Final Report of the Tutored Research and Development Project II Model-Free Target Tracking

Frances Ryan, Eduardo Daniel Bravo Solis

***Abstract*—An important part of research and development in the area of model free target tracking is the evaluation of your tracker. During development it is useful to understand how your tracker is performing on a particular dataset, where it may have issues and how it can be improved. Although certain benchmarks are openly available such as the Visual Object Tracking (VOT) toolkit, these benchmarks are usually limited in the information they provide and often require a cumbersome installation and setup. In this work we develop a Python-based evaluator with a user-friendly interface allowing customization of the analysis. The evaluator allows the comparison of multiple trackers and multiple sequences using groundtruth and output data. The interface allows the user either to perform a wide ranging analysis comparing their tracker to state of the art tracker results available from VOT challenges or perform a more detailed analysis on specific sequences or challenges. This higher level of customization makes the tool ideal for use during prototyping and development stages. A low performing frame search functionality allows a search for where the tracker is performing particularly poorly in terms of overlap which can be useful in flagging any issues or flaws in the design. The application also allows the immediate visualization of results in html format as well as the possibility to generate a Latex output. The metrics calculations in our evaluator were found to be reliable by comparison with those of the Online Tracking Benchmark(OTB) and the VOT benchmark. We were able to successfully achieve a working evaluator meeting all of the functional requirements and many of the non-functional requirements we outlined on beginning this part of the project.**

I. Introduction

The goal of single object tracking is to track some object of interest throughout a sequence of video frames, overcoming challenges such as occlusion, illumination changes and changes to object appearance. Model-free or generic object trackers must track an unknown object based on an initial bounding box provided in the first frame. In recent years, convolutional neural networks have been quite successfully applied to the task of object tracking. During the first part of the project we were able to gain an understanding of CNNs, study some of the key architectures and how they may be applied to tracking. We then reviewed some particularly important deep-learning based methods applied to tracking and were able to compare some of these methods using the Visual Object Tracking toolkit. Evaluators such as VOT are hugely important to provide a standardized and repeatable way to evaluate new trackers and demonstrate their contribution to the field. The VOT toolkit is implemented in Matlab and allows the running of a tracker according to a protocol along with the evaluation of the tracker and comparison to other state of the art and baseline tracker results. Although toolkits such as VOT provide an essential function and are designed to perform as reliably as possible, their ease of use and convenience in terms of setup and installation is a major disadvantage. Particularly since many research groups are moving toward the use of more accessible languages such as Python for the design of trackers, therefore evaluator implementations in Matlab can be inconvenient especially at the prototyping and testing stages which may not require the same level of detail as these benchmark evaluators. Therefore, in the second part of our project we have aimed at designing a more accessible, user-friendly evaluator executed in Python. Our evaluator allows the assessment of a trackers results entirely in Python providing the calculation of various important performance metrics and allows comparison with the results of other trackers. The main objectives of the second part of our research project can be summarized in three main tasks:

1) Implement an evaluator in Python for the assessment and comparison of tracker performance
2) Design a user interface that will allow the user to customize the type of evaluation required
3) Allow the exporting of plots and tables in a convenient format

In Section II we provide a brief outline of some of the relevant literature for the work. Section III provides a detailed description of the functional and non-functional requirements of the project. Section IV provides an outline of how we approached the project and planning. Sections V and VI provide a detailed description of how we designed and implemented the evaluator. Section VII contains a description of how we verified and tested our implementation. Section VIII provides some examples of the evaluator output. Finally, the conclusions we have arrived to from this element of the research project are discussed in Section IX.

II. State of the Art

A. Existing Evaluators

There are several evaluators that claim to be easy-to-use, available on the web [1], but most of them are Matlab-based. This is a software that is losing popularity for user-friendly applications, since the user is required to purchase a licence and go through a long installation process before use. Other python-based approaches for tracker evaluators can be found online [2] and [3]. However, they do not offer results that are similar to the VOT, where the metrics are clear and easily comparable with other algorithms.

Furthermore, evaluators such as VOT and Online Tracking Benchmark(OTB) are designed to act as benchmarks with the very specific aim of providing a method of running trackers in the same way so they can be compared accurately. However, this means that often they provide a limited set of statistics for trackers without much flexibility.

B. Software selection

Several frameworks offer the possibility of developing graphic interfaces for the user but few have the flexibility of working in multiple operating systems, being free of charge and easy to use. Python [10] is an interpreted programming language which has been earning popularity in the last decade due to its simplicity and versatility allowing everything from fast prototyping to rigorous software. Being a free and open source option, it offers the possibility of having a fast installation process for the final user and information online to help at the development stage. For the graphic interface libraries, there are different free options available online. A break-down of the advantages and disadvantages can be found in table I.

Based on table I, PyQT5 is the strongest choice. Being a popular library it guarantees a larger amount of documentation that could ease the coding process. Having more universal tools is always preferable for developing and maintaining the software. In addition, compared with all the other options, this library is one of the easiest to install and debug.

C. Metrics for Evaluation

In order to design an evaluator for trackers, it is important to include the calculation of meaningful metrics. There are a wide variety used in visual tracking, with some being more popular than others. The following section outlines some of the possible metrics that could be included in an evaluator.

Center location error (CLE) is one of the oldest performance measures but is still quite popular due to its simplicity and the fact that it requires only one point per frame to perform comparison. It is a measure of the difference between the target's predicted center and the center of the ground-truth, equation 5, where X_G is the groundtruth center and X_T is the tracker error. The measure may be summarized as an average or root mean square error. In certain cases a normalized center error has been used where the CLE is then divided by the detected target size [4]. In other cases, to make the measure more meaningful the measure has been expressed as a precision plot showing the percentage of frames which are within a given threshold distance from ground-truth [6].

