2022 Collin's Capstone Documentation

Table of Contents

Quick Notes:

- A Github version of the scripts can be found in: https://github.com/oneils2018/CollinsCapstone
- The Google Colab classification model script is here: https://drive.google.com/file/d/1IeBYIqjMfTfcQVr6pW-VIqHP-cFar_-D/view?usp=sharing
- The default classification model can be downloaded here: https://colab.research.google.com/drive/18P014j4ePPQMr0vTkuVf0Oyw_M81OtjJ?usp=sharing

How to run Python Scripts and use a Python IDE

How to Install Python and Run Scripts

In order to run Python scripts, the newest version of Python must be installed. Navigate to https://www.python.org/downloads/ to download the latest version, run the execution file, and follow the steps to install. Once Python is installed, Python scripts can simply be run by double clicking on the file.

How to install and use PyCharm (Python IDE)

PyCharm Installation:

1. Go to https://www.jetbrains.com/pycharm/
2. Click the blue or black download button. The website should take you to a download that is specific to your operating system.
3. Click the download button under "Community"
4. The installation should be like any other program. Just click the "Next >" button to move through each step.
    a. One thing to note is that creating a desktop icon is off by default.

Loading python scripts into a project:

1. Click on "New Project"
2. Leave everything on default besides the name if it needs changing.
3. Click "Create"
4. Locate the folder where the new PyCharm project is stored
5. copy any python script into the folder and the PyCharm project should update to show them.

Module installation (easy to install and manage):

1. Go to "File" on the top left.
2. Go to "Settings".
3. Click on "Project:" where to the left of the colon is the project name.
4. Click on "Python Interpreter"
5. Click on the plus "+" icon above the column name "Package"
6. Search which modules to add.
7. Click on the "Install Package" button at the bottom to start installation.

Modules

The easiest way to run and install required modules is by using an integrated development environment (IDE). The one used for example in this document is PyCharm Community Edition 2021.3. The other option is to use pip to manually install the modules.

Required modules for everything		
Module	**Purpose**	**Installation**
numpy	Popular module for high level array and matrices math problems.	PyCharm Python interpreter
math	Basic math module.	Built into Python
tkinter	Basic GUI module.	Built into Python
tktooltip	Allows for popup tooltips over tkinter GUI elements.	Command Line: pip install tkinter-tooltip
threading	Way to enable the use of multiple threads.	Built into Python
os	Way to interface with the operating system.	Built into Python
subprocess	Way to create subprocesses.	Built into Python
tensorflow	For Tensorflow operations.	PyCharm Python interpreter
keras	Used to process images.	PyCharm Python interpreter
matplotlib	For graphing purposes.	PyCharm Python interpreter
numba	Provides a way to convert sections of code to execute as if they were machine code. Increases performance.	PyCharm Python interpreter
dv	Used to read aedat files.	PyCharm Python interpreter
glob	Used to read files in a directory.	Built into Python
re	Used to make sure files are read in correct numerical order.	Built into Python
random	Random number generation.	Built into Python
time	Used to track how long processes take.	Built into Python

Note: When running the training script in google colab, the required modules for that script should automatically be loaded.

Brief Script Overview

The code within this program is organized as such:

- The **main.py** is the python script that runs the rest of code. It is responsible for loading the Davis camera recording, and then using functions defined in other files to filter the data and save the resulting images within the folder it is run in. It is a function so that the GUI can call on it but it can be run without the GUI if the function is called elsewhere.
- The **GUI.py** script is like main.py but is only needed when a graphical user interface is desired to edit and test different variables or files. It uses the **Variables.py** file to edit the variables that the rest of the scripts uses.
- Most functions you will see below have jitted_ in front of them. This is a function that has been passed to the numba library to convert them to machine code that makes them run much faster.
- If you are trying to follow the function calling path, go to the file name after the jitted_ within the folder that the main.py is located.
- Right now, the main.py includes a functions.py file, which contains a python script that includes all the other functions within the program.

The order in which functions are called within this file work as such:

1. Loads the aedat4 file -> only if it has not already be loaded in a previous loop.
2. Pass the data to the filter1 function and the filter10 function. Each output is stored separately
3. Results of both filters are independently sent to filter2. Filter2 is run multiple times, as each run through, a number of points are removed.
4. After going through filter2, both results are passed to a function called dual_pollarity_filter. This function takes points that are found in both the 0 and 1 polarities at the same x and y coordinates, and plots them in purple. Without this, all 0 polarity points at the same x and y coordinates will be completely overshadowed by the 1 polarity hits with the same coordinates. These points are stored in a third array, and the 2 different 1 and 0 hit polarity arrays remain unchanged.
5. The 0 and 1 polarity arrays are combined into a new array that is passed to the object_tracking function. This function draws boxes around the filtered objects detected. Note that within the object_tracking function is a call to another function called image_classifier. This function is called depending on a user defined variable make_test_data. It classifies objects based on a trained tensorflow model. If make_test_data is set to True, it will output the zoomed in images of objects found within the boxes. If False, it outputs the whole image with tracked objects having a predicted label above them.
6. Results are saved to the local directory of the main.py file. The program will loop through this process over and over until it reaches the end of the aedat4 file.

NOTE: Descriptions on what each function actually do can be found within their respective file.

Files table

In Folder File Name	Description
Filter1.py	This function takes the positive polarity data points from the Aedat file and filters it based on distances from a specific point and its distance from its neighbors, calculated using the X and Y values, as well as time in the form of a Z axis.
Fiilter10.py	This function takes the negative polarity data points from the Aedat file and filters it based on distances from a specific point and its distance from its neighbors, calculated using the X and Y values, as well as time in the form of a Z axis.
Filter2.py	This function takes the output from Filter 1 or Filter 10 and filters it just using the distances from a hit point and its nearest neighbors, not taking into account the time variable.
GUI.py	This file is to start the GUI interface that connects the rest of the scripts in the same folder.
Variables.py	This file holds all the alterable variables that are used in the different algorithms. Editing the values here is like editing the values with the GUI. The variable descriptions are also found here.
dualpolarityplotter.py	Function that takes the found hits and both polarities, that if have a pixel on the same point are combined and plotted in a new color.
functions.py	File that imports the other functions and applies the numba jit to turn them into machine code.
image_classifier.py	This function uses the Tensorflow trained classification model to classify objects.
main.py	Main file that connects the rest of the files and functions. It is a single function that can be ran separately from the GUI.
object_tracking.py	Function uses the jitted teacking.py to track objects and optional test data.
object_trajectory_algorithm.py	Prototype algorithm that plots a 3D graph of ab object's path and trajectory.
tracking.py	Function takes filtered data and outputs the object center and left, right, top, and bottom bounds min_distance is the minimum distance that the algorithm reaches before it "gives up" trying to find more pixels and just sets the last pixel hit as the bound min_hits is the minimum number of hits required to actually create a box tolerance is the number of pixels to go past the original bounds to create a box that is bigger than the object
utilityScripts.py	Useful scripts that can perform automated tasks. Not part of any algorithm directly.
Google Colab Files	**Description**
Cone_Cube.ipynb	A script to train a new classification model. https://drive.google.com/file/d/1IeBYIqjMfTfcQVr6pW-VIqHP-cFar_-D/view?usp=sharing

Graphical User Interface (GUI)

The team has designed a GUI that allows the user to change variables and inputs that would affect the functionality and performance of the algorithms and scripts. To launch the GUI, run GUI.py from a Python There are multiple pages accessed by clicking on the tabs at the top of the window that focuses the display to show different functionality. Some pages include quick directions and warnings that guide the user in how to use the GUI. There are also popup tooltips on the pages that tells the users how different variables affect the algorithms and scripts. The GUI window is also user adjustable making it easy to resize depending on the situation.

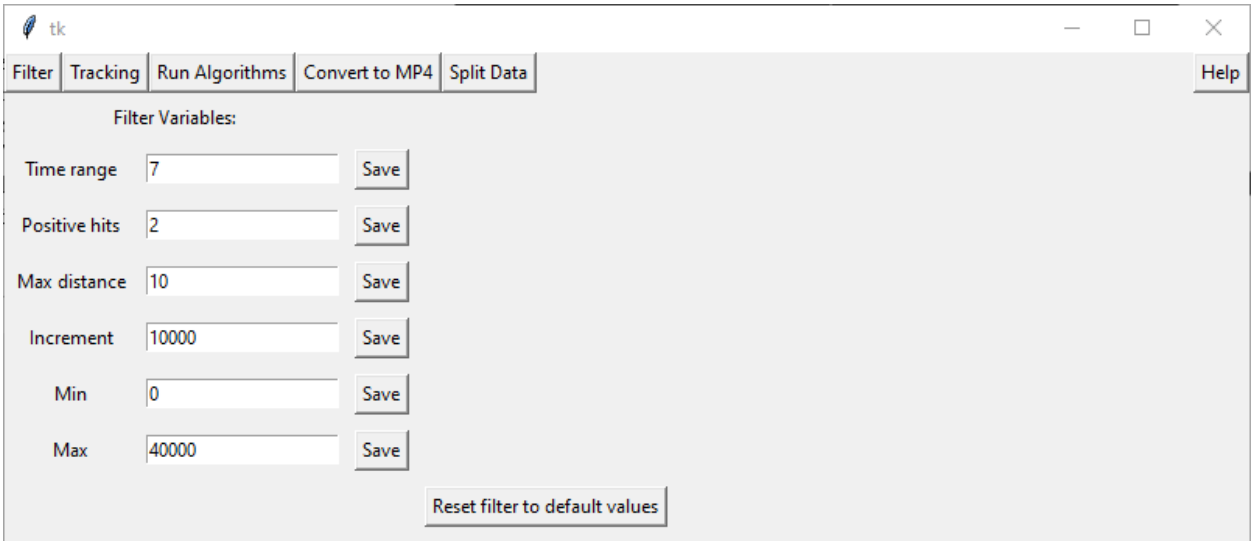


Figure 1

Upon starting the GUI the 1st page come into view. It includes the variables to adjust the filtering algorithm. Default values are prefilled inside the entry fields to give what the team found were values good for general performance. The user can click in an entry field and change the values according to their needs. The values are saved when the save button corresponding to the variable is pressed.

