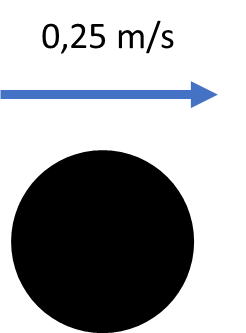
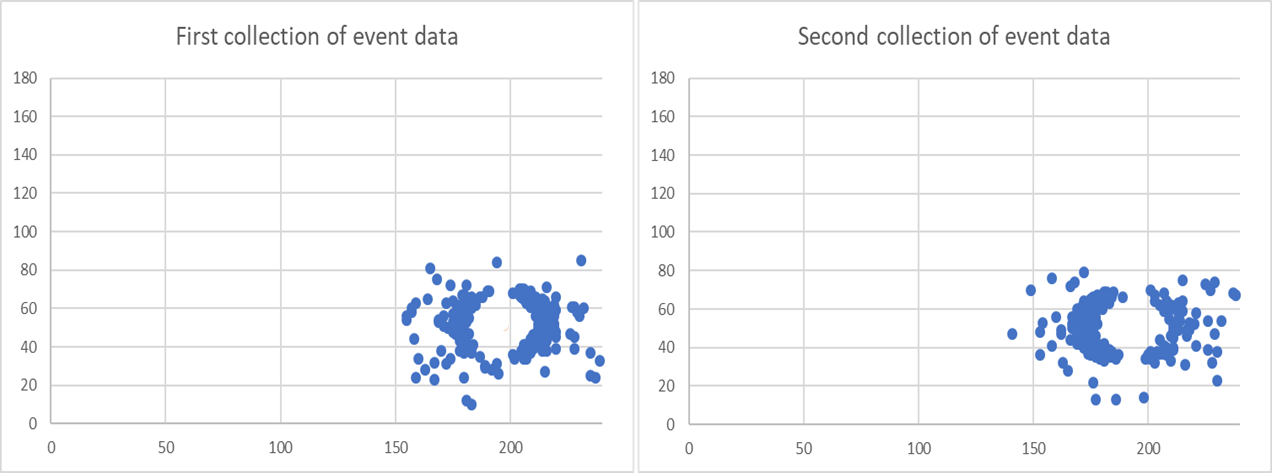
Event-based camera – DVS 240

This document is meant for people that are going to start to work with the Event-based vision sensor from Inivation (DVS240). This is a special kind of vision sensor that is able to detect moving objects. The sensor does this by detecting the light intensity in an environment with multiple pixels. With this technology, movement can be detected with less data required because only light changes are registered and a lot of the pre-processing needed for normal vision isn’t necessary anymore. An example of movement versus data is shown below to get an idea.

What can be seen in this example is that the movement of the circle is translated by the camera into events. This document is going to describe how the camera needs to be set-up, what is required to make the camera work and how the events are collected, saved and used. First the requirements are described and elaborated.

# Requirements

To be able to work with the camera for event data vision, a couple of things are necessary:

* Event-based camera (sensor + lens) DVS 240
* Processing device (Laptop, Raspberry, Nvidia Jetson, etc…)
* Python 3
* Libcaer library from Inivation <https://gitlab.com/inivation/dv/libcaer>
* For calibration, DV software from Inivation <https://inivation.gitlab.io/dv/dv-docs/docs/getting-started.html>

A close up of a device

Description automatically generatedA close up of a camera

Description automatically generatedThe camera should look as follows:

A picture containing wall, indoor

Description automatically generatedThe camera should consist out of the base (sensor, rectangular shape), the lens (cylindrical shape) and a blue special micro USB cable. The sensor should be handled with care and the lens should only be removed when necessary. If this is done screw the metal protection cap on the device as shown below. This is to protect the sensor and prevent dirt build up on the sensor.

The camera needs to be calibrated before being able to use it. This can be done in the DV software from Inivation. The DV software is made to be able to see the camera data in events or in normal frames. How the UI of this software looks is displayed on the next page.

This detected light intensity is compared with the previously detect light intensity to determine if an event has occurred. If the light has increased or decreased in light intensity with 11% or more than the camera shows this as events.