The region overlap measure, considered as a measure of tracker accuracy has become highly popular as it accounts for both the detected region size and position. This is simply the intersection between the predicted and ground truth region divided by the total areas as shown in equation 1. This measure is sometimes seen as analogous to the F-measure in information retrieval 2TP/(2TP + FN + FP) [4]. The overlap measure is usually averaged across the entire sequence

A measure occasionally used is tracking length which is simply the number of correctly tracked frames to the trackers first failure, different failure criterions have included visual inspection which can be highly biased or a threshold on the CLE or overlap measure. However, a major drawback was that it used only the first part of the video sequence which means disregarding significant amounts of data. The failure rate, commonly interpreted as a robustness measure, addresses this issue and requires the reinitialization of the tracker after a failure, it is then simply the number of failures within the sequence or set of sequences, see equation 2 where $\mathcal{F}_\tau$ is the set of all failure for frame numbers f_i and F_τ is the number of failures in a given sequence or set of sequences. Again, a failure here may be decided based on some threshold e.g zero overlap. [7] One recognized drawback of this measure was that it doesn't reflect the distribution of failures within a sequence so the measure has sometimes been further interpreted to provide a fragmentation measure – which may be used alongside the failure rate to measure whether failures are uniformly distributed across a set of sequences. The fragmentation measure equation is shown in equation 3 where, again, F_τ is the number of failures and f_i denotes the position of the i-th failure and N is the total number of frames assessed. Furthermore, as is the case for the VOT results the failure rate is converted to a reliability measure for easier visualization, see equation 4 where again the notation is as above. Except that here S is some number of frames such that the measure may be interpreted as the probability that the tracker will successfully track S frames. [4]

Another measure, which assesses overall performance, is expected average overlap (EAO). This metric was introduced in VOT2015 and is basically the average overlap (with the ground-truth) on a large set of sequences for different possible sequence lengths. VOT make use of their reset-protocol to calculate this measure - by taking segments of the sequence in between failures. For example taking segments of around length N_s(sequences may be padded with zero overlaps or trimmed to the appropriate length) the average overlap is computed for this length and the value reported for this length sequence is the average across many segments of this length. In 6 N_s is the interval N_{lo}:N_{hi} and Φ_{Ns} is the average overlap on the sequence of length N_s. To obtain the expected average overlap this measure averaged for a large set of N_s long sequences. Furthermore, it can be presented as a curve by evaluating this for a range of N_s = 1:N_{max}. The final ranking for the VOT benchmark produces a single EAO for each tracker by further averaging the value across all the sequence lengths. [7]

A summary of the various measures and metrics is provided in table II. The final implementation of our evaluator included the calculation of the first four metrics in this table the EAO was not calculated as it is more complex and highly dependent on the VOT reset protocol.

III. Requirements

A. Functional Requirements

The main functional requirements of our software were defined as follows:

- The evaluator should output relevant metrics on tracker performance
- A user interface should allow some customization of the evaluation in term of metrics and sequences to use in the analysis
- The evaluator should allow the comparison of multiple tracker's results
- The output metrics should be clearly tabulated and graphed

TABLE I: Python libraries for visual interface - table outlining the assessment of various possible options for creating user interface

Framework	Pros	Cons	Installation	Conclusion
PyQT5	Can be used with PyQt5 designer, easy handling of widgets and events, signals and slots system easy to use	Not the most intuitive library for coding widgets	PYQT5 with pip install, pyQt5 designer can also be installed with pip	Seems to have a good balance between simplicity and functionality, somewhat familiar from using with C++ in Acquisition
Tkinter	May not need separate installation as is usually with python, seems to have a lot of documentation	Quite basic widgets, limited	Since it may already come with python, can work with just importing otherwise may need to install tcl/tk and change python setup to recognize paths to installation	Functionalities are very basic
Wxpython	A lot of features and widgets, uses native widgets so widgets take the appearance of native applications. A lot of tutorials and documentation	Not as flexible as QT in terms of user control, uses callback (pointer to s a function)functions which can become complex depending on what's needed. More cross-platform adjustments required than others	Pip install or with conda/Anaconda	For this purpose, the native appearance of widgets is not so important.
Kivy	Good for interactive applications, versatile, high-performance	Documentation and examples limited, more suited to games development/graphical apps	Pip install some dependencies and Kivy	More graphically elaborate than needed
PyForms	Interfaces can be easily defined with short code, advanced functionalities with minimal effort.	One of the newer frameworks so seems like documentation and tutorials are scarce	Pip install, also requires QT4	Not enough documentation compared with previous options
PyGUI	Like Wxpython also uses native widgets so matches applications already on system, Simpler than pyQT or Kivy	Does not appear to have much documentation, fewer tutorials than some others	Download available online	The widgets seem to look nice and simple from the examples but also tutorials and documentation not enough

TABLE II: Description of different possible metrics to use for evaluating trackers, equations and notation described in detail in section II

Measure	Definition	Formula	Additional Notes
Region Overlap(Accuracy)	Overlap with the ground-truth (intersection over union)	$\phi = \frac{1}{N}\sum \frac{R_g \cap R_t}{R_g \cup R_t}$ (1)	Accuracy usually averaged across dataset, VOT take special approach to avoid bias where they don't assess accuracy for frames where tracker has failed
Failure Rate(Robustness)	How many times tracker fails. The rate of failure of the tracker.	$F_\tau = \lvert\mathcal{F}_\tau\rvert, \mathcal{F}_\tau = \{f_i\}$ (2)	Can choose some minimum overlap threshold instead of zero overlap also. Various approaches have been taken to use this measure in a more informative way - See following metrics Reliability and Fragmentation
Fragmentation	Related to failure rate - an informational measure of the distribution of failures in a sequence.	$\frac{1}{logF_\tau}\sum_F t\frac{-\triangle f_i}{N}\log\frac{\triangle f_i}{N}$ (3) $\triangle f_i = \begin{cases} f_{i+1}-f_i, & \text{when } f_i < \max(F_\tau) \\ f_1+N-f_i, & \text{when } f_i = \max(F_\tau) \end{cases}$	Maximum value of 1 reached when failures uniformly distributed through sequence and decreases as inter-failure intervals are more uneven
Reliability	Related to failure rate - allows easier visualization by providing an upper bound to the robustness-defines an exponential failure distribution	$R_s = \exp(-S\frac{F_\tau}{N})$ (4)	May be interpreted as the probability that the tracker will successfully track up to S frames since last failure. Used by VOT with S = 30
Precision(Center Location Error)	Average Euclidean Distance between the center location of tracked targets and ground-truth.	$\delta = \lVert X_G - X_T \rVert$ (5)	Usually expressed as precision plot – percentage of frames with estimate location within some chosen threshold distance from ground-truth center. Other times summarized as average or RMSE. Used in OTB.
Expected Average Overlap	Average overlap on a large set of sequences for different possible sequence lengths. Takes segments of sequences between failures	$\Phi = \frac{1}{N_{hi}-N_{lo}}\sum \Phi_{Ns}$ (6) $N_s = N_{lo} : N_{hi}$	Introduced in VOT 2015 – makes use of VOTs reset protocol

