GOTO DB Access Script v3 User Guide

Setup

- This document is a short guide to the use of v3 of the GOTO Photometry Database query script developed by Ross McWhirter during Q1-Q2 2021.

- The script is currently delivered in a tar archive which must be extracted to a user chosen directory.

- The output folder will contain 5 files: A get_lightcurves.py file which is a Python command-line script containing the code and the target of user requests, An input_example.dat file which demonstrates the format a list of target sources for input to the Python script, this user guide GOTO_dbaccess_guide.pdf, and two files, pipinstalls.sh and pip3installs.sh which are shell scripts to install the dependencies of this software using the pip or pip3 commands depending on the versions of Python2 and Python3 installed.

- This code and the install scripts can be safely used inside a conda environment.

- For the version of this script distributed internally to the IAC, there will be an additional three files. These are named GOTOinfo.txt, GOTOrsync.txt and rsync.pwd. These files are required by the software and they contain logon information for the GOTO Photometric database and the download server for image files.

- If these files are not provided, instructions will be given for their creation in the next three sections of this document. The specific database information will not be provided and must be requested from a member of the GOTO project.

- Figure 1 demonstrates the directory of a freshly installed copy of this software.

Creating GOTOinfo.txt

- The GOTOinfo.txt file is a text file which contains five lines indicating parameters required by the get_lightcurves.py script to connect to the GOTO Photometric database.

- It is recommended that this file be provided by an authorised member of GOTO as collaboration membership is required for the use of the data.

- If this file cannot be provided but you have been authorised to access the data, the file can be constructed as follows.

- The first line is the database name used by the Photometric database.

- The second line is the port number the database is hosted on. This is an integer value.

- The third line is the username required for access to the database.

- The fourth line is the password of the above username.

- The fifth and final line is the address of the server hosting the database.

- Any flaws with this file will trigger an appropriate error in the get_lightcurves.py script allowing the user to debug any problems.

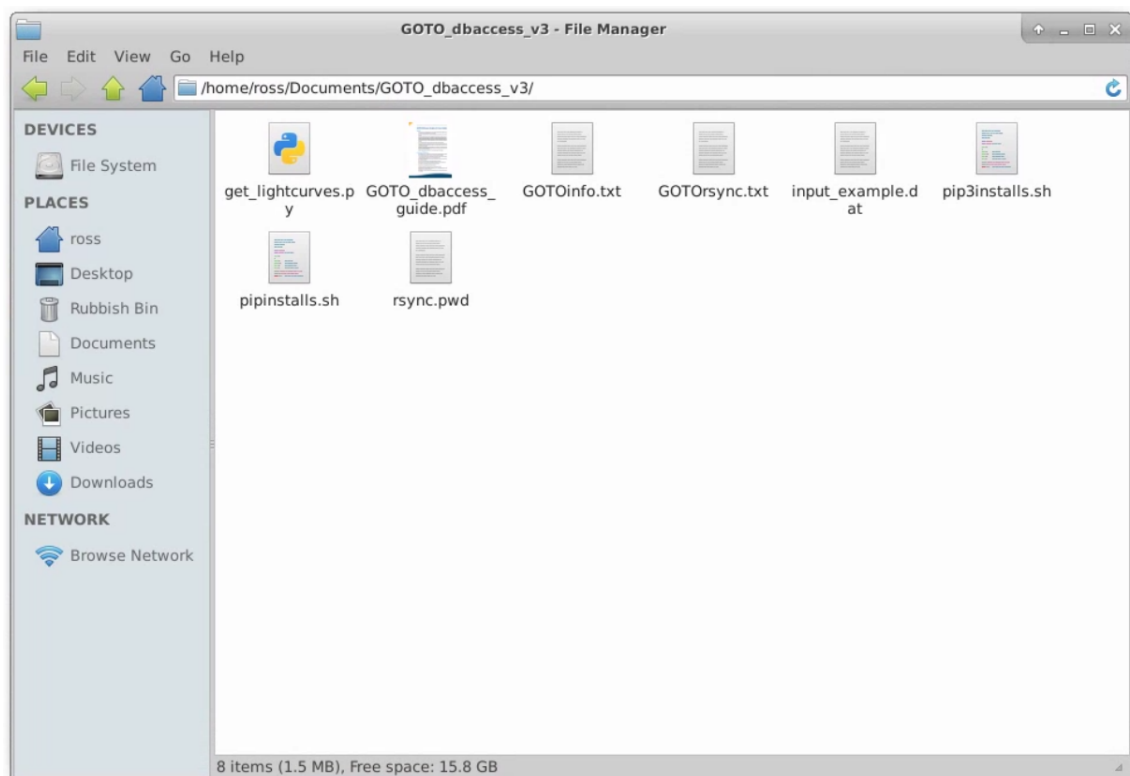- Once this file is correctly constructed, the get_lightcurves.py script will be able to query the database.

