the Ladybug5+ would need to be used with a network capable of quickly transmitting extremely large volumes of data. This greatly limits options with regards to implementation hardware, especially in systems utilising wireless networks. Note that this also does not account for any additional packetizing data that would be needed to implement such data transfer over UDP.

**Application Performance**

Tests of the application performance were executed on a 3.4 GHz, quad core Intel i7-6700CPU. The first test used no compression format, and produced images as raw .BMP files. The results were obtained by examining the timestamp of each file as it was created to determine the rate at which they were produced, and are summarised in Table 2.

The result of just under 4 frames per second is clearly disappointing. When first observed in LadybugCapPro, the Ladybug5+ was capable of recording video at up to 16 fps. This result is only 25% of that result. Such a low framerate would be near unusable as a useful video feed for a rover or vehicle. However, there are several improvements that could be made. Currently the application that produced these results does not feature any multithreading; each individual image must be saved before the application proceeds to the next image. This means that 6 operations must be completed before a single frame is produced. By removing this inefficiency, it would be feasible to achieve performance comparable to that observed within LadybugCapPro. Based on performance observations when using the Ladybug5+ within LadybugCapPro, it seems feasible that a framerate of around 10-15 fps would be possible to achieve after employing this method.

|  |  |  |
| --- | --- | --- |
| **Frame** | **Timestamp** | **Instantaneous FPS** |
| 1 | 12.16.04 | 1 |
| 2 | 12.16.05 | 2 |
| 3 | 12.16.05 | 3 |
| 4 | 12.16.05 | 4 |
| 5 | 12.16.05 | 5 |
| 6 | 12.16.06 | 3 |
| 7 | 12.16.06 | 3.5 |
| 8 | 12.16.06 | 4 |
| 9 | 12.16.06 | 4.5 |
| 10 | 12.16.06 | 5 |
| 11 | 12.16.07 | 3.666667 |
| 12 | 12.16.07 | 4 |
| 13 | 12.16.07 | 4.333333 |
| 14 | 12.16.07 | 4.666667 |
| 15 | 12.16.08 | 3.75 |
| 16 | 12.16.08 | 4 |
| 17 | 12.16.08 | 4.25 |
| 18 | 12.16.08 | 4.5 |
| 19 | 12.16.09 | 3.8 |
| 20 | 12.16.09 | 4 |
| 21 | 12.16.09 | 4.2 |
| 22 | 12.16.09 | 4.4 |
| 23 | 12.16.10 | 3.833333 |
| 24 | 12.16.10 | 4 |
| 25 | 12.16.10 | 4.166667 |
| 26 | 12.16.11 | 3.714286 |
| 27 | 12.16.11 | 3.857143 |
| 28 | 12.16.12 | 3.5 |
| 29 | 12.16.12 | 3.625 |
| 30 | 12.16.13 | 3.333333 |

Table 2: Frame Rate Test, No compression

**Projected Framerates on alternate platforms**

The test results were run on a 3.4 GHz processor. This is far superior to the processor onboard the Jackal UGV, which utilizes an Intel Core i5 4570T. The 4570T is a dual core processor with base clock 2.9 GHz, and is approximately 15% less powerful than the 6700 in single threaded tasks. This means that a performance drop can be expected if the Ladybug5+ is deployed on that platform. On the current single threaded application this would equate to a drop from 3.8199 fps to 3.2581 fps.

**Mounting Solution**

The mounting solution manufactured following cooperation with SES was delivered on DD/MM. There was some excess clearance between the camera and the bracket, however this was easily fixed using washers on each screw. The mount is rugged and secure, and should be ideal once the software is ready to deploy. It is important to note this mount is designed exclusively for the Ladybug5+ camera and Clearpath Jackal, thus it will not support other systems without further modification. However, this simple design should be easy to reuse on other projects using the Ladybug5+ should the need arise.

# Discussion

The investigation into the Ladybug5+ as potential 360° imaging hardware experienced a number of problems over the course of the project. These mainly stemmed from the difficulties with using the provided software to control the camera. Whilst the Windows application LadybugCapPro worked as expected, the majority of attempts using the Ladybug SDK and API were met with little success.