Aside from the minimum functional requirements that the software had to meet, we also defined some further desirable functionality that would make the evaluator more useful and unique. These were:

- The evaluator should allow the exporting of the graphs and tables from the analysis into a convenient format such as Word or Latex
- A detailed analysis option should be provided which outputs sequences/frames for which the tracker's performance was particularly poor.

B. Non-Functional Requirements

In addition to the functional requirements outlined in section III-A we also had some further constraints on the implementation and desirable features which we believed would be beneficial for the software.

- The evaluator should run in Python
- The software should be easy to install and to use to compare trackers.
- Minimize the use of external libraries to reduce installation and setup requirements
- User interface should be intuitive and clear
- Output from the evaluator should be aesthetic and easy to read

IV. Development process

To follow a proper software development process, several tools were used for the planning, development and testing stages. Although this information is not included in most papers, it was added in this version for justifying the robustness of the code and results.

- For time management and a semester-long planning we followed the Gantt chart shown in figure 1. This table went through several updates based on the weekly stand-up meetings. Activities are listed until the end of the semester but the team will consider to make further test and release version updates in the following years.
- In the scope of the SW development, we used GIT for local and GIT-HUB [8] for online version control. The code is public and can be downloaded directly from the platform [9]. Since early stages, the team performed peer review meetings. In this extent, the programmers were able to receive feedback on their coding standards or logic. This helped the members of the team to be aware of the latest updates and the current state of the software.
- For the testing part, the team performed simple user cases for the interface flow and visualization of the results. Before final release, a more rigorous verification is planned.

With all these tools, the workflow was smooth and stable, allowing to achieve goals more effectively compared to previous projects that did not use these methods. In addition, under this rigorous supervision the code is less prone to errors and can be develop in a modular and reusable way.

V. Software Architecture

Figure 2 provides an outline of the architecture and information flow used in the design of the application. On beginning the application an initialization of the interface occurs which retrieves information already available in the working directory which should be setup according to the README provided in the Github repository. The interface is then loaded and allows the user to select which sequence-tracker data to be evaluated and compared along with what metrics they want to output.

Once the user has selected the desired options these are taken to the data import stage. Here, based on the user-selection the required data is loaded from the user work space. Further details on how the data is loaded and stored is provided in Section VI. Once the data has been imported in an appropriate format it may then be sent to the evaluation stage - these functions perform the calculation of the metrics based on the imported results and ground truths. The output from the evaluation stage is then transferred to the visualization functions - these use matplotlib to carry out the plotting of the information for clear interpretation of the evaluation and comparisons. Additionally, a HTML report is generated providing an immediate view of the results to the user. The user may also include an option to export the results in Latex format.

Once an initial report has been generated the interface remains open so the user can select further analysis options if desired.

VI. Implementation

A. Datasets

The datasets initially used for prototyping the evaluator was data from the VOT benchmark. This data provides 60 sequences for testing a tracker. The VOT toolkit itself provides a method for running a tracker in a particular manner which, as discussed in Section I, then provides an easy platform for ranking and comparing state of the art trackers and this is what is done each year in the annual VOT challenge. After the publication of the results of the VOT challenge the top trackers and their corresponding result annotations are published online. These are readily available at [11]. The groundtruth results associated with the 60 sequences are also available with download of these sequences.

Furthermore, some results from the visual tracker benchmark, also known as the online tracking benchmark(OTB) were used for the verification of center location error which isn't calculated by VOT and also to reinforce the verification of the overlap.

B. Language and Libraries

The language used for the implementation of the software was Python3 - this language was chosen as it is easily accessible for researchers everywhere without having to buy a license. Additionally, python code can generally be written in a more concise manner when compared to languages such as Matlab which makes it easier to read and less error prone. Finally, Python allows the use of a wide variety of packages that allow

Task	SubTask	Week 35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
Review and redefine goals	Check previous work and define new goals																	
	Review state of the art																	
	Define functionalities																	
Prototyping	Assess types of metrics																	
	Evaluate of Python interfaces libraries																	
	Draft and prototyping																	
SW development	Developtment of metrics calculation																	
	Interface coding and test																	
	LaTeX and html visuals																	
Test and verification	Debugging and error handling																	
	Publish software																	
Documentation	Write final report																	
	Prepare Slides																	