Figure 1: An example of the directory of a fresh install containing the GOTOinfo.txt, GOTO-rysnc.txt and rsync.pwd files in a Thunar file manager window.

Creating GOTOrsync.txt

- The GOTOrsync.txt file is a text file which contains three lines indicating parameters required by the get_lightcurves.py script to connect to the GOTO file server for RSYNC file download.

- As with the GOTOinfo.txt, it is recommended that this file be provided by an authorised member of GOTO as collaboration membership is required for the use of the data.

- If this file cannot be provided but you have been authorised to access the data, the file can be constructed as follows.

- The first line is the username used by the file server for data access.

- The second line is the address of the server hosting the GOTO image files.

- The third and final line is the port number the database is hosted on. This is an integer value.

- Any flaws with this file will trigger an appropriate error in the get_lightcurves.py script allowing the user to debug any problems.

- Once this file is correctly constructed, the get_lightcurves.py script will be able to download fits images from the GOTO file server when combined with the rsync.pwd file described in the next section.

Creating rsync.pwd

- The rsync.pwd file is a text file which contains a single line with the password of the username in GOTOrsync.txt for the file server required by the get_lightcurves.py script to connect to the GOTO file server for RSYNC file download.

- The file can be constructed as follows.

- On the terminal, navigate to the folder containing the get_lightcurves.py (as well as the supplementary files described in this document).

- Type 'echo "<PASSWORD>" > rsync.pwd' where <PASSWORD> is the appropriate password for the username in the GOTOrsync.txt file and press enter.

- Then type 'chmod 600 rsync.pwd' to allow get_lightcurves.py script read access to the password file.

- If you do not have access to the password, contact an appropriate member of the GOTO collaboration.

Creating a target catalogue data file

- Unless an existing directory of light curves is provided to --input, with the --existing argument present, a target catalogue data file is required to provide input information to the multi cone-search method in the get_lightcurves.py script.

- This target catalogue file is very simple and contains one of more rows. Each row is a target and has four values separated by spaces.

- The first value is a name for the target which will be used in the naming of the output light curve file. Do not repeat the same name multiple times as this will overwrite the previous light curve with no overwrite checking in this case.

- The second value is the Right Ascension coordinates of the target in decimal degrees. The Hours/Minutes/Seconds or Degrees/Arcminutes/Arcseconds units are not currently supported.

- The third value is the Declination coordinates of the target in decimal degrees. The Degrees/Arcminutes/Arcseconds unit is not currently supported.

- The fourth and final value is a cone-search radius around the RA and DEC coordinates in units of arcseconds. The option has been provided in this file instead of an input argument to the script in the event that different cone-search radii are required for each target object. NOTE : Do not make these cone-search radii too big or it will slow down the script significantly and result in multiple contamination events from nearby sources.

- Please inspect the input_example.dat file in the installation directory for an example of how to format your target catalogue files.

Using the get_lightcurves.py script on command line

- The get_lightcurves.py script is designed for a command line interface in a terminal window. Open the terminal window and navigate to the directory in which the script files are located. It is not necessary to run from this directory and if you are comfortable with accessing programs from command line, feel free to use the script as required. This example will continue with the assumption that you have navigated to this directory.

- The script is called with the command 'python get_lightcurves.py' command. The script has a number of arguments and some are required for successful operation.

- The following options are input commands for the get_lightcurves.py script and are input using a double hyphen followed by the name of the input (with no space) then a space and the value.

- For example: 'python get_lightcurves.py --input input_example.dat' which is the command to provide an input catalogue of targets for the script.

- If an argument is missing it may result in the program exiting during runtime. The error message will indicate which argument is missing so that the user may fix this. Required arguments are noted in the options below.