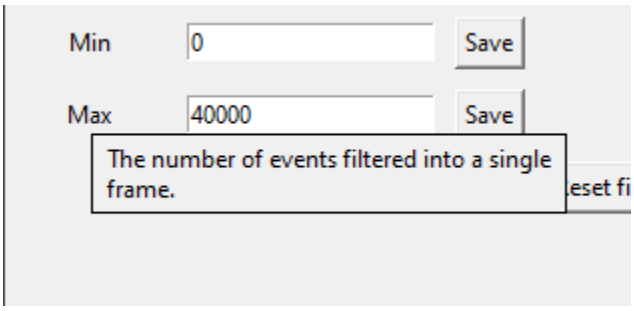


Figure 2

Tooltips popup and follow the cursor when hovering over a variable name.

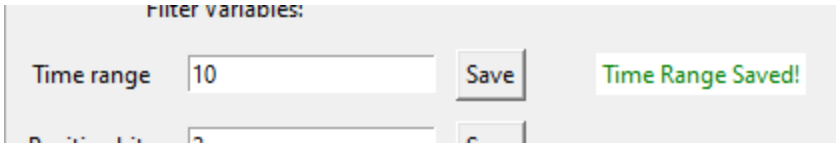


Figure 3

A save message pops up when the user presses the save button corresponding to the variable.

Figure 4

A reset message pops up when the user presses the reset button.

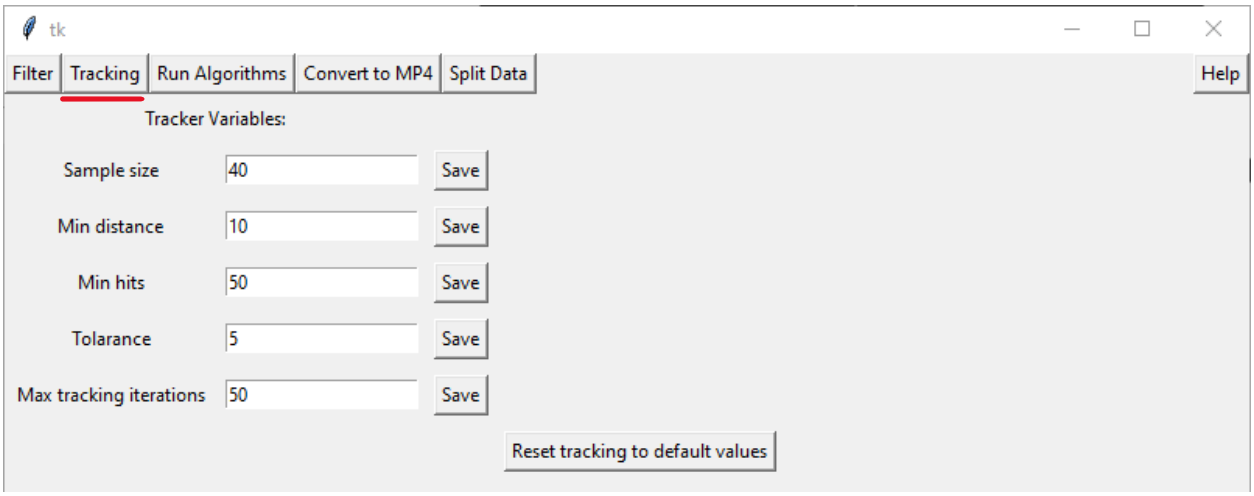


Figure 5

The tracking variables can be accessed by clicking the “Tracking” tab on top. This page includes variables that can alter the tracking algorithm. The same functionalities such as messages and tooltips are also present on this page.

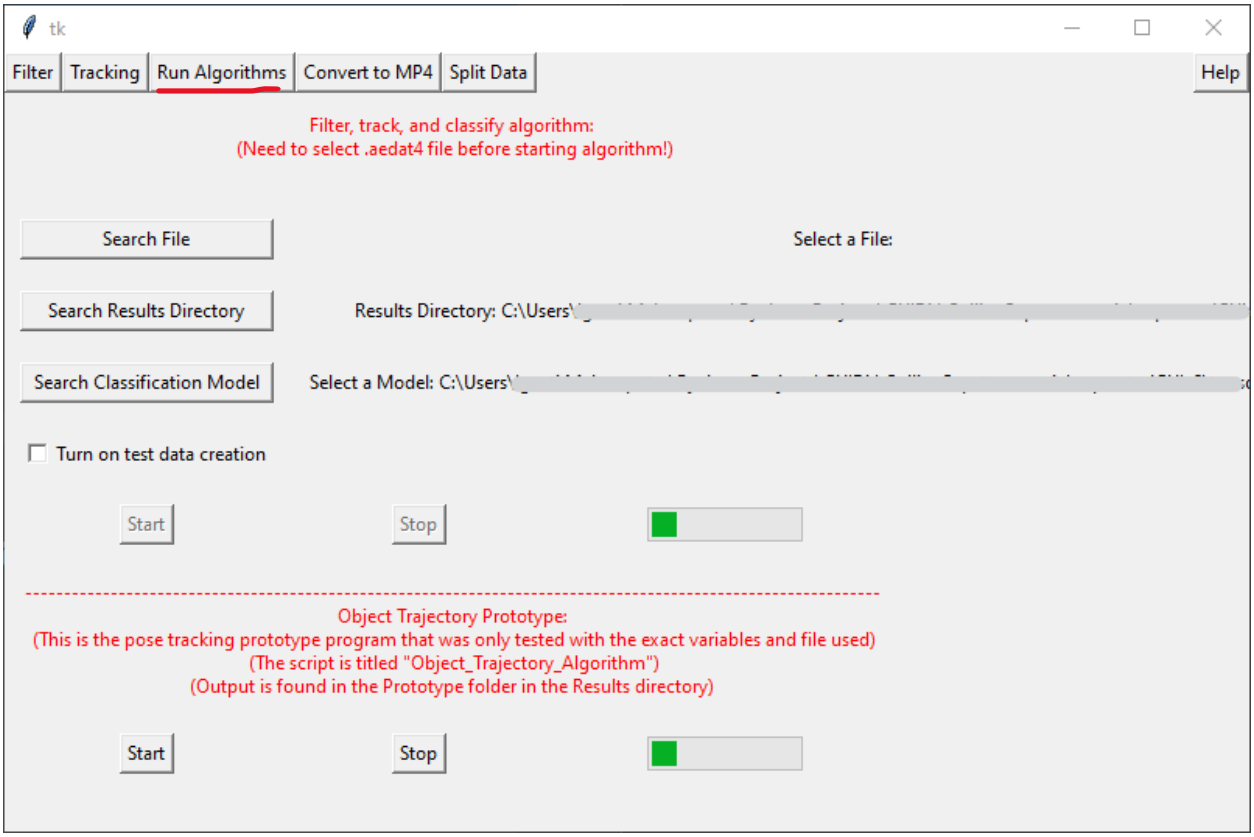


Figure 6

The user can run the main algorithms by clicking the “Run Algorithms” tab. Some options are already preselected for the user such as the results directory, where all the output files will be store, and the model, the artificial intelligence model used to identify objects. This is because the GUI would know where it is located and automatically creates a results directory if left unchanged. The default object classification model should be in the same folder as the GUI to pick the model automatically for the user. The start and stop buttons for the filter, track, and classify algorithm are initially grayed out to prevent the user from starting the algorithms before selecting an aedat4 file, the file type produced by the DAVIS346 neuromorphic camera. The option to turn on test data creation is used to create images that can be used to train a new classification model. If the option is not taken, the algorithm will execute normally and classify the objects it sees. In the object trajectory prototype section, the user can run a prototype object trajectory algorithm. This is to display the progress the team has made with determining the pose of an aircraft.

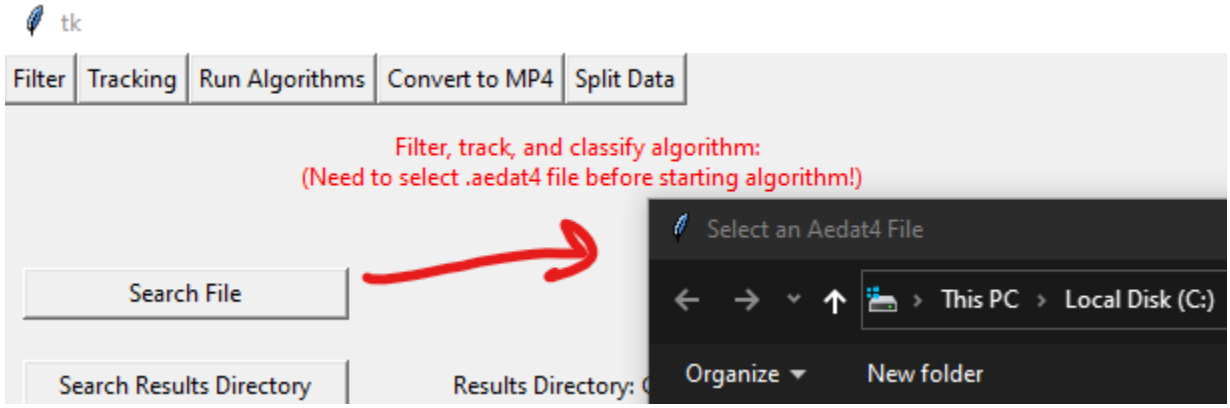


Figure 7

Clicking on the search buttons causes file explorer to popup allowing the user to select a file or folder depending on that is being asked. The results directory and model can be manually changed by the user with this functionality. The text to the right of the search buttons would update to reflect the file or folder that was selected by the user. When an aedat4 file is chosen, the start and stop button would be clickable by the user. To the right to the start and stop button is a progress bar that indicates if the algorithms are running. The prototype demonstration also has a progress bar but there is no requirement that must be met for the user to run the program.