Fig. 1: Gantt chart - States the time table for the development process

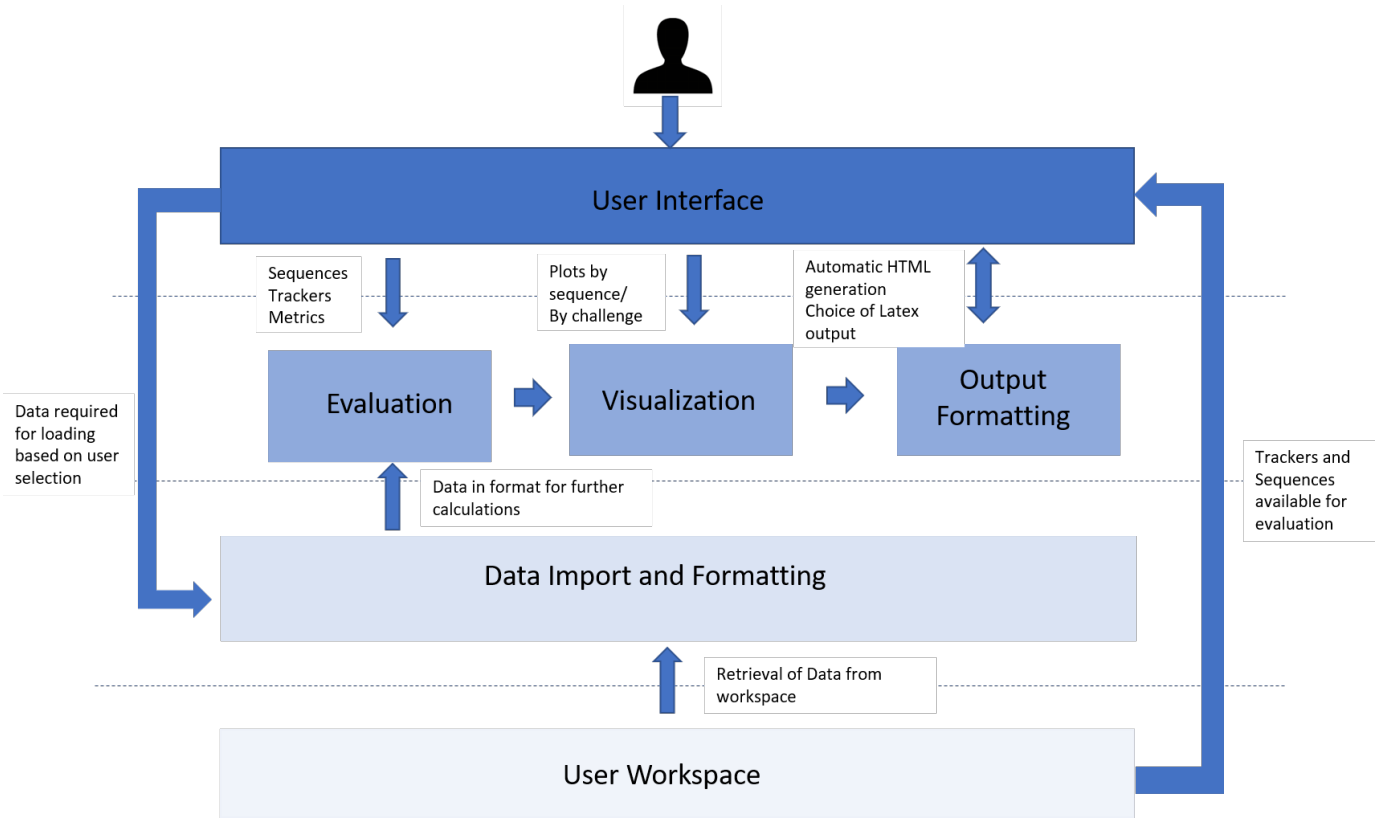

Fig. 2: Schematic of Software Architecture used for application

for high quality data visualization and representation, as well as useful libraries for performing efficient calculations and data handling. These packages are usually relatively easy to install and import for use in a Python module.

One important aim of the project was to try to use as few external libraries as possible to minimize the installation requirements and dependencies. The following libraries however were found to be of particular importance or provided efficient functions for some aspects of the application:

- Numpy
- Matplotlib
- PyQt5 - as discussed above this was the chosen library for implementation of the user interface.
- Shapely - the shapely library allows for detailed manipulation and analysis of geometric objects. It provides extremely efficient methods for measurement of metrics such as intersection and overlap.
- wget - this provides methods for retrieval and download of files online and is required if the user wants to download any of the VOT sequences and ground truths

without requiring the Matlab toolkit.
- pyhtgen - this allows for the generation of HTML code and was used as an accessible way for the user to immediately visualize the results

A couple of other libraries were also used but are used for optional functionalities so are not required by the user, these were:

- pylatex - this library is not compulsory but was used for the generation of a latex output for the results however since the user is free to select this option or not this library may not be required.
- scipy - for reading the .mat files storing some of the OTB results needed only if the user wants to use the option to download OTB data for comparisons

C. Interface

The figure 3 shows an example of test case of the interface. The list of trackers as well as the available sequences are variable and depend on the existing data in the corresponding file folders. It is recommended to add also results from well known trackers to have a more meaningful comparison when the results are displayed. The user is responsible for setting up this information before running the evaluator. When the program runs, it displays the interface where the user should determine the following configurations:

- Trackers - List of trackers available to include in the analysis. This window lists the name of the folders in the corresponding directory where each folder should contain the *.txt* file with bounding box prediction results.
- Evaluation Extra Features - Extra features of the evaluator includes a display of the least scored frames in the analysis per tracker and a functionality to export the results to LaTex format.
- Type of results - The output plots and tables can be configured to be sequence or challenge based.
- Sequences - Since the analysis is based on the VOT challenge [6], the list of sequences should be the same or similar to the ones in the official workbench. These should be stored in a folder named *sequences* and the interface will list them for the configuration.
- Metrics - Type of metrics to include in the analysis. Selecting both Accuracy and Robustness will produce an AR graph.
- Challenges - For a more particular evaluation, the user can select the challenge that they are most interested in. If the selected sequences does contain these challenges, a warning message will be displayed.

D. Data Import and Formatting

The evaluator should have the ability to compare several trackers on multiple sequences - this means that the loading and storage of data is of high importance so the remaining evaluation and calculations can be carried out as efficiently as possible. To make this possible the data is imported after the user selection is made from the interface which is based on the data present in the user's workspace as discussed in

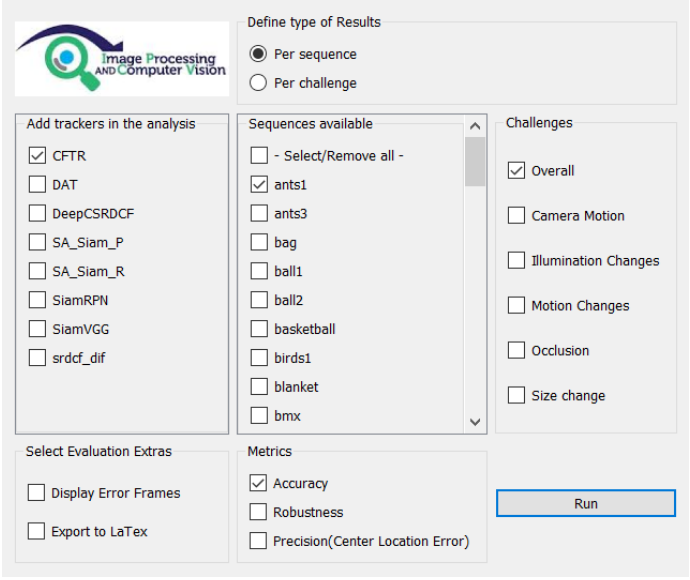

Fig. 3: Interface Layout - Example of a configuration, with several trackers and sequences available in the workspace.

