# GRASP Documentation

Nicole Duncan

May 29, 2012

## 1 Introduction

The GRIPS Aspect Program (GRASP) is designed to control acquisition of aspect system data, compression and initial processing. The present version of the software discussed in this note is GRASPv0 dv15. This document provides instructions on how to use the software, a description of the software design and some detail on error handling. This development version is not complete, open issues and next steps are discussed at the conclusion of this document.

## 2 Using the Software

The GRASP software is built for a Linux environment and is currently implemented in the CentOS operating system. The program is built for use with Prosilica cameras and uses the PvAPI development software to communicate with the camera. The PvAPI DLL must be installed and the required header files (PvAPI.h and PvRegIO.h) are available with the PvAPI SDK. Information about the PvAPI functions and attributes is available in a programmer's reference manual, available on the Prosilica website. A second document, the AVT GigE Camera and Driver Attributes guide provides more detailed information on camera and acquisition options.

The most current version of the GRASP software is GRASPv0 dv15. The appropriate make file is included to compile the software. This version of the software does not perform any compression or analysis, it shoots and saves images.

After starting the software, the user is prompted at the command line to enter the desired cadence, exposure time and if they would like to save the images to disk. After entering this information, the program begins snapping photos automatically. The terminal will print a message each time

the camera snaps a picture and each time a frame was successfully queued. If errors occur, an error message will pop up, telling the user what type of error occurred. Error handling messages also pop up, telling the user what actions are being taken to correct the error. The program can be paused or exited and the options altered by pressing Ctrl + C. Pressing Ctrl + C also brings up a list of diagnostics showing how many times certain errors occurred and statistics from the camera stream.

On the GRIPS lab aspect computer, saving to disk causes issues if the full 1MB image is written. This is described in the error handling section, and might be improved on another computer. I suggest not saving to disk or reducing the region of interest (as described below) to test the full 30 image/sec camera capability.

# 3 Software Design

The GRASP software utilizes the PvAPI software development kit (SDK) to control camera acquisition of images. The general philosophy is to use the SDK minimally, instead relying on standard C++ tools whenever possible. For example, we create our own timers and generate interrupts which trigger the acquisition software instead of relying on the SDK's option to snap at regular intervals (determined by some unknown clock). Also, we use standard C++ thread blocking techniques to wait for a frame to be populated instead of relying on the SDK's PvCaptureWaitForFrameDone function.

In the present version, the software queries the user for cadence, exposure time and if the images should be saved. Following these queries, the program automatically snaps images and handles errors as they arise.

The software begins by polling the system to acquire cameras and start the image and acquisition streams. A timer generates interrupts, at the user defined cadence, which queues a frame into a frame buffer and triggers the camera to begin an exposure. The main thread is blocked until the frame completes or is timed out. If a frame times out, the acquisition is taking longer than desired and the acquisition is terminated. When a frame completes, an interrupt is generated which causes the program to process the frame. If the user wishes a frame to be saved, it is during processing that saving occurs. Once the processing is done, the program waits for the next timer alarm, starting the process over again.

Errors are generally handled by restarting the image capture or acquisition streams. Common errors are described later in this document.

An application flow diagram is appended to the end of this document.

## 3.1 Error Handling

Most errors can be traced back to the function calls to queue a frame or trigger the camera to begin an exposure. Since these are functions that are provided by the SDK to interact with the camera, it isn't possible to "fix" the problem. Instead, we use error handling to catch these issues as they arise. Since we only have 1 camera/computer set-up it is not clear if these problems are tied to our set-up or if they persist on other systems as well. Below is a list of common errors and the strategies employed to resolve them. Open issues and next steps to mitigate them are described at the end of this document.

- Software Trigger: Occasionally calls to PvCommandRun(Cameras[i].Handle, "FrameStartTriggerSoftware") do not return. This function generates a software trigger to begin an exposure. In general this call works, but after a while it will simply not return and the remainder of the code will not execute. Usually, restarting the image capture and acquisition streams will cure the problem. However, after $\simeq$10 times the program will not be able to stop the stream, and the program waits indefinitely trying to restart the stream.

- Successive queue errors: When three successive frames are unsuccessfully queued, it is likely that all the following frame queues will also be unsuccessful. Restarting the image acquisition stream after three successive queue errors is adequate to remedy this problem.