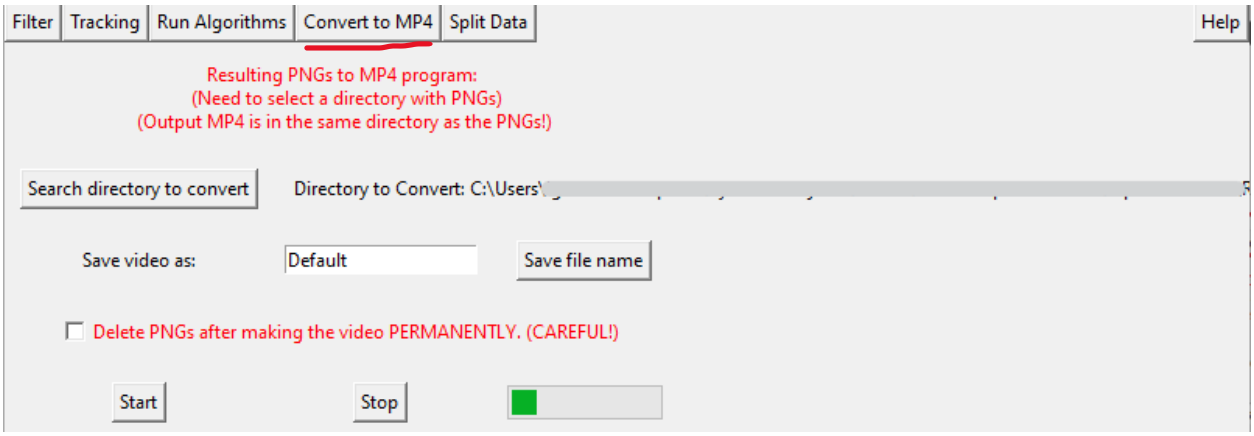


Figure 8

The user can access two utility scripts that can perform some basic tasks. These scripts are not related to the algorithms directly, but they provide actions that can be performed on the outputs of the algorithms. The “Convert to MP4” page allows the user to convert any PNG files inside a chosen directory into an MP4 video. Like the “Run Algorithms” page, some fields are prefilled for convenience. The user can pick any directory to convert into MP4 so the start and stop buttons are enabled by default incase the user already have a directory they plan to convert. The checkbox option allows the user to specify whether they want to delete the files used to produce the video.

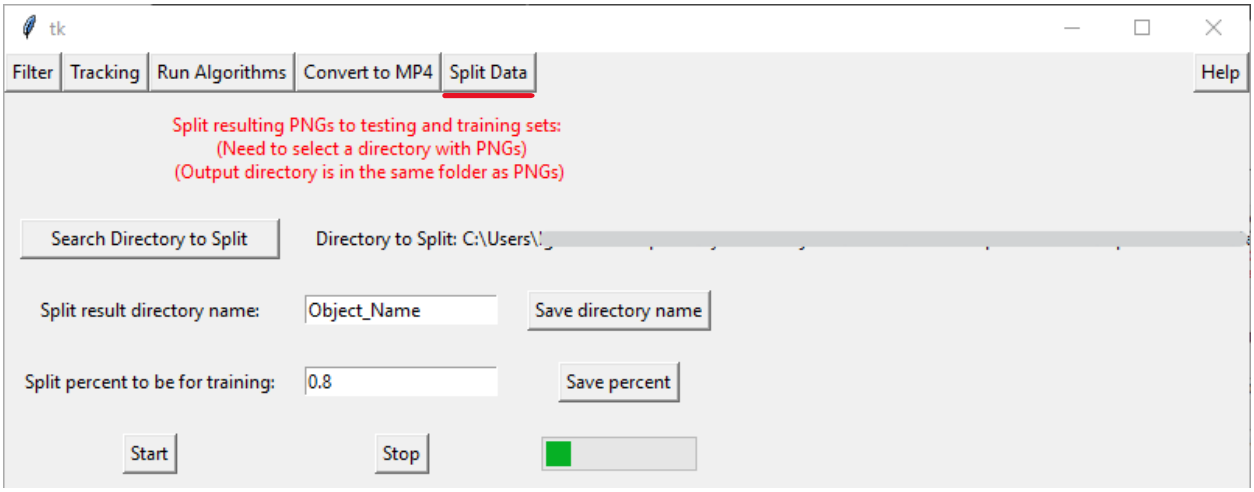


Figure 9

The “Split Data” page offers the user a tool to split the output data of the algorithms into test and train folders. This is useful if the user wants to train their own model by using the output of the filter, track, and classify algorithm. The directory that the program will split can be manually changed by the user in case they need to split data in another folder. The directory name entry box allows the user to name the folder of the split data for labeling purposes. The split percentage allows the user to specify what percentage of the data should be training or testing. Like the previous pages, the entry fields and directory option have default values for convenience.

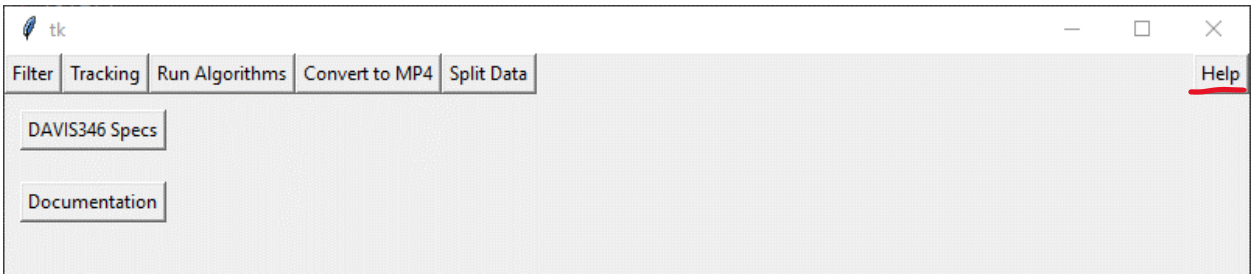


Figure 10

The “Help” tab includes buttons to open PDFs that contain in-depth explanation of how the algorithms work. These PDFs are stored where the GUI file is stored so they can be accessed offline and without starting the GUI.

Filter Algorithm

This section will explain how the Filter Algorithm operated. This filter algorithm assists with the large amount of noise that was captured by the Davis Camera. Two Filter algorithms were developed to assist with the isolation of moving objects from a background noise.

Definitions:

Noise: An appearance of undesired traces and variations in the brightness or color of an image.

Hit: an activated pixel on the DAVIS camera that has made it through filtering.

filter1.py

The first filter takes every event, with an event being a change in light intensity, and try to find the local events within a predetermined 3-dimensional threshold using the distance formula. The X-Y plane is given by the camera and time is used as the Z axis. If any event has a considerable number of local events, it can be considered as a valid event.

The first function will take the positive polarity data points from an Aedat file and filter it based on distance from a specific point and its distance from its neighbors. It utilized the X and Y values to perform the calculations and the time using the Z axis.

The function will check if a hit is found. If a hit is found, it will note down its X and Y coordinates as well as its current timestamp and it will store it in temp1_time to be the Z variables. Then it would reset the number of found hits.

Afterwards it would then index through the timestamps with a range from the index_time_rance to index+time_range. Later on, when a hit occurs, it will note its X and Y coordinate and the timestamped as temp2_time. Thereafter,it would use the distance formula to calculate the distance between the hits found in time range, accounting for distance through time.

After some time, the function will check if the distance that was calculated is less than or equal to the max_distance, we will add one to a hit variable. If the number of hits is larger than or equal to the set of the positive value, then it will be added to plot.

Inputs:

- events: a change in light intensity,
- min:
- max:
- time_range:
- max_distance: The maximum number of pixels between the last hit and the next hit to the right.
- positive:

Outputs:

- x_hits, y_hits (int arrays): x and y coordinates for all hits in the frame.

filter10.py

This function will take a negative polarity data point from an Aedat file and will filter it based on the distance from a specific point and the distance from its neighbors. It will use the X and Y values to some calculation and the Z axis as time.

The function will check if a hit is found. If a hit is found, it will note down its X and Y coordinates as well as its current timestamp and it will store it in temp1_time to be the Z variables. Then it would reset the number of found hits.

Afterwards it would then index through the timestamps with a range from the index_time_rance to index+time_range. Later on, when a hit occurs, it will note its X and Y coordinate and the timestamped as temp2_time. Thereafter, it would use the distance formula to calculate the distance between the hits found in time range, accounting for distance through time.

After some time, the function will check if the distance that was calculated is less than or equal to the max_distance, we will add one to a hit variable. If the number of hits is larger than or equal to the set of the positive value, then it will be added to plot.

filter2.py

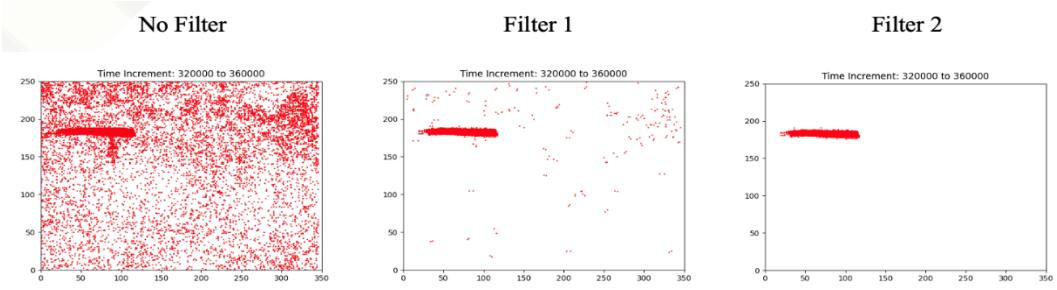
The second filter performs the same as the first filter. It takes the output from filter 1 and filters it using the distance from a hit point and its nearest neighbors but ignores the Z-axis (time) which aids in removing any excess noise besides the desired object.