Section V. The output results of the chosen trackers, data for the groundtruth for the sequences and data from the challenge tags are loaded and stored in nested dictionaries. Python dictionaries allow the storage of many different types of data of different dimensions. This was important as, for example, the results and groundtruth may not always have had the same dimensions some may provide rows of length 8 describing polygons whereas others would provide rows of length 4 describing bounding boxes. Dictionaries also allow very fast and clear access to different required datasets.

A potential problem foreseen for the VOT dataset was that the sequences along with their groundtruth and by-frame challenge information are not easily accessible but rather are downloaded on setup of the VOT toolkit in Matlab. This is since the sequence information is compressed in .json files, scripts to read these are then run on initialization of the toolkit to retrieve the sequences. Therefore, to ensure that use of the evaluator is independent of Matlab a function was written that allows the download of these sequences and groundtruth information for the VOT datasets for each year. An additional function was created which allows the download of tracker results data from the corresponding year. However, it should be noted that these results are easily downloadable as .zip files from the VOT website and the download process is quicker if the user downloads the corresponding .zip manually and places it in the directory of the evaluator. The function will then unzip and setup the tracker information in the required directory structure. A related issue arises with the visual tracker benchmark or OTB, here the sequences and groundtruths are available as.zip files however, the tracker results although they are available to download on their website the results are provided .mat files. Again, a function was created to download and read these files from the website and set them up in the correct directory format.

In order to align with the desired functional requirements

of the evaluator and to present this possibility in a simple, yet user-friendly way these download are provided if the no folder for trackers/sequences is present in the workspace or if these folders are empty. The option is provided in the form of a simple QMessageBox and dropdown menu as shown in figure 4.

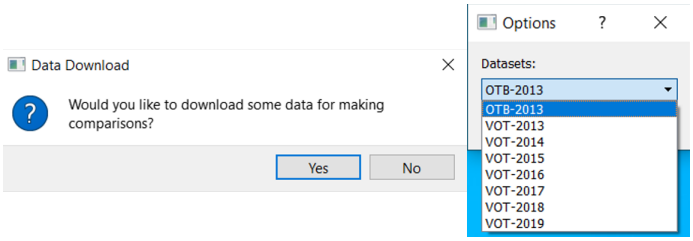

Fig. 4: Pop-up Message Box and drop down menu providing user with possible data download options for download of sequences, groundtruths and tracker results for making comparisons

E. Metrics Calculation

1) Accuracy and Robustness: As mentioned briefly in section VI the Shapely library was used as an efficient method for calculating the overlap. It was decided that it was not necessary to exactly replicate the method used in the VOT benchmark. This was because the method used here is quite complex and computationally expensive where a number of preprocessing steps are carried out on the groundtruths and results, including rasterizing the polygons - they then loop through the entire groundtruth and result polygon or bounding box and compare if the two objects are overlapping. Since the aim of our project is not to create an official benchmark but more as a useful tool for researchers this level of detail wasn't believed to be required once we could achieve similar ranking of trackers as that of the VOT benchmark. In our approach, we initially carry out a rounding of the results to simulate the rasterization approach from VOT this was found to produce results closer to that of the VOT benchmark. We then simply convert the groundtruth or result to a *Polygon* or *box* class from the shapely library which allows the calculation of the geometric intersection and union which can then be used to calculate the overlap. The results were found to match results from the online tracking benchmark very closely and to match the tracker accuracy ranking in the VOT toolkit. Further discussion on this testing is provided in VII. The average overlap is calculated in a similar way to the VOT benchmark, by averaging across the entire output from all sequences not including failures (where overlap was zero).

Robustness was calculated first by performing a failure count. The definition of failure was taken to be that of the VOT benchmark which is where the overlap measure goes to zero. The calculation of the final robustness value to be plotted was then calculated as per equation 4 with S = 30.

2) Center Location Error: The user is also provided an option to output the center location error(CLE). To calculate this, very similar information is required to that of the accuracy metric. Therefore, if selected, it is calculated alongside the accuracy by sending the output result and the corresponding groundtruths to the function. Again the shapely library is used to efficiently calculate the center of the polygon or bounding box provided using the *centroid* class. It is then simply a calculation of the Euclidean distance between the two points. After collecting the CLE over all frames in a sequence the average for the sequence is calculated, also an overall average is calculated over all frames assessed for each tracker.

F. Low Performing Frame Search

As mentioned in Section III, one of the desired functionalities of the tracker evaluator was to perform a search for the frames for which the tracker was performing particularly poorly. This allows an investigation into the areas which are bringing the tracker score down. This search mechanism is implemented for the overlap metric - so those frames which have the lowest overlap score. If 'Display Error Frames' is selected by the user on the interface (figure 3) this search is carried out by sending a list of the by-frame accuracy scores for each tracker, challenge and sequence to a function 'extract_min_frames' which then returns the accuracy score and index of the five lowest scoring frames(if 5 are available otherwise 1). These values are then stored in the form of a tuple along with the sequence name. The use of tuples in this case is extremely convenient as it allows searches within certain elements of the tuple while ensuring all the required information for later display of the frames stays together. Finally once this information has been gathered for all sequences a further search for the minimum 5 frames across all of the minimum frames collected from the sequences is carried out. The information about these frames are again stored in a tuple with the information about the sequence and frame index which can be sent to the output section in order to be displayed in the html report. The final display is done by using the information provided by the tuple - sequence name and frame number to load the associated image(if available in the user workspace) as well as accessing the originally loaded data dictionary to obtain the groundtruth and output information to plot the corresponding bounding boxes.