- Frame and queue errors: It has been observed that when a frame error (error 16: Data Missing) proceeds a queue frame error, the program will crash. To remedy this issue, the program catches this combination of frame and queue errors and restarts the image capture stream.

- Camera is unplugged: Randomly, the software will report that the camera has been unplugged, although the camera LED indicator lights show that there is a good ethernet connection and it is fully powered.This is an outstanding issue. The steps which will be taken to attempt to remedy this issue are outlined in "next steps" below.

- Saving: Saving the whole 1MB image to disk can create problems which causes the program to crash on my lab computer. This is likely an issue with my computer, reducing the filesize remedies the issue on my machine. You can adjust the size by cropping the image, this is hard coded into the software in the function CameraSetup, under the

heading define ROI. In the function ProcessImages there is a hard coded option to switch between saving as a .bin or .tiff file.

- Image completion time: Usually it takes $\simeq$ 20ms for a frame to be transmitted from the camera into the computer memory. However, there are outliers which will take up to a second to be transmitted. To avoid the outliers, threadblocking and timeouts are used to check if a frame completes within the allotted time. If it hasn't finished, the image is aborted by clearing the frame queue.

# 4 Open Issues

Open issues include the software trigger and camera is unplugged errors described above.

## 4.1 Next Steps

At present, the GRIPS team only has 1 camera/aspect computer set-up. Communication with the camera manufacturer indicates that there might be a problem with the speed/ interfacing of the NIC card and computer with the camera. The general suggestion is to check system performance and speed. This potentiality is being investigated, any information provided by the HEROS team on these issues in their test set-up would be extremely helpful.

The software will be updated to change the camera discovery method. Currently the software polls the system to discover cameras. It is more desirable to open cameras by their address.

It is unclear if simply restarting the image capture stream creates a new thread to the camera without destroying the old thread. If new threads are created each time, this could be the reason why the error handling for the software trigger issue only works $\simeq$10 times before crashing the program.

# 5 Example Console Output

On the following pages are examples showing the program start, take 2 images then exit. Another example shows the error handling messages for a software trigger issue.

Waiting for a camera to be plugged in...

Camera 104533 is now plugged in.

Grabbing camera 02-2185A-06593 with ID 104533
Now setting up camera with ID 104533

How quickly should pictures be processed?
Pictures per second: 20

How long should the exposure value be in microseconds?
Upper limit based on pictures per second: 50000
Exposure time: 20000

Would you like to save the pictures to the hard drive? (y/n): n

Starting camera with ID 104533
Packet Size sucessfully determined.

Frame buffer size: 1228800; 1280.000000 by 960.000000
Camera with ID 104533 is now acquiring images.
This is the TICKSLCM: 20
timer created correctly
timer armed correctly

SNAP 0 : 20120529_190523_169180
Frame re-enque successful. Trigger SNAP.
SNAP 1 : 20120529_190523_268739
Frame re-enque successful. Trigger SNAP.

Displaying settings for camera with ID 104533
---
Images per second: 20
Exposure time in microseconds: 20000
Camera capture paused? false
Saving images taken by this camera? false

Timeout Count: 0
No return from trigger: 0
Successive Queue failures: 0
Frame and Queue errors: 0

Statistics for camera: 104533
Frames Completed: 20
Frames Dropped: 0
Num of erroneous packets received: 0
Num of packets sent by camera and NOT received by host : 0
Num of packets sent by camera and received by host: 16800
Num of missing packets requested to camera for resend: 0
Num of missing packets resent by camera and received by host: 0

Would you like to end the program? (y/n):

SNAP 956 : 20120529_190919_18900
Frame queue successful. Trigger SNAP.
SNAP 957 : 20120529_190919_68854

Restarting Image Capture: trigger fault.
Acquisition Stopped.
ProcessImages skipped b/c: CAMERAS[i].Frames[j].Status != ePvErrSuccess
Frame buffer queue cleared.
Image capture stream terminated.
Image capture stream restarted.
Acquisition Started.

Frame queue successful. Trigger SNAP.
SNAP 958 : 20120529_190920_468759

# Application Flow Diagrams