Inputs:

- x_hits, y_hits (int arrays): x and y coordinates for all hits in the frame from filter1

Outputs:

- x_hits, y_hits (int arrays): x and y coordinates for all hits in the frame that is updated without the time variable



Object Tracking Algorithm

This section explains the execution of the object tracking algorithm through pseudo code. These functions take activated pixels over a certain time range that have passed through filtering and draw boxes around detected objects. The complete algorithm consists of a function called tracking in the file tracking.py and execution of the tracking function within a function called object_tracking.py.

Definitions:

Hit: an activated pixel on the DAVIS camera that has made it through filtering.

Miss: a pixel that was not activated or did not make it through filtering.

Bound: edge of the detected object, represented as a pixel. One component of the pixel (x or y) is always in line with the center of the object. Once all bounds have been found, they can be represented together as a box around the object by drawing a square that intersects each of the four pixels.

Candidate: the current hit being observed and evaluated as to whether or not it will become the bound.

tracking.py

One call to the tracking function will only return bounds for a maximum of one box. To draw multiple boxes for multiple objects, this function must be called multiple times, which is done in the object_tracking function.

This function begins by taking the first hit in the provided arrays of hits and drawing a square around it with that first hit at the center. This is the sample square whose size is user defined. All hits within the sample square are averaged. This allows us to approximate the center of the object within the localized area provided by the sample square. If there aren't enough pixels in the sample square, we'll make note

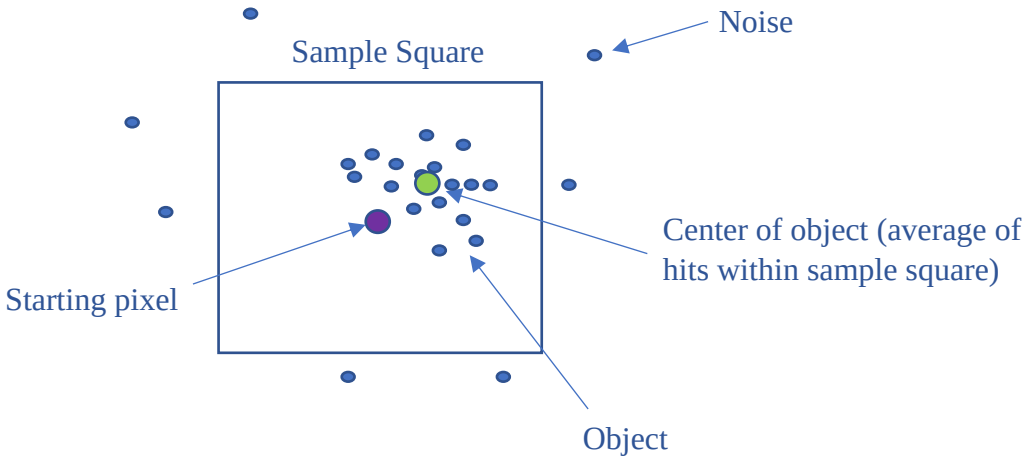


Figure 11

of those pixels so that we can delete them later and exit without providing bounds for the next function to draw a box.

Next, the function starts at the center of the object and traverses right, left, up, and down trying to find the edges of the object. Using the right bound as an example, the program works its way to the right, pixel by pixel, where each hit it comes across is a candidate for the right bound. Each hit is a qualified candidate until the algorithm hits a certain user-defined number of misses without finding another hit, or it finds another pixel to the right before then, in which case the new pixel is now the candidate. If the threshold for the minimum number of misses is met, the last candidate is set as the right bound.

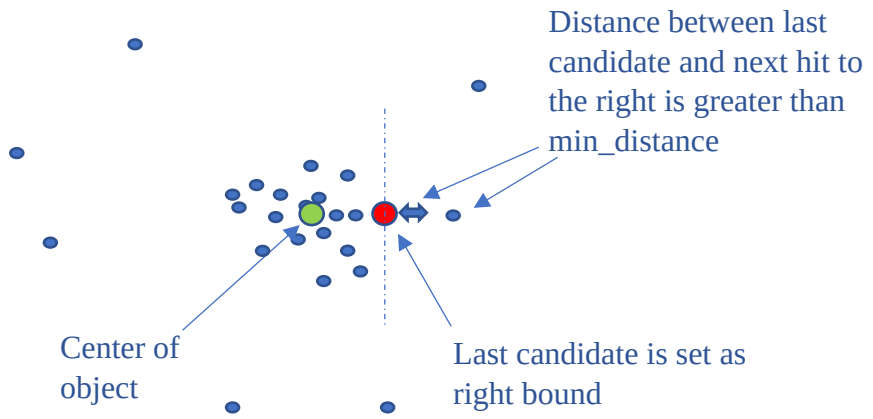


Figure 12

This is repeated for all bounds so that the edge for each side has been identified. Each of the red and green dots below are outputted as X and Y coordinates for the next function to use to draw a box.

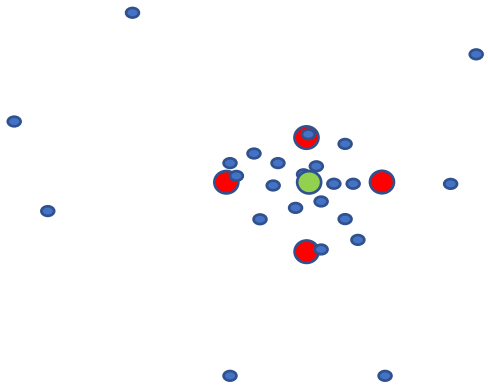


Figure 13

Additionally, the hits inside of the bounds are deleted from the array of hits, and the remaining hits are outputted so that the outside function knows that these hits have already been dealt with. Each time the tracking function is called, at least one hit will be deleted: either the hits in the sample square if there aren't enough hits, or the hits within the bounds if there are enough hits and we draw the bounds

around the object. As we call this function over and over, hits are getting deleted until there aren't any left to deal with.

Inputs:

- sample_size (int): sets the size of the sample square from which the average is taken. This is the number of pixels from the starting pixel to the right, left, top, and bottom of the sample square. This equals half of one side of the square.

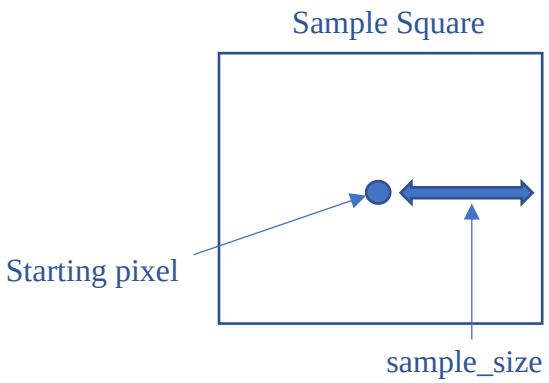


Figure 14

- min_distance (int): let's say we're finding the right bound of the object. Min_distance is the minimum number of pixels between the last hit and the next hit to the right. If the number of misses in a row between the last hit and the next hit to the right is greater than min_distance, then this means that the next hit is likely part of a different object or is noise and should not be inside the tracking box. The X value of the last hit is then set as the right bound. Min_distance is the same for all of the bounds.

- min_hits (int): sets the minimum number of hits within the sample square to recognize them as an object. If there are too few hits, it's likely noise and we won't draw a box.

- tolerance (int): number of pixels to increase each bound (adds to X values of left and right bounds and Y values of top and bottom bounds). This ensures that the box is a little bigger than the actual edges of the object.

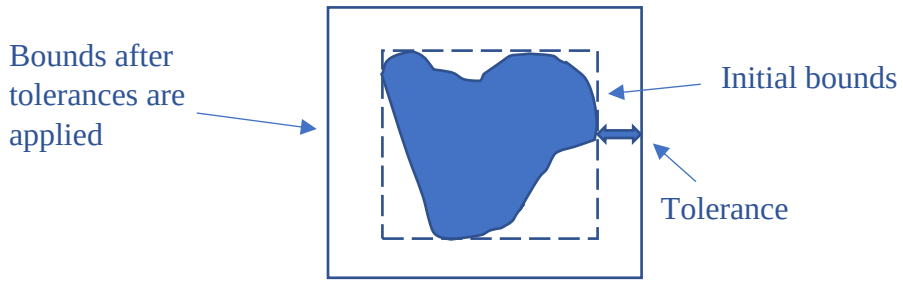


Figure 15

- x_hits, y_hits (int arrays): x and y coordinates for all hits in the frame. As the tracking function is called within the loop in the object_tracking function, these arrays will get smaller and smaller as hits are getting deleted when they are boxed or ignored.

Outputs:

Note: all mentions of bounds are after tolerances have been applied

- x_average, y_average (int): X and Y coordinates for the center of the detected object.
- x_right, y_right (int): X and Y coordinates for the right bound (the Y coordinate is the same as the center).
- x_left, y_left (int): X and Y coordinates for the left bound (the Y coordinate is the same as the center).
- x_top, y_top (int): X and Y coordinates for the top bound (the X coordinate is the same as the center).
- x_bottom, y_bottom (int): X and Y coordinates for the bottom bound (the Y coordinate is the same as the center).
- x_outside, y_outside (int arrays): X and Y coordinates for all hits outside of the bounds.
- x_inside, y_inside (int arrays): X and Y coordinates for all hits inside of the bounds.