Tracker(Benchmark)	Ave Diff	Max Diff
VR(OTB)	1.031e-05	4.996e-05
TM(OTB)	1.422e-05	5.000 e-05
RS(OTB)	2.005e-05	5.000 e-05
PD(OTB)	1.826e-05	5.000 e-05
MS(OTB)	1.695e-05	5.000 e-05
DLSTpp(VOT)	0.01074	0.0982
DSiam(VOT)	0.0099	0.0968
ECO(VOT)	0.0118	0.1026
SiamVGG(VOT)	0.0106	0.1389
UPDT(VOT)	0.0088	0.0688

TABLE III: - overlap verification results for the tests on each of the different benchmarks OTB and VOT for each tracker, negligible differences were found between the OTB benchmark and our calculations but a larger difference for the VOT data showing our calculation methods align more closely with OTB overlap calculation

VII. Tests

A. Metrics Verification

In order to ensure that the metrics had been calculated and implemented correctly the results were compared to the sources discussed in Section VI. The accuracy was compared to results from the VOT benchmark and the OTB benchmark. In order to compare the overlap calculation frame by frame, 5 trackers and 5 sequences were selected from the two different benchmarks. From VOT these sequences were bag, fish1, graduate, matrix, singer2 and from OTB they were carDark, car4, david, david2 and sylvester. The trackers chosen from the VOT benchmark were picked to ensure they included trackers which output both polygons and bounding boxes. These were the trackers DLSTpp, DSiam and SiamVGG outputting rectangular bounding boxes and ECO and UPDT outputting polygonal bounding boxes. From OTB the trackers were VR, TM, RS, PD and MS. The 5 sequences from OTB totalled to 3000 frames and from VOT totalled to 1872 frames. Table III shows the verification results for the tests on each of the different benchmarks for each tracker. These were mainly the average absolute difference and the maximum difference overall for the overlap calculation for each frame taken from the reference benchmark(VOT or OTB) calculation results run in Matlab and the results we calculated in Python. It can be seen that for the comparison with the OTB results the difference is minimal and may be due to small numerical and rounding differences between Python and Matlab. The differences in the VOT data are slightly larger showing that there is some difference between the calculation method but we can see that the maximum difference is 0.14 which is reasonably small. To ensure there was no trend or bias in the differences an investigation of the distribution of the differences was carried out for the VOT data. This is shown in figure 5 for each of the tracker results used from VOT. The figure demonstrates that mainly the differences are normally distributed around zero with maybe a slight skew to a positive difference for the two trackers which output polygonal results - ECO and UPDT. This skew may be due to some differences in the handling of polygonal results from the different methods. To check if the method difference had a significant impact on the ranking of trackers a comparison was made to the tracker ranking based on Accuracy. The results of the comparison are shown in figure 7. In any case, it was visible that there are differences between the results of the overlap calculations performed in the VOT toolkit vs. the OTB toolkit which was expected as the VOT method uses what may be considered a more up to date approach. Our method aligns with that used by OTB and doesn't significantly change how the tracker would be ranked with VOT unless both trackers performed very closely.

The robustness and reliability results were verified in a similar way using data from VOT. Again the same set of 5 sequences and 5 trackers from VOT were used to verify both the failure count, failure rate and reliability data. These were found to match except in cases where the overlap calculation resulted in zero overlap when the VOT protocol didn't record a failure. so there are small variations, however, since the overlap

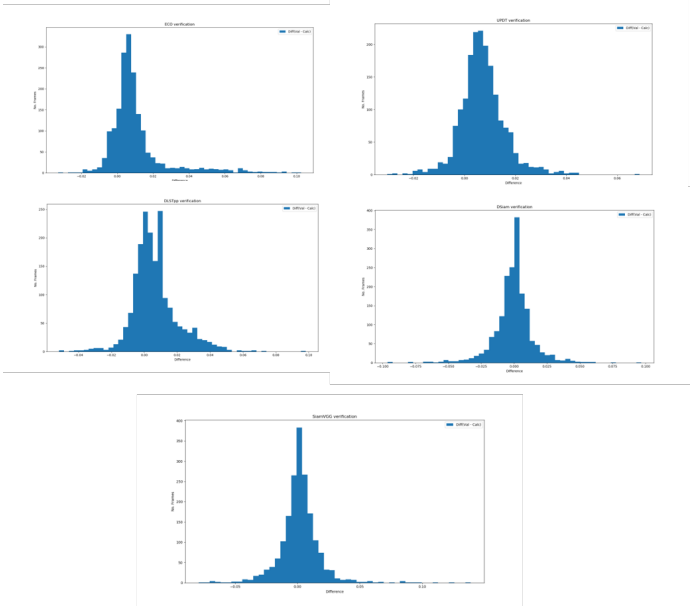

Fig. 5: Distributions showing differences between VOT reference and our calculations: can be seen that the differences are centered around zero and there was no consistent bias or pattern in the differences

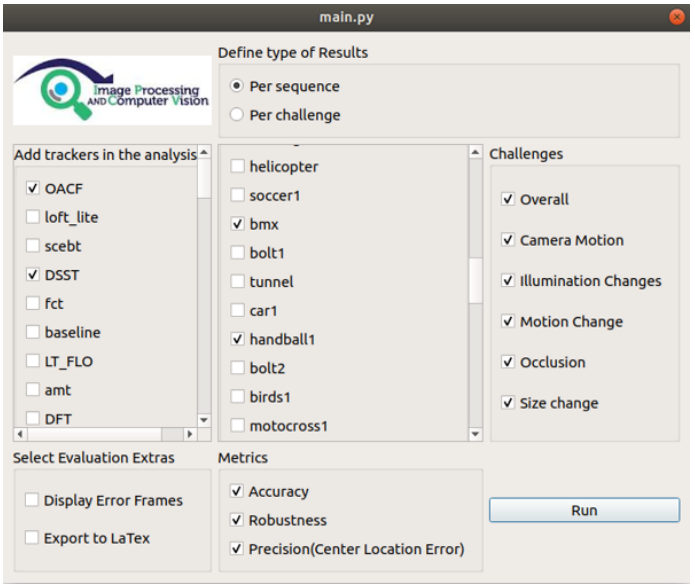

Fig. 6: Testing the interface on Ubuntu, PyQT widgets take the form of windows from the native OS

calculation was verified against the OTB this was reasonable. Similarly, the center location error calculation was checked against the OTB computations in the same way as described above and found to match without error.