- **ARGUMENTS:**

- --input : REQUIRED == The path to either a file containing a catalogue of targets with the format shown in input_example.dat or a directory containing the light curves of a previous operation of the script. EXAMPLE : --input input_example.dat

- --output : The path to an output directory for light curves files generated during the operation of the script on this call. This argument is not used if an existing input directory is used. It defaults to a new folder in the directory the script is called from named 'light_curves'. EXAMPLE : --output light_curves

- --existing : This is a flag which does not take a value. If used as shown in the example, the script uses --input as a directory of existing light curves otherwise the script assumes the --input target is a new catalogue file. When used in this mode, many of the arguments are disabled as it is designed purely to rapidly add new detections and/or non-detections to the existing light curves by querying from the most recent timestamp in the existing light curve files. EXAMPLE : --input light_curves --existing vs --input input_example.dat

- --nondet : Another flag argument like --existing. Natively, this script only queries for photometric detections within a cone-search of the target coordinates. However, the user can request that the script perform a polygon search on all GOTO images in the database to determine if the target coordinates were imaged but the photometric pipeline did not detect a source. By supplying the --nondet flag, the software will perform the polygon search which will significantly slow down the speed of the script but it will also return all non-detections.

- --hashhead : Another flag argument. This adds a hash to the 'column names' row of the output light curve files. Depending on the software the user intends to perform further analysis of the light curves, the user may want to have the column names treated as a commented row.

- --startdate : The script allows the user to define a start date in Modified Julian Date (MJD) for the light curve queries. Has the value of 0 MJD as default which covers (easily) the initial date of GOTO observations. EXAMPLE : --startdate 58800 which will result in no observations, including detections and non-detection prior to 58800 MJD being returned.

- --enddate : Similar to --startdate but for the last date in Modified Julian Date to be queried. Has a default value of 66154 which is the 1st of January 2040, well after the GOTO project is expected to end. If you happen to be using this software after 2040 and GOTO is still in operation, you will need an update!

- --quality : The GOTO images have a quality flag between 0 and 255 depending on potential issues with the image reduction and photometric pipeline where 0 is perfect and 255 is extremely poor quality. The user can define to refine the quality requirements of the returned photometry through the use of this flag. EXAMPLE : --quality 128 – Quality flags better than 128 will include all medium and high quality data but discard low quality data.

- --imgid : This is a flag argument. Providing this argument will retain the img id column in the returned light curve files. This column can be used for debugging issues with the returned light curves or for obtaining the image fits files.

- --filepath : Another flag argument. Similar to --imgid, this argument adds a file path column to the output light curves which can be used for access the image data. Both the (img)id and filepath columns are important for the automated downloading of the fits image files using the --getframes argument described below.

- --noowchk : Another flag argument, short for 'no overwrite check'. By default, if this script detects that it may overwrite an existing file or folder, it will prompt the user to authorise this process with a Y/N keyboard input. If the user is **confident** that they do not require this check for a given operation and wish it to run autonomously, this argument should be supplied to bypass the overwrite check.

- --getframes : Another flag argument. This flag requests that the image file paths from the photometric database be used to attempt to download the science frame fits files from the GOTO file server. If --existing is used, it will only attempt to download frames from the current run. If you require files from previous runs, please rerun the script with appropriate start and end dates without the --existing flag. The downloaded science frames will be saved into sub-directories named after the target name in a 'frames' folder inside the supplied output folder. As each GOTO science frame is around 100 MB in size, the script will prompt the user for a Y/N keyboard input for every target in a given run. The script will inform you of the number of files to be downloaded and it is recommended to only access at most a few GB of files. If you need more specific data downloads, manually access the file server through a web portal. Ask a member of the GOTO collaboration for more information on this. In the event a run must be aborted, use the 'Ctrl-C' shortcut to cancel the downloads but note, you will need to press 'Ctrl-C' for EACH download.
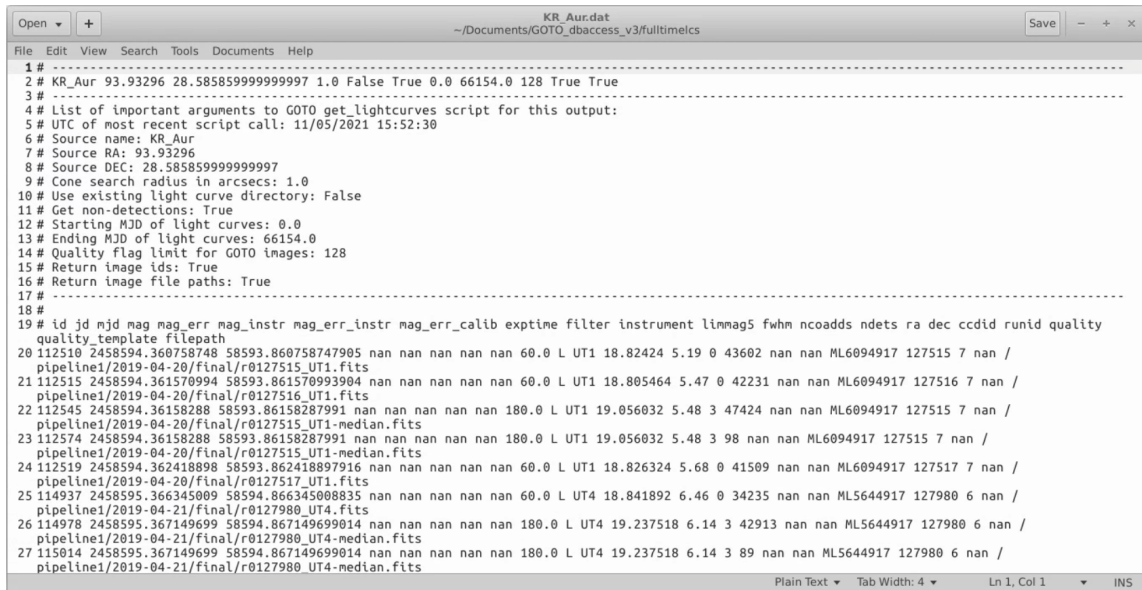
- --noszchk : Another flag argument, short for 'no size check'. Using this flag will by-pass the prompt to authorise downloading the science frames with a Y/N keyboard input. If the user is **confident** that they wish to automate the script such as if it is being used for a nightly query and will only download < 1 GB of frames per target per day, this argument should be supplied to bypass the file download size check. Note: Use of this flag is **NOT RECOMMENDED** for any operation other than the specific one described here, and even then, only if the number of targets is small. Heavy use of the GOTO file server will result in access to this resource being withdrawn.