Pseudo Code Representation:

Note: any mention of a pixel (hit, average, candidate, traverse, bound) has two components: x and y. When talking about a specific component, the variable will be followed by .x or .y (e.g. candidate.x)

```
Set sample_size, min_distance, min_hits, and tolerance //See descriptions above

Input array for all hits in frame

Create sample square whose center is first hit in array

If edge of sample square goes beyond the frame

      Set edge of sample square to edge of frame

Set average to 0

Set count to 0

For each hit
```

```
      If hit is inside the sample square
            Add to average
            Increment count
Divide average by count
If count is less than min_hits
      Delete all hits inside sample square
      Return remaining hits
      Exit function

// Find right bound
Set candidate to average
Set traverse to average
While traverse.x has not reached the right edge of the frame
      If there's a hit at traverse
            If the distance between traverse.x and candidate.x is greater
            than min_distance
                  Set right_bound to candidate
                  Exit loop
            Else
                  Set candidate to traverse
Iterate traverse.x

// Find left bound
Set candidate to average
Set traverse to average
While traverse.x has not reached the left edge of the frame
      If there's a hit at traverse
            If the distance between traverse.x and candidate.x is greater
            than min_distance
                  Set left_bound to candidate
```

```
                  Exit loop
            Else
                  Set candidate to traverse
Decrement traverse.x

// Find top bound
Set candidate to average
Set traverse to average
While traverse.y has not reached the top edge of the frame
      If there's a hit at traverse
            If the distance between traverse.y and candidate.y is greater
            than min_distance
                  Set top_bound to candidate
                  Exit loop
            Else
                  Set candidate to traverse
Iterate traverse.y

// Find bottom bound
Set candidate to average
Set traverse to average
While traverse.y has not reached the bottom edge of the frame
      If there's a hit at traverse
            If the distance between traverse.y and candidate.y is greater
            than min_distance
                  Set bottom_bound to candidate
                  Exit loop
            Else
                  Set candidate to traverse
Decrement traverse.y
```

```
Add tolerance to right_bound.x and top_bound.y

Subtract tolerance from left_bound.x and bottom_bound.y


If a bound goes beyond the frame

      Set bound to edge of frame

For each hit

      If hit is not within all bounds

            Delete hit


Return all bounds and remaining hits
```

object_tracking.py

This function is required to identify multiple objects within a certain time range. For our purposes, hits within the set time interval can be referred to as a frame. The object_tracking function simply runs the tracking function repeatedly until every hit is accounted for within the frame, whether that means that they are ignored due to too few hits within the sample space, or we determine that it is an object and output bounds to draw a box. For either scenario, those hits are deleted from the arrays, and the updated arrays are inputted into the next iteration of the tracking function. When the arrays are empty, the function plots the boxes on a graph.

Inputs:

- x_combined, y_combined (int arrays): x and y coordinates for all positive and negative hits in the frame.
- index3 (int): index for loop in main function. index3 iterates for each frame
- x_hits, y_hits (int arrays): x and y coordinates for all positive hits in the frame.
- x_hits0, y_hits0 (int arrays): x and y coordinates for all negative hits in the frame.

Outputs:

No outputs. Plots are drawn from within this function.

Pseudo Code Representation:

```
Input hits array

While hits array is not empty

      Run tracking function
```

```
If there is no object in sample space
        Delete all hits within sample space from hits array
Else
        Use returned bounds to draw box
        Delete all hits within box from hits array
```

3D Object Trajectory Modelling

The object trajectory modelling algorithm attempts to generate a 3D model from the 2D filtered data. Additionally, each point on the 3D trajectory will be assigned a vector to show direction of travel. This algorithm is very minimal and requires extensive development, which will be described. In the repository, the algorithm consists of two files, both of which are contained under the Object_Trajectory_Modelling folder. This is because the current implementation requires very specific sample parameters. The files are variables.py and object_trajectory_algorithm.py. The variables file contains the exact parameters needed to get a good output. The current sample uses the Night_Vapor.aedat4 data file.

When generating a 3D model, we must define the axes. The arbitrary axes that are currently used are shown below with Figure 16 being the 3D axes and Figure 17 being the corresponding 2D axes:

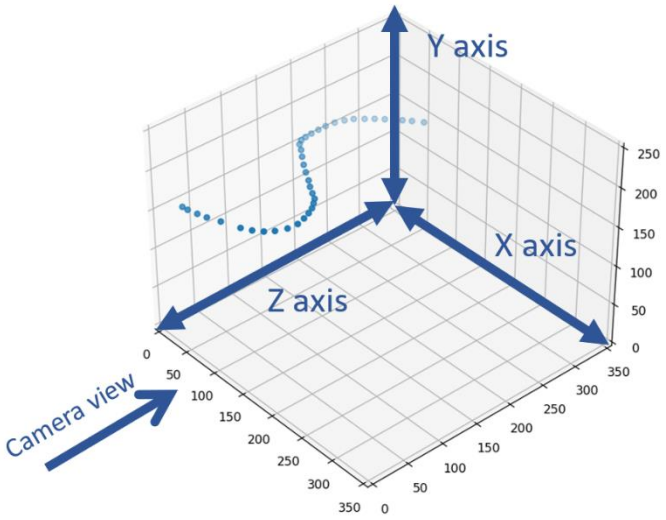


Figure 16

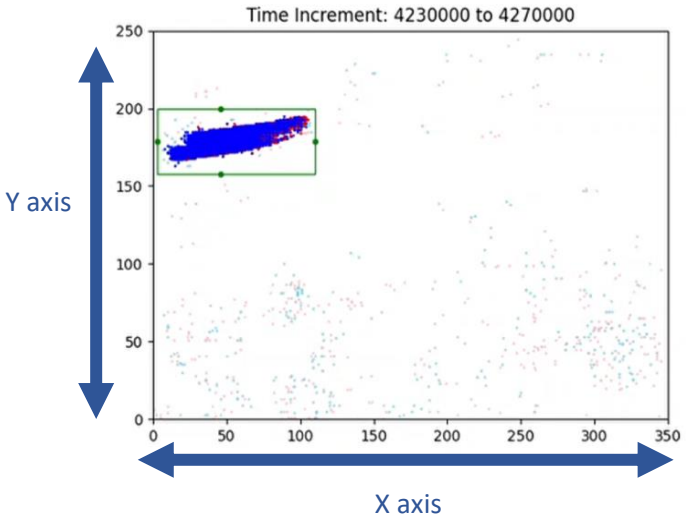


Figure 17

Each point will be assigned an X, Y, and Z coordinate based on the axes in Figure 16. X and Y in the 3D model are the same as X and Y in the 2D plot. Z is what must be calculated. The first attempt uses the pixel trail behind the aircraft to "guess" the Z component. This assumes the fact that when the aircraft travels solely in the Z axis, the trail is minimal, while when the aircraft travels solely in the X or Y directions, the trail is maximized. For our simple test data, we start by only using travel in the X axis because of the minimal travel in the Y direction. The equations below describe how we calculate each coordinate for a certain point. Note that the subscript *2d* denotes a coordinate taken from the 2D plot.

$$x_n = x_{2d,\,n}$$

$$y_n = y_{2d,\,n}$$

$$z_n = z_{n-1} \pm (w_{2d,\,max} - w_{2d,\,n})$$

$w_{2d,\,max}$ = max width of all boxes across sample time

$w_{2d,\,n}$ = width of current box

n = current point

$\pm$ determined by direction of travel in Z axis

The resulting Z coordinate is an arbitrary number that must be scaled accordingly. In Figure 16, Z is scaled between 0 and 350.

After plotting the points, we must assign a direction-of-travel vector to each point. First, we must define how to denote direction in the 3D plane. This will be done using two angles: ϕ and θ.

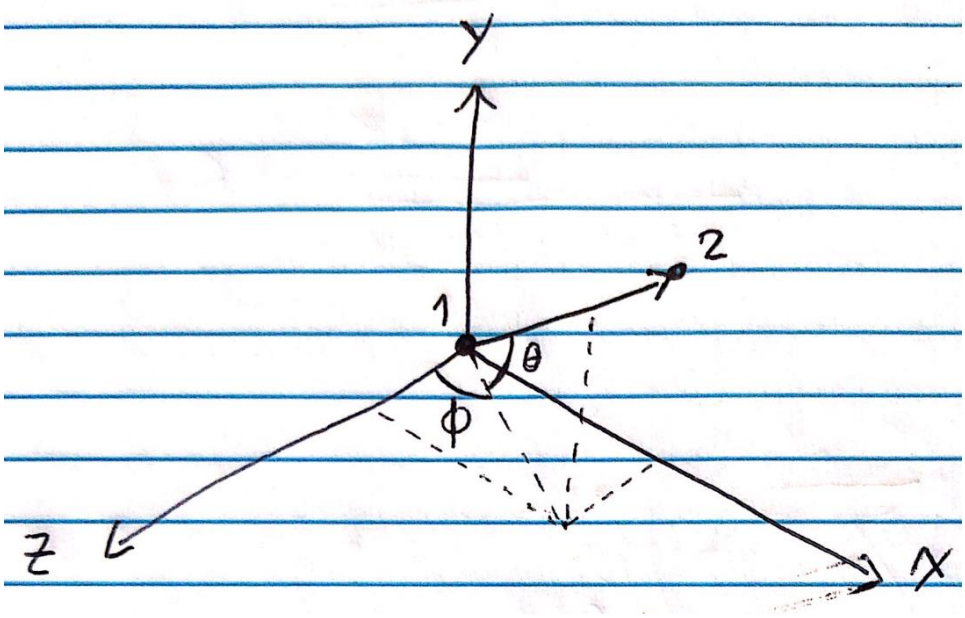


Figure 18

ϕ is the angle on the X-Z plane while θ is the angle on the Y-X or Y-Z plane.