The initial attempt installed the SDK on a Windows 10 machine, using Microsoft Visual Studio 2017 as the development platform. Upon completion, it was found that all example code written in C++ was broken, and could not be built. Investigation revealed that the compiler was failing to find standard libraries such as *stdlib* and *math.h* despite them being perfectly accessible to other applications. Attempts to manually reference these libraries by linking the direct path would also fail, as any additional dependencies within the referenced libraries would not update and cause the compiler to fail again. At this point, focus shifted to other methods that were showing more promising results due to time constraints. No further effort to implement the C++ code on Windows was made until learning the SDK was only supported for Visual Studio 2015 via support. Unfortunately, this still had no effect on the outcome as all C++ code suffered the same issues as before. At the time of writing this problem has still not been resolved. Should it become desirable to use the SDK on a Windows platform, it is recommended to follow up all issues with FLIR support, or develop using the C# libraries instead.

The decision to move to C# development was purely out of necessity, as at the time that was the only functional methods available to create and modify applications that could control the Ladybug5+. Whilst there was little success in delivering capability to the overall system, the work with the C# API proved invaluable in understanding the way the Ladybug API operated and was designed to be used. This was doubly important given the lack of useful documentation provided by FLIR with regards to the Ladybug API.

As stated in the results, it is unclear why the custom C# application failed to run. The error was isolated to the line of code that called the string builder. This line appended the data from the *pData* pointer to a string in order to be written to a log file. However when this was called, the entire application would become blocked and not resolve without force-quitting the application. It is possible that this might be cause by two processes within the application seeking to utilise the same resource simultaneously, however it is unclear what exactly is causing this issue to arise. There is no multithreading in this program, and no other programs are accessing the Ladybug at the same time as this application was. Nonetheless it would be pertinent to modify this code to use semaphores on key resources to avoid potential blocks.

Another explanation is that the code has not written to the location *pData* is referencing, therefore meaning there is nothing to be written to the string builder. This depends on the implementation of the sb object, however if the function waits until something is appended, this would cause an infinite waiting period that would block the program.

Perhaps the best test to conduct on this issue would be to implement a timeout feature to the call that is causing the block. This would make sure that if nothing is executed after a given time, the code can continue as normal without breaking the entire program. Whilst not solving the issue at the cause, it would allow further work on the code to investigate further.

It wasn’t until after several issues were encountered that investigating potential limiting factors of the concept were investigated. In hindsight this ought to have been done much sooner, as it may have prevented a large amount of unnecessary work. However, given the unique time constraints imposed on this project, it was assumed that several of these points could be overlooked. Regardless it would have been of great benefit to have investigated further in the literature review as the results found here were of great import to the results of the project.

As it stands, the network requirements are technically possible to be met with both wired and wireless routers. Routers often claim output speeds of over 1000 Mb/s on 5 GHz Wi-Fi. However, what has not been tested is if that connection speed can be utilized entirely by two individual devices. If that can be guaranteed, then it would be feasible to assume that the connection would be suitable for use to stream video, otherwise a severe bottleneck could be experienced. The best course of action would be to test multiple different wireless routers and use them to directly stream data between two PCs. If and only if the router can maintain a speed in the range of 500-1000 Mb/s would this system be sound for use.

Of course, a far more reliable system would be to utilise a wired ethernet connection. Ethernet technologies are capable of supporting speeds of 10 Gb/s, which is ample given the results from the Networking Analysis. Such a connection would also likely be the first method used when prototyping a streaming system. This is one of the reasons why establishing this system on a larger vehicle before doing so on the Jackal makes much more sense. The equivalent system for a regular land vehicle is a logical first step for prototype, and has fewer limitations with regards to networking technology. Following a successful test deployment as a wired system, the streaming system could then simply be adapted to a smaller form fact and wireless arrangement should the hardware prove sufficient.

This trend of favouring large scale deployment first continues with the results of the Application Performance tests. 4 fps is simply insufficient to deliver useful vision of the outside world when operating a vehicle such as ASLAV or similar. As discussed, it is highly recommended to attempt parallelisation of the image saving process into independent threads. This would allow 6 processes to run concurrently instead of successively. As a full frame cannot be complete until the last camera image has been saved, the framerate could be increased dramatically by doing so, improving the speed perhaps 4 or more times over. This brings the framerate to around a much more acceptable range of 12-18 fps depending on other overhead factors. However, this solution is only feasible if the processor running the application is capable of supporting such a number of threads. Unfortunately, the Jackal’s Dual Core i5 is likely insufficient for such an intensive task. FLIR recommends a Dual (preferably Quad) core processor running at at least 3 GHz alongside a 512MB Nvidia graphics processor. This is slightly beyond the Jackal’s i5 4570T which only runs at 2.9 GHz.