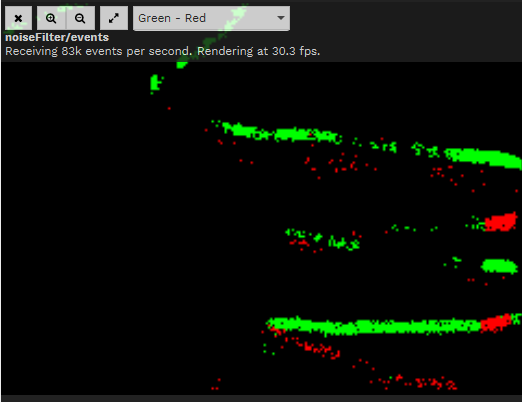
# DV software

The DV software consist out of the following parts:

* Event data
* Frame data
* Accumulator data
* IMU data
* Settings

## Event data

The Event data show at the moment that the camera is connected, the events that it is able to detect. This is shown with a distinction in positive (green) and negative (red events). Positive events are parts of the environment that are detected that have increased in light intensity. Negative events are parts of the environment that have decreased in light intensity. This is going to be elaborated further, later in this document. Below an image shows an example of a hand moving in front of the sensor and this is being translated in events.



## Frame data

The frame data part of the software shows what the camera is able to see in APS mode. This shows what the camera would see if the camera is used as a normal vision camera. An example of this is shown below.



This mode is used to calibrate the lens of the camera. The lens has to be set to give a clear view of the environment, to be able to detect events as sharp as possible. The settings on the lens are:

* Tele – Wide
* Open – Close
* Near – Far

The first setting determines if the object you’re looking at is going to appear closer or further away. This is because the focal length is changed where 4 mm is for object that need to be viewed wider and 12mm if objects need to be viewed closer. This is necessary to know for the eventual calculations that are done with the events.

The second setting described the aperture of the camera. The aperture of the camera describes how much light is going to hit the sensor. In normal cameras this can determine of the camera receives too much light or not. When a normal camera receives too much light the frame needs time to adjust to be able to see the environment. The Event based sensor doesn’t experience because it is able to operate in many light intensities as is going to be explained later in this document.

The third and last setting described the sharpness of an image. This setting is going to determine if the environment is going to appear sharp or not. When the focal length is set to Tele then the lens needs to be set to near in this setting to be able to see the environment clearly. To make sure this is done correctly the DV software is used in the APS mode. The first 2 settings can be set without the DV information but the 3 setting has to be adjusted accordingly with the DV software to get a clear image.

## Accumulator data

This part of the DCV software shows what the camera things is happening in the environment with the detected events. For example if the circle that is going to move in front of the camera is considered, then the accumulator is going to show what the environment looks like before the circle is moved in front of it. When the circle moves in front of the camera it is going to show the circle moving in the environment. This mode hasn’t been researched further and the capabilities are not entirely clear.

## IMU data

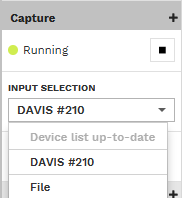
This data shows what the IMU of the camera is able to register and show the data from the accelerometer and the gyroscope. It represents the acceleration in x, y and/or z direction in m/s2. The gyroscope represents the x, y and/or z direction in °/s. This mode hasn’t been researched further and the capabilities are not entirely clear.

## Settings

The settings represent the what can be achieved with the camera. The main settings that are important for the camera are:

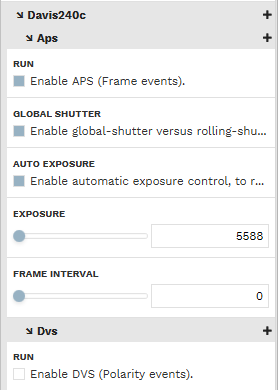
* Input selection
* Enable APS
* Enable DVS

## Input selection

Input selection describes which camera is used and if it is active or not. An example of the input selection is shown below.

Under input selection you have to choose your device. It shows all the recognized devices with their name and the DAVIS #210 represents the DVS240 camera. This needs to be selected to be able to read out the camera. The other important thing is the running above the input selection. This needs to be running in order to read out the data. When it is blocked the camera is not selected or connected properly.

## Enable APS or DVS

These settings enable the camera to show what it sees frame based or event-based. In APS mode the camera shows what it sees in frames. If the camera needs to show what it can see in events than the DVS mode has to be enabled. What it represents has been explained above.

This is what is necessary to set up the camera and be able to use it without writing software for it.

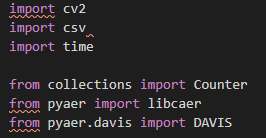
# Software

In this chapter the software is going to be elaborated further. This software has been custom written for the camera in visual studio code for Linux. A couple of packages have to be installed to python:

* Opencv - <https://github.com/opencv/opencv/tree/3.4.9>
* Libcaer - <https://github.com/inivation/libcaer>

Next to that a config file has to be saved to the computer where the program is going to be run.