The algorithm determines the direction-of-travel vector by drawing vectors between the previous point and the current point and the following point and the current point. The resulting angles from these two vectors are then averaged to find the angles of the direction-of-travel vector of the current point. This is shown mathematically below:

$$\phi_n = \frac{1}{2}(\tan^{-1}\frac{x_n - x_{n-1}}{z_n - z_{n-1}} + \tan^{-1}\frac{x_{n+1} - x_n}{z_{n+1} - z_n})$$

$$\theta_n = \frac{1}{2}(\tan^{-1}\frac{y_n - y_{n-1}}{\sqrt{(x_n - x_{n-1})^2 + (z_n - z_{n-1})^2}} + \tan^{-1}\frac{y_{n+1} - y_n}{\sqrt{(x_{n+1} - x_n)^2 + (z_{n+1} - z_n)^2}})$$

$n =$ current point

$n - 1 =$ previous point

$n + 1 =$ following point

Using these angles, the direction-of-travel vectors are plotted, shown below:

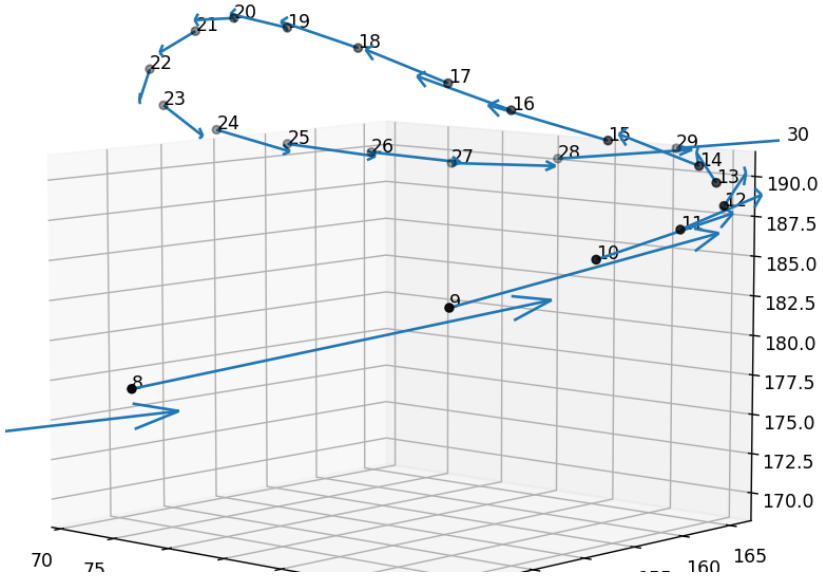


Figure 19

Important note: if determining pose, it is vital to note that the direction-of-travel vector may not correspond with the imaginary line drawn between the nose and tail for these RC planes (longitudinal axis in aviation terms). In the sample data from the Night Vapor RC plane, the deviation between the nose and the direction-of-travel is quite significant due to its high angle-of-attack. If we must calculate the pose, this deviation must be considered.

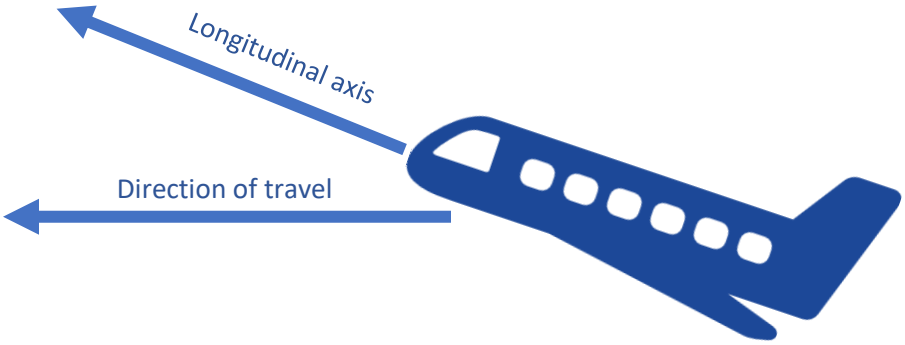


Figure 20

The current implementation of generating a 3D trajectory model has a lot of limitations. The following scenarios are not currently supported:

1. **Plane travelling significantly far away from the camera**
    - Problem: Distance affects the width of the box, so if the plane is further away than before, the width of the box shrinks regardless of travel in the X component. If the plane is travelling only in the X component, but it's further away, the decrease in the width will make the program think that it's also traveling in the Z component.
    - Potential solution: We will only use data without rapid descents/climbs so as to use the height of the box for determining the distance. We could possibly observe if the height of the boxes are increasing or decreasing. If they're decreasing, we need to account for the fact that the tail of the object in the x component will be smaller than if it was closer, and vice versa.
2. **Plane travelling both towards and away from the camera**
    - Problem: Program assumes plane is travelling away from the camera for the current sample.
    - Potential solution: We will only use data without rapid descents/climbs so as to use the height of the box for determining the direction of travel without worrying about a trail. We get an average height of a series of boxes that occur shortly before a turn. We then get an average height of a series of boxes after a turn. If the average after is larger, we'll guess that the plane turned towards the camera, and vice versa.
3. **Plane travelling with only Y component**
    - Problem: Program only looks at the width of the box, not height. If plane travels in the Y component with no X or Z components, since the width of the box will be small, the program will assume travel in the Z component.
    - Potential solution: Given the current data, it is very rare for the Night Vapor to do a full-on nosedive or strictly vertical climb. We plan on only using data without characteristics close to these two scenarios. The Y component is already being used for the two scenarios above.
4. **Plane travelling with significantly variable speed during a turn**
    - Problem: There's no way to determine the rate of travel in Z component, therefore there is no way to determine exact distance travelled in Z component since we only have 2D information.

- Potential solution: We will have to accept this limitation and assume relatively constant speed during a turn. The Z coordinates are then scaled for the user to tailor as needed.

5. **Plane with complex turn characteristics**
    - Problem: The Night Vapor is easy to use because of its simple turn characteristics. The Night Vapor turns almost exclusively with yaw control. A traditional aircraft will turn by banking which uses roll control. If we need to determine the pose, the roll will increase the complexity significantly.
    - Potential solution: No potential solution at this time.

TensorFlow and Object Classification

This section explains how objects detected from the object tracking function are classified, as well as the TensorFlow documentation to create and use a custom model to perform the object classification. These classifications are made on the zoomed in images captured from the boxes created in the object tracking function, these images are passed to image_classifier.py which contains a trained model that will perform the object classification.

image_classifier.py

This file contains the scripts to load a pre-trained model, named tensorflow_model.h5, that resides within the main.py directory. Each time the object_tracking.py finds an object and draws a box, it saves a zoomed in image of the contents of that box. If the make_test_data variable, located at the top of the variables.py file, is set to true, these zoomed in images are saved to the main.py file directory to be used to train a TensorFlow model. If the make_test_data variable is set to false, these images are passed to the image_classifier.py file, and then removed thereafter. When passed to the image classifier.py file, the script passes the image to the trained model, makes a prediction, and then passes back a string representing its prediction. As the model makes multi object classifications, it outputs an array of logits to represent its confidence on each possible classification its trained to make. These logits are compared to each other, whichever one having a higher value and an overall confidence of above 70% is chosen as the string prediction to be passed back to the object tracking function. Should the highest classification prediction fall below 70%, an unsure string is passed back. These predictions are then printed on top of the boxes drawn from the object tracking function, with the model's confidence printed next to its prediction.

Figure 21

TensorFlow Model Training:

To train our custom model, google Collab was utilized. The link to this Collaboratory is provided within the README.md file. The specific type of neural network used is a convolutional neural network. This type of neural network works exceptionally well at making generalized image classifications. To train a new model using the Collaboratory provided, one must first create a labelled dataset. This can be done by taking recordings containing a single type of object and running the main.py file with make_test_data set to true. All images created must be stored in a specific file structure that will be uploaded to the Collaboratory each time a new model is to be trained. For each type of classification, all labeled images representing that classification must be stored within a folder that is labeled with the classification of that object type. From there, all object type folders must be stored within a master folder labeled “Test_Dat”. A visual representation of the file structure used to train a model to predict between a cube or a cone is provided below:

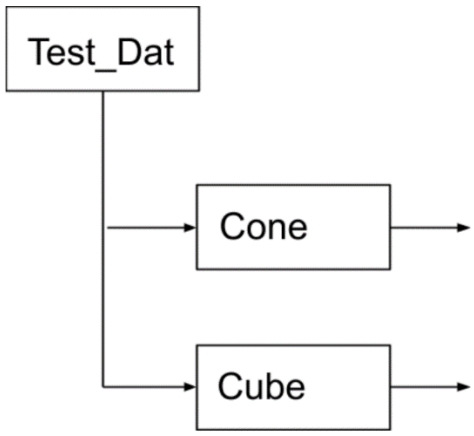


Figure 22

Once the labeled data has been created and stored in the file structure depicted above, it should be uploaded to the google Collaboratory provided. Note that each time a session is terminated on google Collab these files must be re-uploaded. Once uploaded, each code block must be run sequentially from top to bottom. These code blocks first download the required packages, load the labeled images using karas image_dataset_from_directory, build the convolutional model structure, compile the model, and finally train the model on the data provided. Once training has been completed, the model can be saved using one of the last code blocks containing " model.save('Cube_Cone_Model.h5')”. It should be noted that using this command takes around 15 minutes to download the .h5 file, a much faster way to download the trained model is possible by linking one's own google drive to the Collaboratory and running the code blocks with comments explaining how to do so. In addition to this, it is strongly recommended to use a google Collab pro account to have access to much faster GPU’s and longer run times.

Raspberry pi & Neural Compute Stick 2:

This section of documentation contains a guide on how to get this project running on a raspberry pi 4, with Intel's Neural Compute Stick 2 being used as an edge device to load the trained TensorFlow model and perform the image classifications. The structure of this section contains important notes for the installation, as well as commands to be run inside the Raspberry Pi's terminal.