Based on these figures it seems likely that a performance bottleneck could be experienced by any Ladybug application run on the Jackal’s internal PC. Given that performance is already mediocre with hardware fit for purpose, it seems unlikely that the result of using the Ladybug5+ on the Jackal will yield useful results. Of course, this can be rectified with appropriate hardware upgrades. Should it be possible to upgrade the internal PC of the Jackal to more recent hardware, an improved CPU and GPU would most likely be sufficient. This unfortunately beyond the scope of this project, which aims to use commercial off the shelf parts to keep engineering costs low. Therefore, the conclusion again is that unless hardware can be sufficiently improved, the Jackal UGV is likely an unsuitable platform to deploy the Ladybug5+.

Given the results indicating the platform preference is moving away from the smaller Jackal UGV, alternate platforms must be considered. The nearest and most obvious candidate for testing and deployment would be the LOAVES project. LOAVES aims to test a number of upcoming AVS systems for future use on military vehicles. Given the lack of constraints with regards to computing hardware and networking, it is a sound choice. This is of course dependent on if there is room within the planned slate of projects within LOAVES, but work can continue on the 360° Vision Sensors project regardless. Once the assumption of running the system on the Jackal is removed, a great deal of software development can still be done implementing and polishing the system before direct implementation is required to test.

Other platforms should of course be considered. Clearpath offers vehicles such as the Husky and the Warthog, which offer similar capabilities to that of the Jackal on a larger scale. With a larger deployment platform, it may be feasible to use higher quality computing and networking hardware necessary to operate the Ladybug5+.

Overall, the recommended roadmap to testing and deploying 360° video sensor technology is as follows. Work should continue on the Linux platform, as this is the platform that has been most successful in implementing the Ladybug5+ and its API. Research should focus on a hardwired network solution first, making the system functional, then optimising performance. Reducing total data transmission rate requirements should be prioritised, as this will lead to a more lightweight system, and support deployment on wireless systems. The first proof of concept would be deployed on a larger vehicle platform such as LOAVES. If this demonstration is successful, the viability of wireless version of the system could then be reassessed. The results of this assessment would then decide the direction of further development.

## Conclusion

This investigation looked into the implementation of the Ladybug5+ for use in Land Vehicles, specifically smaller autonomous vehicles such as the Clearpath Jackal. It examined methods to implement the device as a video input to record and stream footage over a network. It did so across two software platforms and documented the performance on each.

The findings were that whilst the Ladybug5+ is capable of recording extremely high-quality footage, it has a number of drawbacks that reduce its viability:

* The software package provided to control the device has poor documentation and unreliable example code that hinders development. It would require a great deal of man hours to effectively develop applications using the Ladybug API.
* The Ladybug5+ has high processor requirements that can limit performance. The custom application developed using the API was only capable of a very low frame rate that is likely bottlenecked by the testing system CPU. It is recommended that future implementations utilise a workstation class CPU to avoid this bottleneck.
* The Ladybug5+ requires a high-speed network in the range of 1 Gb/s or greater to provide functional framerates when streaming. This could be improved upon with custom compression or reduced image quality. Hardwired networks are recommended for future testing in order to supply necessary bandwidth.

All together these results demonstrate that the Ladybug5+ is likely unsuitable for implementation on small mobile platforms such as the Clearpath Jackal. This is due to the limited processing and networking capabilities the platform possesses that could severely restrict the performance of a video streaming application that runs on it. However, there is still potential for the Ladybug5+ to be utilised in larger platforms that can accommodate the networking and processing requirements. Therefore, it is recommended that future research examine the implementation of the Ladybug5+ on LOAVES to test its viability on larger land vehicles. The Ladybug5+’ 360° image capability is still highly desirable for land vehicles in AVS, and it could bring welcome improvements to future vehicles developed by Land Division.

# Bibliography

[1] (n.d.). *Defence Standard 00-82*. Australia: Department of Defence.

[2] Clearpath Robotics. (2014). Jackal Datasheet 2014. Kitchener, Ontario, Canada.

[3] Clearpath Robotics. (2014). Jackal Unmanned Ground Vehicle User Manual. Kitchener, Ontario, Canada.

[4] Clearpath Robotics. (2015). *Jackal Tutorials.* Retrieved from Clearpath Robotics: www.clearpathrobotics.com/assets/guides/jackal/index.html

[5] Department of Defence. (2017, September 15). *Australia Generic Vehicle Architechture Standard*. Australia: Deparment of Defence.

[6] FLIR Systems. (2017, December 9). FLIR Ladybug5+ Techical Reference. Richmond, BC, Canada.

[7] FLIR Systems. (2017). LADYBUG5 Datasheet. Richmond, BC, Canada.