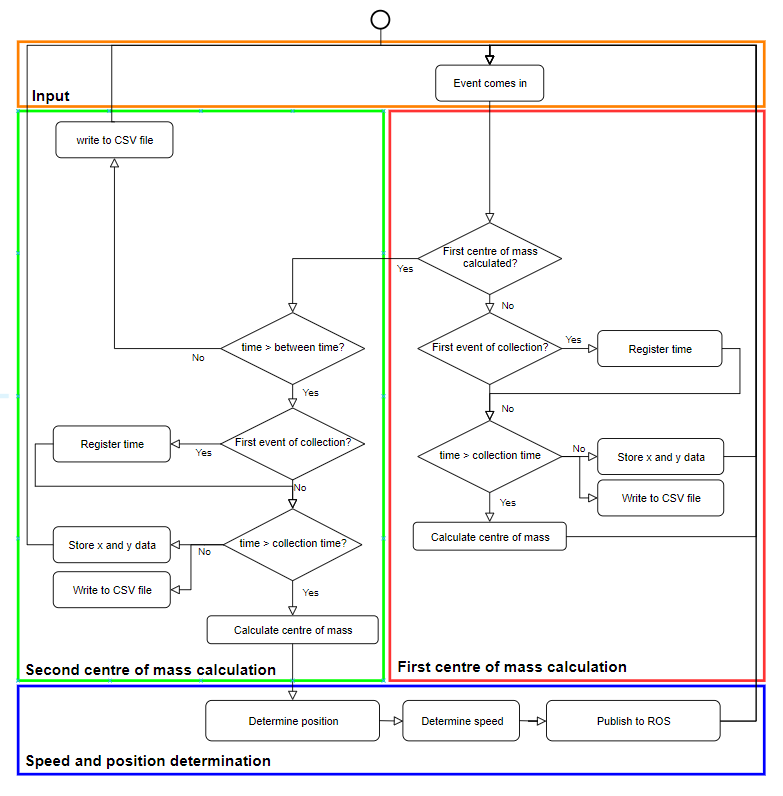
<https://github.com/duguyue100/pyaer/blob/master/scripts/configs/davis240c_config.json>

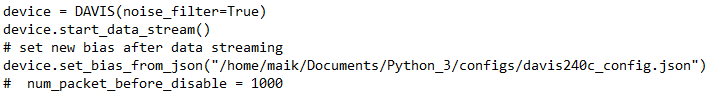
The Davis240c\_config.json file has to be downloaded and added to the program. This file doesn’t need to be changed but it is necessary to be able to use the camera within python. The beginning of the python program looks as follows:

This shows the different packages that have been imported for the program.

* CV2 is the package for OpenCV
* Csv is the comma separated values package
* Time is to include time messages
* Counter is to be able to count certain objects
* Libcaer and DAVIS from pyaer to be able to work with the camera.

The software has been designed as displayed in the following flowchart on the next page. The python file that you have to use is Position\_Program\_V5. It is not perfect nor fully tested but it gives you a general idea of what you have to do to be able to work with the camera. I recommend you try to write your own program with the information in this document as a guideline.

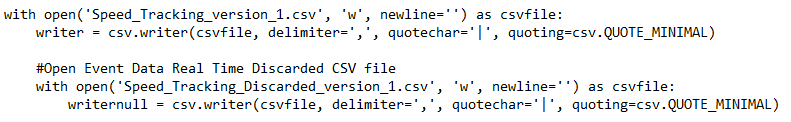


Firstly the different variables and list are initialized. These variables and list are going to be used in the rest of the program and will be explained once they are used. After the variables and list there is a piece of code that looks as follows:

This is necessary to be able to use the camera. The first line defines that the camera that is used is a DAVIS with the noise filter enabled. This is necessary because the noise filter will be used to compare the right events with each other. The second line is a command to start the camera and start the data stream. This launches the camera to detect events and send these to the program.

After that the camera settings have to be defined. This is done in a text file that has been prepared and can be found on the github. The filename is: davis240c\_config.json. On the third line the path to the file has to be put in just as shown.

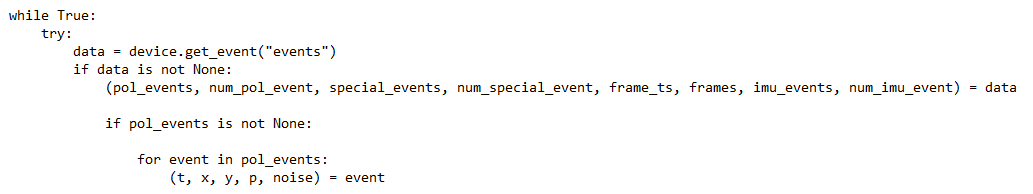
After this piece of code we get the following lines:

These lines both have the same function but are slightly different. What these lines do is they open a .csv file to store the received data in. A Comma Separated Values file is used to divide values in different columns in excel. For example if we get a time, x and y value we don’t want these values in the same cell. Instead of that we would like to see every value separately in cells on the same row.