GUIDE TO GET PROJECT WORKING ON RASPBERRY PI:

You must install the Buster version of raspberry pi OS as the Bullseye

version has python 3.9. As of right now the TensorFlow wheel for the armv7

version only supports up to python 3.7.

IMPORTANT NOTE:

Do NOT use sudo unless specified below. It will mess up the virtual

environment used.

sudo apt-get update

sudo apt-get upgrade

cd Desktop/

mkdir tf_pi

cd tf_pi/

python3 -m pip install virtualenv

virtualenv env

sudo reboot now

cd Desktop/

cd tf_pi/

source env/bin/activate

apt-get install gfortran

sudo apt-get install -y libhdf5-dev libc-ares-dev libeigen3-dev gcc gfortran libgfortran5 libatlas3-base libatlas-base-dev libopenblas-dev libopenblas-base libblas-dev liblapack-dev cython3 libatlas-base-dev openmpi-bin libopenmpi-dev python3-dev

pip install -U wheel mock six

NOTE:

---> Go to https://drive.google.com/uc?export=download&id=1iqylkLsgwHxB_nyZ1H4UmCY3Gy47qlOS

---> Download the file, and move it to the tf_pi directory

pip install tensorflow-2.5.0-cp37-none-linux_armv7l.whl

exec $SHELL

source env/bin/activate

pip install opencv-python==4.5.3.56

pip uninstall numpy

NOTE:

---> Go to https://www.piwheels.org/project/numpy/

---> click show more

---> Download the 1.21.4 numpy version labeled "numpy-1.21.4-cp37-cp37m-linux_armv71.whl"

---> You could use pip to download the specific version but the raspberry pi

---> will attempt to build the wheel its self (Will take actual hours).

---> Move the download into the tf_pi folder

pip install numpy-1.21.4-cp37-cp37m-linux_armv7l.whl

git clone https://github.com/oneils2018/CollinsCapstone.git

pip install matplotlib

pip install scipy

pip install dv

---> Close the terminal to exit virtual enviornment.

--->Navigate to tf_pi/env and open pyvenv.cfg

--->Change include-system-site-packages from false to true.

--->Open a new terminal.

--->Run the following commands:

sudo apt-get install llvm-11

cd Desktop

cd tf_pi

source env/bin/activate

LLVM_CONFIG=/usr/bin/llvm-config-11 pip3 install llvmlite --ignore-installed

---> Once done close the terminal

---> Navigate back to the pyend.cfg and change the true back to false.

---> Open a new terminal and run the following commands:

cd Desktop/

cd tf_pi/

source env/bin/activate

pip3 install numba --no-deps

Enviornment should now be fully setup

How to run the code:

right click on main.py and open with thonny python IDE

click on "switch to regular mode"

Restart Thonny python IDE.

Open tools > Options and select the Interpreter tab.

Click on the dropdown for "Which interpreter or device should Thonny use for running your code?"

Select "Alternative Python 3 interpreter or virtual enviornment"

Click the three dots on "Python executable"

Navigate to the tf_pi bin folder and double click activate.

Click OK.

HOW TO GET NEURAL COMPUTE STICK WORKING

cd Downloads/

sudo mkdir -p /opt/intel/openvino_2021

---> Go to https://storage.openvinotoolkit.org/repositories/openvino/packages/

---> Download the 2021.4.752 version

sudo tar -xf l_openvino_toolkit_runtime_raspbian_p_2021.4.752.tgz --strip 1 -C /opt/intel/openvino_2021

sudo apt install cmake

source /opt/intel/openvino_2021/bin/setupvars.sh

echo "source /opt/intel/openvino_2021/bin/setupvars.sh" >> ~/.bashrc

sudo usermod -a -G users "$(whoami)"

---> Reboot

sh /opt/intel/openvino_2021/install_dependencies/install_NCS_udev_rules.sh

CONVERTING .H MODELS:

To run a model on Intel's Neural Compute stick 2, the .h5 trained models must be passed through intel's OpenVino inferencing engine. This engine breaks the model down and restructures it to run much more efficiently on the compute stick's hardware. Documentation can be found on how to do this on OpenVino's website. Once a model has been passed through the inferencing engine, and all steps above have been completed, everything should be all set to run the trained model on the compute stick. Note that in order to run the main.py function on a raspberry pi, changes must be made to the image_classifier function to make predictions on a model that has been converted via OpenVino. Examples of how to do this can be found by searching "Inferencing using Intel's Neural Compute Stick 2". It should also be noted that to run the main.py file with the compute stick, the IDE used to launch the main.py file must be called from a terminal that first runs the setupvars.sh to have the right paths and variables set up to run the compute stick.

Data collected

The data collected from the DAVIS camera can be found in the folder named “Davis recordings”. From there, there are more folders named off of the data they have inside them with “Collins RC planes” having all the videos we collected of the RC planes and quadcopter. Each file has been renamed to the name of the plane/quadcopter in the video or the quadcopter and the RC plane both in the video. Outside of the folder is the “simple objects data” in there the name of the object in the recording is added on to the end of the file name. Another important area of data we collected was the pictures of the Night Vapor drone in different poses to later be used in pose recognition. In the file “night vapor photos renamed” each picture is labeled with the name night vapor, the approximate y pose in degrees, and the approximate x pose in degrees. After labeling the images it was discovered that the 0 – 90 degrees has one less pose photo taken than the rest of the plane at the same interval. This is why the rest of the photos increment at 18 degrees and the first 90 degrees increment at 22.5 degrees.

Not all the data collected was used by the group when completing the project. In most cases only one or two of the ADEAT files were chosen to be used because of either simplicity or lack of good, clear-cut data in the files. For example, when changing our focus to pose recognition the only good ADEAT files are the night Vapor Alone and the Radian alone.

DV Software

The DV software is the software used in order to collect, visualize, and customize the data taken by the camera. When the program is first loaded up you will be given the default starting screen without the DAVIS visualization even if the camera is already plugged in to the computer.

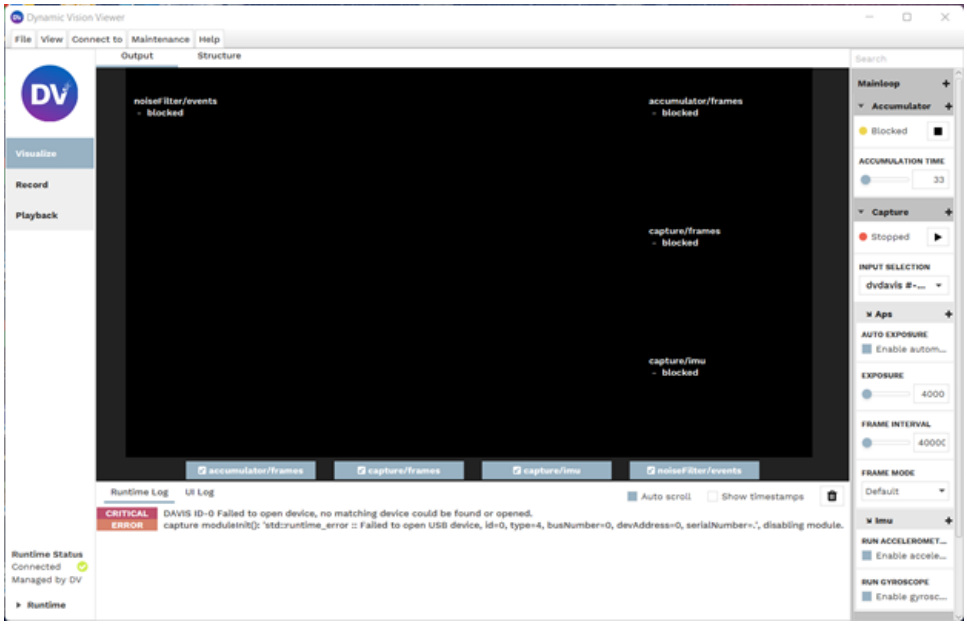


Figure 23

On the top left at the files tab this is where you can save your current workspace or load a previous one. This is not used for saving recordings from the DAVIS camera, it is only for the workspace settings. The view tab is where you can toggle the different side bar elements like the bottom console.

On the left side of the window this is where you can choose what mode you want to use with the three options being visualize, record, or playback. Every time you switch modes the software will ask you if you wish to save your current workspace.

Once you choose what mode you want, and the camera is connected to your computer you must select the DAVIS camera on the right side under the “caption” header where it has a dropdown labeled “input selection” here you will see the camera listed as “DAVIS” then a number immediately following. The other option in the dropdown is “File”. If you have previous recordings, you can load them into the software to playback and manipulate them more with the DV modules. If you choose the DAVIS camera a live feed of what the DAVIS camera sees will begin. If you choose file, you’ll need to go down to the right side to where it says “file” and gives you a text box or a file icon. Here is where you can choose a previous recording and then when you press play it will play the recording

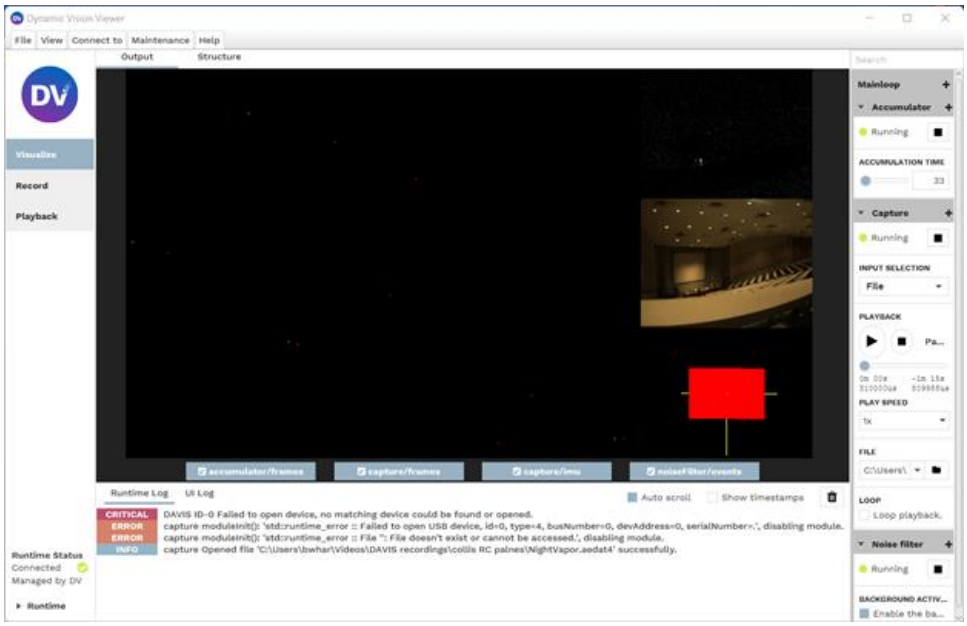


Figure 24

You can watch saved recordings or live feed from the camera in either the visualize or playback tab. In the record tab you can either record from the camera or from a previous ADEAT file.