Outputs of get_lightcurves.py

- This software outputs light curve files into the target --output directory, the existing --input directory or the default 'light_curves' folder depending on the input arguments.

- There can be multiple output light curve files with the naming convention of <target_name>.dat depending on the input of the script. If a catalogue file is used, the target name is provided by the first column of this file and there will be a light curve for each target. WARNING : objects with identical names may be overwritten.

- If the input is a directory of existing light curves, the output will contain the same number of files as the input directory but each light curve will have any new potential detections/non-detections appended to the bottom of the files.

- Each light curve file contains a header with information designed for the operation of the script and a user-readable format.

- This header records: the UTC timestamp of the most recent script call on this light curve, the source name, the source Right Ascension in degrees , the source Declination in degrees, the cone search radius in arcseconds, if the light curve used an existing directory as input (--existing mode), if the --nondet argument was used, the start and end date of the light curves in MJD, the quality flag limit and if the (img)id and filepath columns are returned.

- Below the header, there is a 'column names' row which by default is uncommented unless the --hashhead command is requested, then it has been commented with a '#'.

- Below the column names, the machine readable light curves are stored. Figure 2 shows an example of an output light curve used on the target KR Aurigae from the input_example.dat file using the following script call: 'python get_lightcurves.py --input_example.dat --nondet --hashhead --quality 128 --filepath --imgid --output fulltimelcs

- NOTE : If running in the --existing mode, many of the input arguments have been read from the header of the existing files and used in the new run, overwriting any user supplied arguments except for --output. It is extremely important the header is not altered by the user if the user intends to use the light curve in this mode as it will result in a program exception.