This can be achieved with the CSV file, how this is done will be explained later.

These lines of code can be copied and the only thing that you would have to change is the file name. It the above case this is the Speed\_Tracking\_version\_1.csv.

After all the initializing we are going to take a look at the functional program. The first piece looks as follows:

Firstly, a while True loop is opened in which the event detection is going to be done. This is done in combination with a Try statement. This is to be able to get out of the program without killing the program. As long as the exception command is not used the program will try to execute the code below. The first data from the camera is received with the next line of code. This writes the received data from the camera to this list. As long as there is data being written to this list the program is going to continue. Otherwise the program will keep “checking” if the camera is sending data.

If there is data received from the camera, then this data has to be divided into useable pieces. This is done with:

(pol\_events, num\_pol\_event, special\_events, num\_special\_event, frame\_ts, frames, imu\_events, num\_imu\_event) = data

This divides the data information into useable chunks with their own name. For this application we are only interested in the first piece of data which is a list with all the events. In the next part we are going to check if there are any events in this data set. When there are no events the program goes back to checking the new data set of the camera. When there are events we are going to divide the list into single events. This is done with a for loop that goes through all the events in the pol\_events list. After that the data in the event list is going to get it’s own name. This is done with the following line of code:

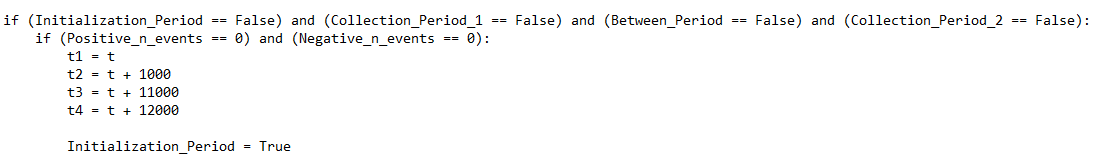
(t, x, y, p, noise) = event

As we can see the event consist of the following pieces:

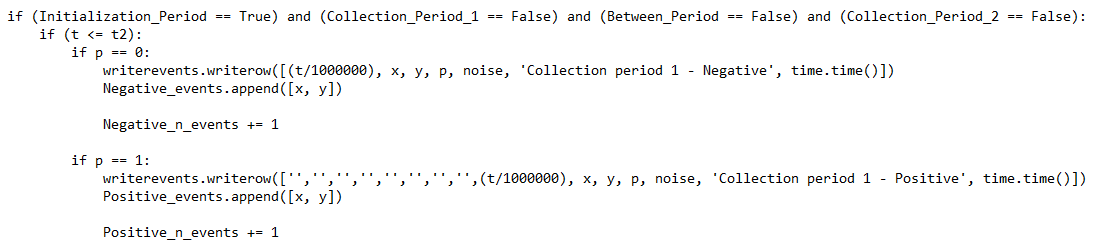
* Time when the event occurred after opening the camera connection
* X position on which x pixel of the camera the event occurred.
* Y position in which y pixel of the camera the event occurred
* P is the polarity on the event, if the event has increased in light or decreased in light.
* Noise is if the camera considers the even as noise because the light change is very small.

This has been done to make the code better understandable. This way it is clear what is being used. After this we get the following piece of code:

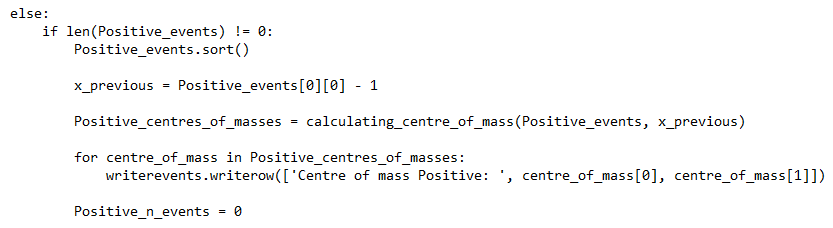
What happens here is that there is checked if an event is considered noise by the camera. If this is the case then the code goes into a different statement where the events are written to one of the csv files to be able to analyse them afterwards. If the event is not considered noise then we go into the rest of the program. This part of the program has been designed like a state machine. You start from somewhere and depending on what happens you go into the next state or into a reset state. The first state is displayed below:

The first state is the first event of a new measurement. First there is checked if all states are false. If this is the case then this state is used to determine the different measuring times. This is done with:

* T1 which is the start time.
* T2 which is the start time + 1 ms to ensure the first measurement of 1 ms.
* T3 which is the start time + 11 ms to ensure that there is 10 ms in between the first and the second time.
* T4 which is the start time + 12 ms to ensure the second measurement of 1 ms.