In the record tab of the DV software the right tab changes to add a record button, and where you want the file to be saved to and what the prefix of the file name is

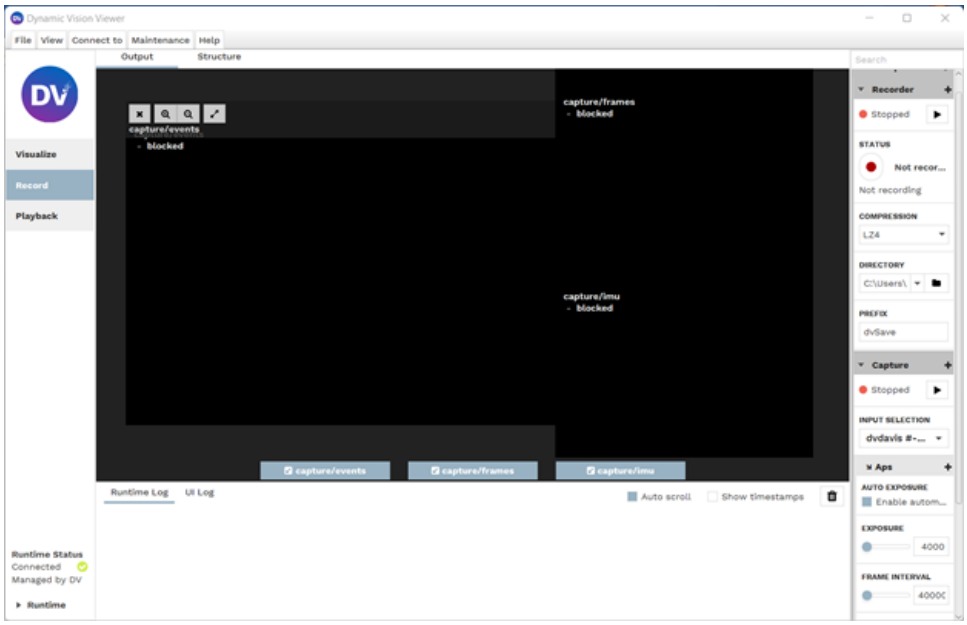


Figure 25

Right above the visualization window there is a tab for “output” or “structure”. Output is what all of the pervious images have shown with the events, frames, inertial measurement unit (IMU), and accumulator. Clicking on structure you will see the behind the scenes modules DV has that take the inputs and visualize them

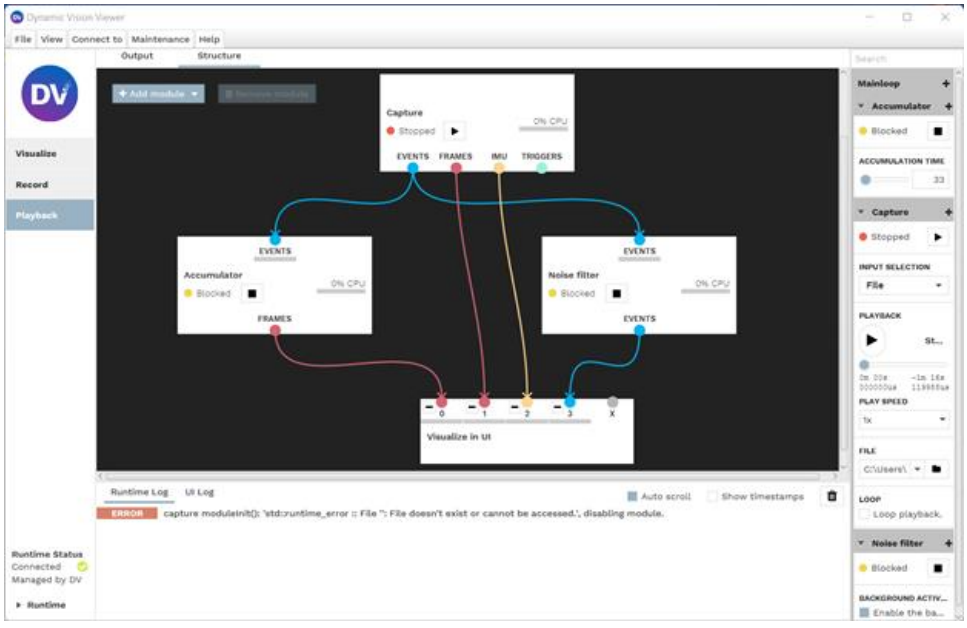


Figure 26

Each of the white blocks are called modules in the software and on the top if they take inputs that is where the line for the input will connect to (if applicable). At the bottom of the module will have the output of the module (if applicable) and an ability to start a line for output. There are different colors which correspond to different data from the file or the DAVIS camera. Blue is the events captured, red is

the traditional camera frames captured, Yellow is the inertial measurement unit (IMU), and green is triggers.

There are many different modules provided in the DV software, these can be accessed by clicking on the “add module” dropdown button on the top left of the structure window.

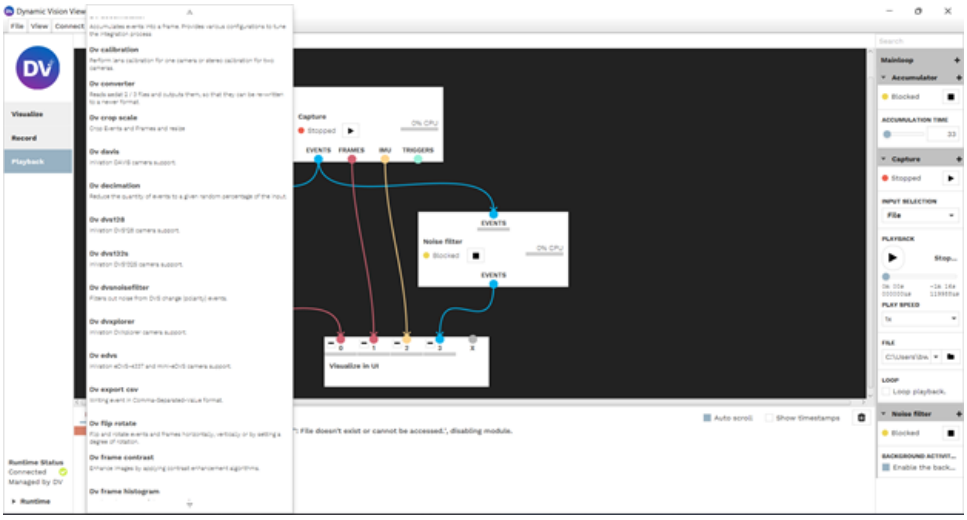


Figure 27

Each of the modules has a short description on what it is or what it does. There are different filter modules, livestream module, and save as a video format module for example. Once a module is chosen a window will pop up with a default name highlighted where you can change it if you wish and then hit save then the new module will appear. Hook up that module with the correct color line either as input, output, or both.

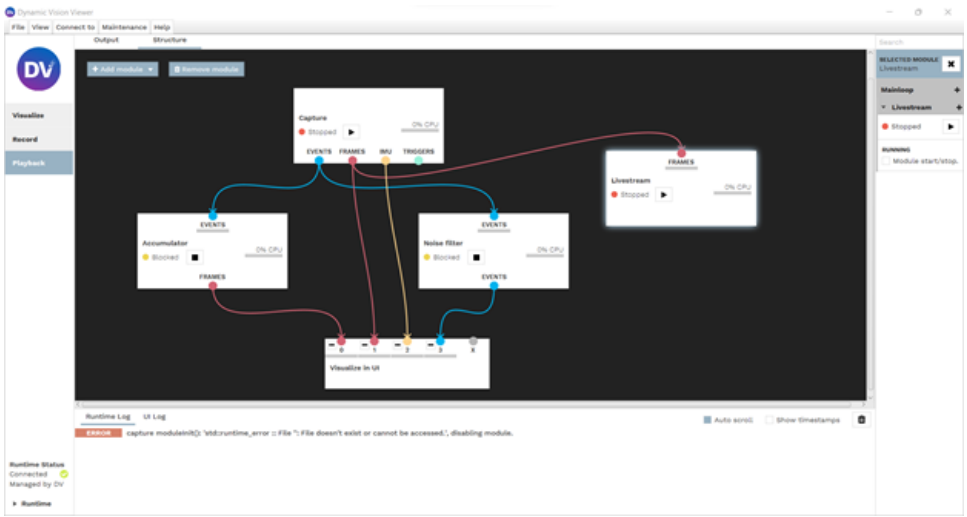


Figure 28

After it is connected properly your new module can be started and stopped in the structure window or on the right side.