One outstanding metric which was not verified was the fragmentation calculation derived from the failure rate. Unfortunately it wasn't possible to find data to compare this calculation with.

	Overall		Camera Motion		Illum Changes		Motion Change		Occlusion		Size Change	
	VOT	Ours	VOT	Ours	VOT	Ours	VOT	Ours	VOT	Ours	VOT	Ours
1	DLSTpp	DLSTpp	UPDT	UPDT	DLSTpp	DLSTpp	DLSTpp	DLSTpp	DLSTpp	DLSTpp	DLSTpp	DLSTpp
2	UPDT	UPDT	DLSTpp	DLSTpp	DSiam	DSiam	SiamVGG	SiamVGG	UPDT	UPDT	SiamVGG	SiamVGG
3	SiamVGG	SiamVGG	SiamVGG	SiamVGG	UPDT	UPDT	UPDT(0.5340)	DSiam(0.5287)	DSiam	DSiam	UPDT	UPDT
4	DSiam	DSiam	DSiam	DSiam	SiamVGG	SiamVGG	DSiam(0.5311)	UPDT(0.5282)	SiamVGG	SiamVGG	DSiam	DSiam
5	ECO	ECO	ECO	ECO	ECO	ECO	ECO	ECO	ECO	ECO	ECO	ECO

Fig. 7: Ranking of Trackers based on overlap from VOT vs. our method. The rankings differ in one case for the challenge motion change when the trackers score very similarly.

Challenge	Tracker	Accuracy	Robustness
Overall			
	CFTR	0.5034	0.9256
	DAT	0.4324	0.8055
	DeepCSRDCF	0.4869	0.9205
Camera Motion			
	CFTR	0.5282	0.9408
	DAT	0.4504	0.8927
	DeepCSRDCF	0.5077	0.9428

Fig. 8: Output results table from .html file

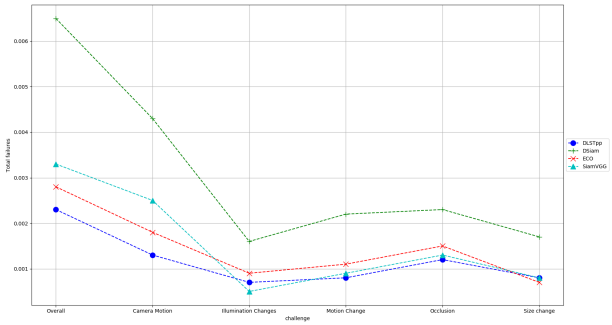

Fig. 9: Fail count graph - Displays the number of failures listing the challenge and overall results.

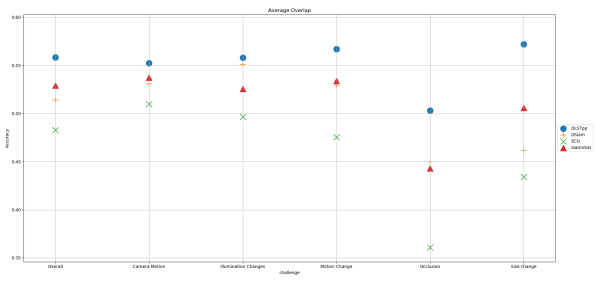

Fig. 10: Average Overlap - Bounding box overlap with reference ground truth along the challenges.

B. *Software Tests*

The application was tested for robustness with some of the different possible data sets to fix possible issues that may arise. Various use cases of the interface were tested:

1) Several randomly selected trackers, all sequences(from a given dataset), all metrics, all challenges - Results type: per challenge
2) Several different trackers from previous 10 sequences(max. possible for per sequence display), all metrics, all challenges - Results type: by sequence
3) Several different trackers, 10 different sequences, all metrics, all challenges - low frame search enabled.
4) Several trackers, all sequences, each metric one at a time separately
5) One tracker, one sequence, one metric, all challenges - by challenge and by sequence
6) One tracker, one sequence, one metric, one challenge - by challenge and by sequence

The above use cases were tested on Windows 10 for the datasets OTB, VOT2018, VOT2013 and on Linux Ubuntu 18.04(figure 6) for the dataset VOT 2015. The first cases were chosen since they test across alot of the data from the given dataset and so would show up any immediate issues that may occur with data loading and formatting. The final cases are those which use minimal data and test how the program handles having no data or minimal data present increasing the chance of empty lists and arrays.

VIII. Results

A. *Results format*

The program produces different types of results depending on the configuration, but mainly is divided in numeric tables and graphs that can be found in the generated *.html* file in the root location of the code.

1) Tables - The evaluator makes a list of the results, containing the trackers performance per each challenge selected (figure 8). The final table output format depends on the metrics selected by the user.

2) Graphs - If the Accuracy and Robustness were selected in the configuration, the system will produce the AR plot, comparing the trackers (Figure 12). Depending on the metrics selected, different graphs are provided. A failure rate graph is included for robustness showing the number of total failures over the total number of frames

assessed (Figure 9). An average overlap plot is also generated, indicating the percentage of bounding box overlap of the tracker with the ground truth throughout the selected challenges (Figure 10). A bar plot showing the center location error can be generated. Each of the graphs also allow the user the option to plot the results averaged over the challenges or if preferred averaged over each sequence and each challenge for up to 10 sequences (to maintain readability of the plots 10 was chosen). This functionality is implemented by selecting the 'By Sequence' option as shown in figure 3

3) Low Performing Frame Search - As discussed in section VI the low performing frame search picks out the lowest performing frames in terms of accuracy. In the *.html* file for each tracker and challenge the lowest five frames is shown along with the groundtruth and result bounding boxes. The sequence name and number is also provided. An example is shown in figure 11.