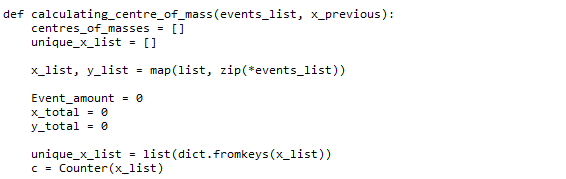
Once this has been defined the program defines the first state to have finished and it goes into the next state which looks as follows:

In this state there is checked if the time of the event is within the first collection time (t <= t2).

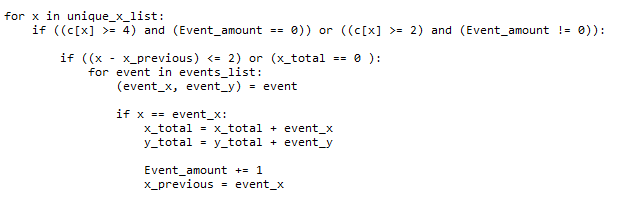
If this is the case then the next thing that is checked is if the polarity of the event is positive or negative. This is to be able to compare positive events separately from negative and vice versa. Once the event is put in positive or negative then it is written to the csv file with the useful information (time, x , y, polarity, noise, state and real time). Then the x and the y values are written into a list and the event amount is added with 1 event. This continues until the time of the event is past the first collection time. After that the following part of the program is executed:

This is the part of the program for the positive events. The negative events part is the same so this will be explained once. Firstly, there is checked if there are any positive events in the list. If this is the case then the list is going to be sorted for both values. After this a separate values is created named: x\_previous. This contains an initialized x value that is standard the lowest x – 1. This is to be able to run the program correctly. Why this is will be explained later on. After that the centre of mass is calculated for the positive events. This is done by using a function called calculating\_centre\_of\_mass.

This function calculates the centre of mass of the x and y values and this function needs the event list and the previous\_x value. What happens inside this function is going to be cut into chunks and explained now and the first chunk looks as follows:

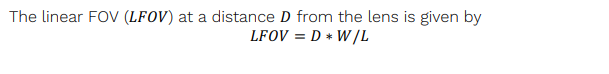


The function starts by defining the given list and the given x\_previous. The variables are going to be used inside the function. After this 2 list are defined, these are a centres of masses list where all the calculated centres of masses are going to be placed and a unique x list which is going to contain all the x values of the events list but only once. After this the list with all the x and y values together has to be separated. This is done with the map(list, zip(\*events\_list). After that a couple of variables have to be defined. This is the event amount, the total x values and the total y values. Lastly the unique x\_list is made and the x\_list is going to be used for counting values.



After that every x value in the unique list is viewed separately. It goes through a couple of conditions for a square figure. This has to do with the fact that not relevant values need to be removed. In this case the first x value has to come across 4 times in the measurement and after that the x values have to come across 2 times to be able to accurately say that these are relevant events. Lastly the x values can’t be further apart then 2 pixels otherwise the centre of mass is not accurate. These settings have been optimized for the testing pattern so these are values you have to figure out for yourself. For squares these are decent starting values.

After that all the x values and all the y values are added together and divided by the amount of events that have occurred. This is then one centre of mass. The rest of the function is written to prevent 0 values to be returned and to ensure that all the centres of masses are returned.

After that there is an in between period where there are no events used for measurements. Then the same process mentioned above is executed for the second collection period. Lastly all the collected centres of masses are compared to each other. This is done by looking for a certain range of pixels apart between centres of masses. Once the centres of masses are subtracted from each other then you have a difference in pixels over time. To calculate this to a difference in meters a formula is given to calculate the amount of meters per pixels. This formula is:

This is the distance from the lens to the surface \* the width of the camera array/ focal length. This is the total distance seen by the camera and this has to be divided by the amount of pixels on the axis to get an amount of metres per pixel. For more information see:

<https://inivation.com/wp-content/uploads/2019/08/DVS240.pdf>

Lastly this distance can be divided by the measuring time to get to the speed