- Science frames requested by --getframes will be saved into a sub-directory in the --output directory named 'frames'. The frames belonging to individual targets will be again split into named sub-directories within 'frames'.
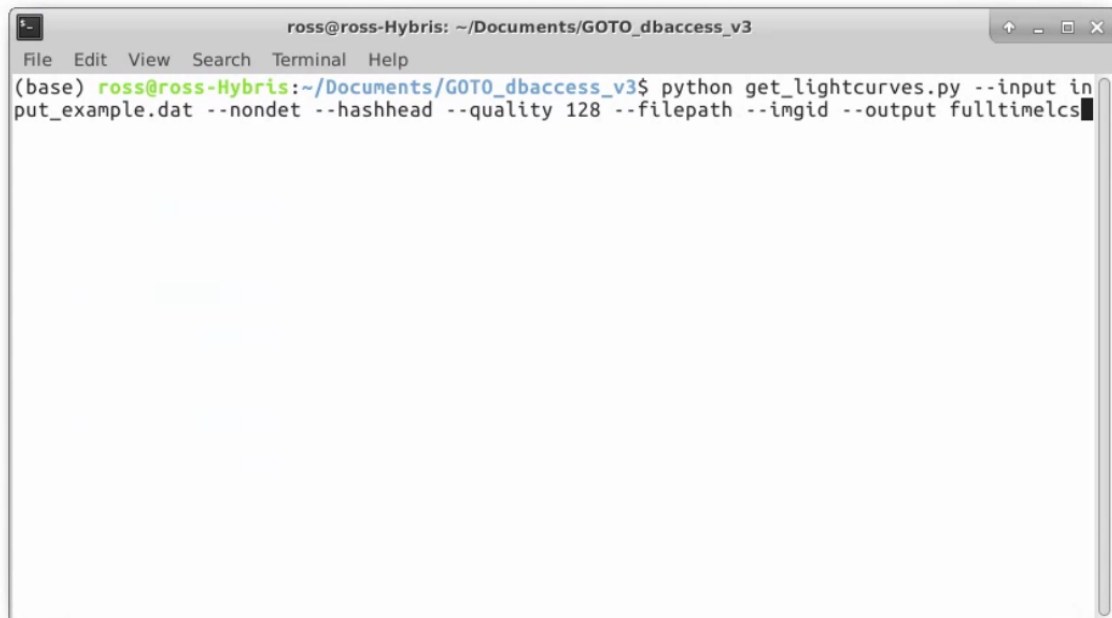
Runtime example

- In this section we will show an example of re-running the script call shown in the above section with the existing output present at the output location.

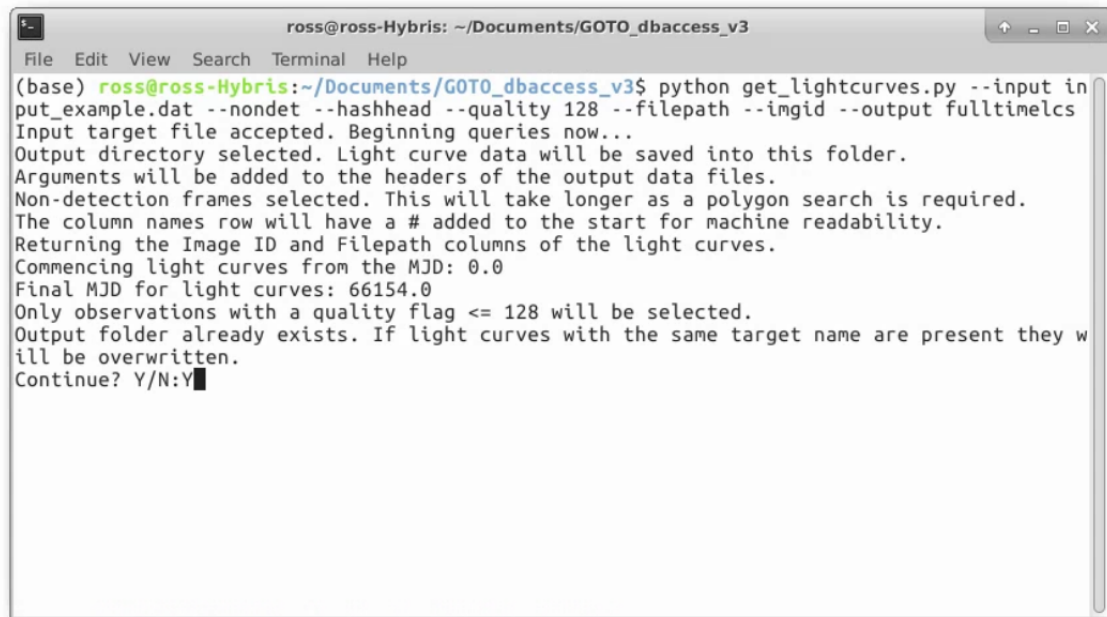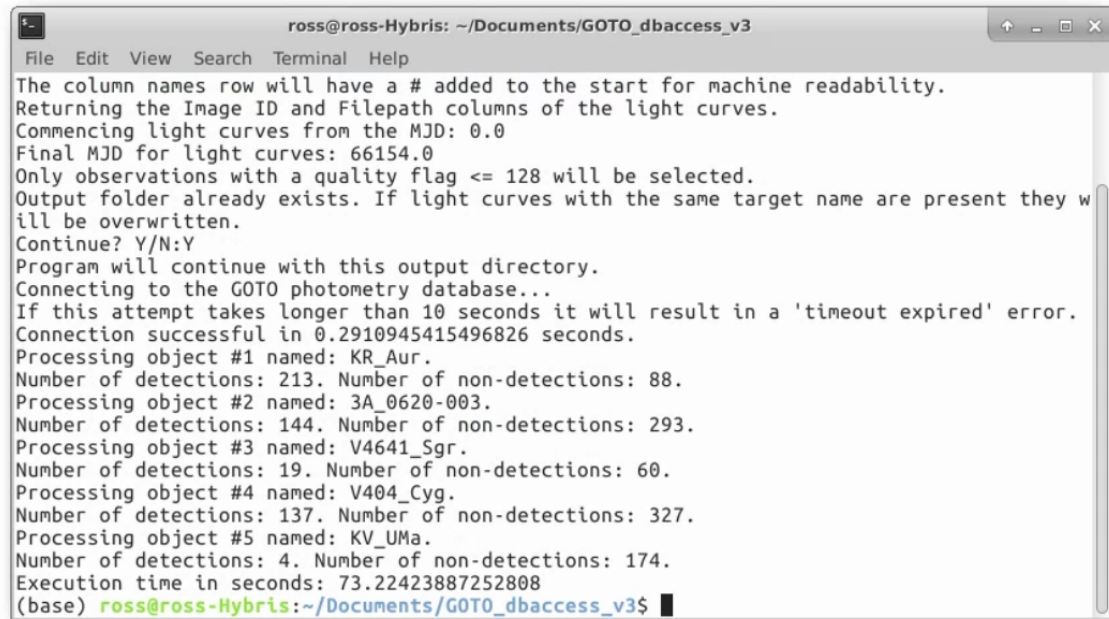- First, navigate to the directory containing the get_lightcurves.py script file.