4) LaTeX report - The program generates a simple report of the results in LaTeX format (*.tex* file) that ease the process of exporting the results to a research scope. This option can be enabled/disabled by the user while configuring the analysis. The compilation of the *.tex* file is not considered on this implementation due to the amount of external dependencies needed for that purpose. For visualization of the results the user should refer to the *.html* file.

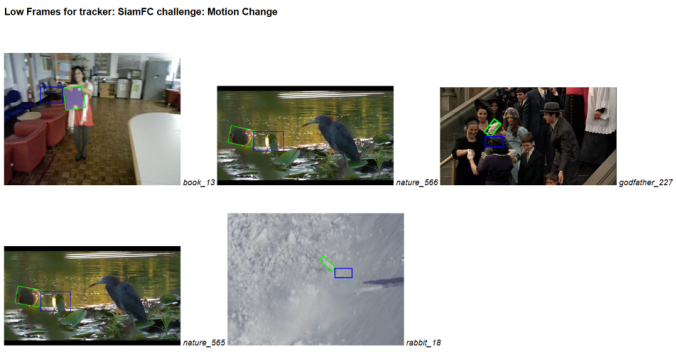

Fig. 11: Example Output in HTML report of low performing frame search

IX. Conclusions and Future Work

During this part of the TRDP we have achieved the design of a tracker evaluator in python which can act as an accessible and useful tool for researchers, particularly for prototyping and performing ablation studies when developing and improving new trackers. Our work during the first semester performing evaluations using the VOT benchmark allowed us to identify the difficulties with using tools such as these and hence, identify potential improvements we could apply. We succeeded in completing all of the mandatory and desired functional requirements outlined at the beginning of the semester and have stuck as closely as possible to the non-functional requirements.

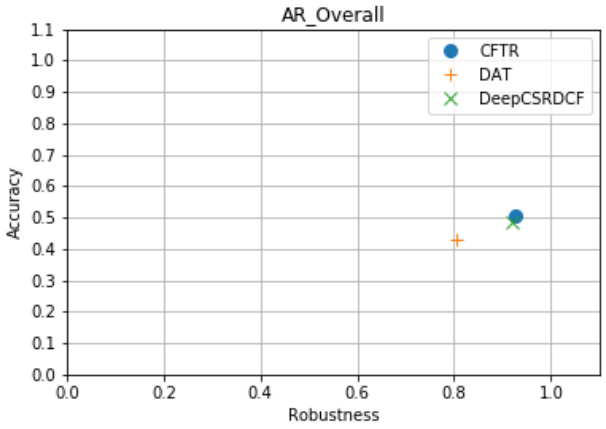

Fig. 12: AR Curve - Three trackers plotted in the AR space, overall analysis

In table IV, we have summarized how our evaluator compares to existing options.

	Ours	**Existing**
+	Implementation in Python with minimal requirements	Often implemented in Matlab which requires license
+	Flexibility of use with different datasets e.g VOT, OTB, custom, once you have groundtruth annotations and output results .txt file	Often tailored to work only with data generated on the specific benchmark e.g for OTB results tracker data must be stored in .mat files before comparison
+	Intuitive with user interface	Often require some manipulation of the code to set up analysis
+	Low performing frame search allowing more detailed information on where tracker is scoring low	Often only provides an overall result without a clear indication of how the tracker is failing
-	Doesn't allow the running of the tracker according to specific protocol	Benchmarks usually allow running of tracker in a certain routine to ensure fair comparisons e.g reset for VOT
-	Not rigorously tested, possible unforeseen cases causing errors	Have been used by many researchers and experts who have added suggestions and fixes

TABLE IV: Comparison of our evaluator with existing available options

Although the code at the time of release is stable and sufficient for a visual analysis of a tracker, there are still some improvement that could be done in order to make it a more complete and self-sufficient evaluator. Future work can be summarized in the following items:

- Perform a rigorous independent verification process of the code, where the requirements are listed and tested in detail.
- Implementation of a wider range of metrics and increased customizability of the visualization
- Addition of options allowing the user to vary and select different threshold values for metrics such as the failure count.
- Addition of extra features of the evaluation according to the public demands. After a period of a beta version release, collect the feedback from the users and update accordingly.

REFERENCES

[1] 'Unified Tracking Benchmark Toolkit', Accessed: 08 Dec. 2019, https://github.com/lukaswals/unified_tracking_benchmark

[2] 'Objective evaluation of tracker performance with the Multiple Object Tracking (MOT) measures', Accessed: 08 Dec. 2019, https://github.com/Videmo/pymot

[3] 'Benchmark multiple object trackers (MOT) in Python', Accessed: 08 Dec. 2019, https://github.com/cheind/py-motmetrics

[4] Luka Cehovin, Ales Leonardi, Matej Kristan, "Visual object tracking performance measures revisited", IEEE Transactions on Image Processing, 2016, vol. 25, no. 3, pp. 1261 - 1274, Jan. 2016

[5] Smeulders, Chu, Cucchiara, Calderara, Dehghan, Shah, "Visual Tracking: An Experimental Survey", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 36, no. 7, July 2014

[6] Yi Wu, Jongwoo Lim, Ming-Hsuan Yang, "Online Object Tracking: A Benchmark", 2013 IEEE Conference on Computer Vision and Pattern Recognition, June 2013

[7] Kristan et al., "The Visual Object Tracking VOT2015 Challenge Results" IEEE International Conference on Computer Vision Workshop (ICCVW), 2015

[8] https://github.com/

[9] https://github.com/edbs08/Tracking_Evaluator

[10] Guido Rossum, "Python Reference Manual" Technical Report, CWI, Amsterdam, The Netherlands, 1995

[11] "Visual Object Tracking", Accessed 08 Dec. 2019, http://www.votchallenge.net/howto/index.html