Figure 2: An example light curve output file from the program call described in the 'Outputs of get_lightcurves.py' section.

Figure 3: The command used in this example as seen in a terminal window on the author's laptop.

Figure 4: The overwrite protection triggered by selecting an existing output folder for the light curves.

- Execute the command shown above and demonstrated in figure 3.

- In this example we are requesting a multi cone-search on the five targets in the input_example.dat file. We want non-detections returned with the column names row commented out, a quality limit of 128, the filepath and imgid columns returned and all light curves saved into the fulltimelcs directory.

- Figure 4 demonstrates the initial output on the command line. We can see that the target file was accepted (the format was correct) and that an output directory has been defined. We are informed that we have requested non-detections and the other options we input at script call. As we did not input a start or end date, the defaults of 0 and 66154 in MJD have been used.

- The first thing we encounter is an overwrite check. As stated above, we are re-running this command so the output directory exists and the script is requesting verification that we wish to overwrite this directory. If we wish to avoid this check in the future we need to call the script with --noowchk. However, in this case, we type Y (or y, it is not case sensitive) to continue. We could also exit the script with N (n).

- With the overwrite check complete, the script will then attempt to connect to the GOTO database. Any issues with the GOTOinfo.txt file will be highlighted here if the connection fails or times-out. Hopefully, if everything is fine and the GOTO database is available, you should see a connection successful message in under a second.

- The script will then begin the multi cone-search procedure with the non-detection polygon search enabled. The script will print out which object it is currently processing and when complete it will inform the user of the number of detections and the number of non-detections.

- The light curves are then written to files in the output directory (as shown in figure 2) and the program will exit with a note of the runtime in seconds.

Figure 5: The script connects to the GOTO database, runs the multi-cone search, saves the output light curve files and exits in 73 seconds.

- These outputs can be seen in figure 5. If the user then navigates to the defined output directory (or the default of light_curves in the current workspace directory), they can find the saved light curve files.

- This example does not show the --existing mode used to get new observations of existing light curves. The process is similar and the script will output instructions on the command line to inform and guide you through the process.

- This example also does not show the --getframes mode used to download the science frames. This process is reasonably simple and can be triggered by just applying the --getframes argument when calling the script. The command line interface will then instruct you on any other input required to download the science frames.

- Remember, if using this script to produce an automated process, include the --noowchk and/or the --noszchk arguments as appropriate or the process will wait for a user input. Again a reminder, take great care if using --noszchk to not bombard the file server (and your hard drive) with download requests.

Final remarks

- Following this guide and addressing error messages from the operation of this software should be sufficient to mitigate any major issues.

- There are no more versions of this software currently planned but if you encounter an issue please contact me via email (if you know me!) or via Github issues. I will be uploading this onto my Github account at https://github.com/P-R-McWhirter and issues can be created from this location.

- Take care and I hope this software is useful! Ross McWhirter, 11th May 2021.