HONEYBEE TRACKING

CP302
Final Report

Date: 13-11-23

—

Submitted By:
Aditya Singh- 2020eeb1147
Shashank Bayal- 2020eeb1206

—

Supervisor: Dr Suman Kumar

Instructor: Dr Mahendra Sakare

Declaration

We declare that all the everything used in the following documentation is either our own words or have been cited with the utmost honesty. We also declare that we adhere to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any of the facts/results/data/ideas in the submission. We understand that violation of this declaration will lead to a disciplinary action and can invoke penal actions from sources that have not been properly cited.

Abstract

The objective of this project is to attempt tracking of identical objects in front of a variable background using an object detection model trained for the specific objects in question. The objects have no re-identifiable features for us to use so the project also attempts to come up with a solution to that issue but no major strides in that direction have been made. We have however been able to make an in-house proximity based visual tracker which is going to be used further in the efforts to solve the problem because an attempt has been made for the code to be very edit-friendly.

Contents

Chapter I

Introduction

The project starts with a need for us have a tracking application in a separate project that has for that time being has been only identifying the location and classification of the species of the insect in a given frame. Now the snap rate of the camera being used for the detection is very low, around 8 frames per second once the motion has been detected. Working on that side is not a part of the project, just the input parameters we have to work with. With that out of the way we can very clearly identify the requirements that we need to fulfil for a complete success of this project, or in other words this is what we're setting off to accomplish in this project.

Having a tracker in the first place, this is the most basic requirement of the project which needs to be acquired before we can even think of the other requirements. This was first thought possible from starting with ByteTrack as a base. Link to the repository is as follows: https://github.com/ifzhang/ByteTrack . This turned out to be broken at the time of the testing done with it to the point of us not being able to get it to run at all, compilation itself not being possible we turned to what we had learned in the research done in selecting ByteTrack in the first place. Knowing how a basic tracker algorithm works in conjunction with a trained detection model we made one from scratch.

That part being solved we move on to the next problem that was foreseen at the start of the project, which was the reason for us not using a tracker as is and wanting to use the tracker as base is the low frame rate we would have to work with. Tracking as a basic concept requires a continuous flow of frames with at least a reasonable frame rate this however is something we don't have in any sense of the word. Getting a few frames that are linked to one-another is to be considered lucky in case. So how does one even define tracking in this case. We'll talk about that in much more detail later but in summary we can consider the case of a few, let's say at least 3 consecutive frames and try to have one 'label' for the same insect in the consecutive frames.

That being done we have the final part to consider that we were not able to do in this project but that will be the logical progression from this point in the problem is to try to solve the problems arising from the low frame rate and the problem of an insect leaving the frame and then coming back into it. This would be done by re-identifying it in case of any other object which would have that be possible for it. This in theory would be solved by having a probability distribution for a specific insect leaving and then measuring how likely it is for an incoming insect is being that insect and not a new entry into the insect number seen in that video.

With the three phases of progression being now defined let us look into them one by one and understand what work was done under the project and how much of the theoretical issues were solved.

Chapter II

ByteTrack

This is the first choice that we looked towards at the start of the project to be used as a base for the code to be written, however we ran into problems immediately after downloading a copy of the repository to experiment with.

ByteTrack is a very well-known object tracker with it being considered a lot more efficient that a simple by the books approach of having a score associated with the objects being tracked that go into the background or are occluded lowered so that they do not interfere with the tracking of the other objects that are still in the foreground.

The following exert from the article https://www.datature.io/blog/introduction-to-bytetrack-multi-object-tracking-by-associating-every-detection-box#:~:text=The%20main%20idea%20of%20BYTETrack,on%20their%20similarities%20with%20tracklets says it better.

> *The main idea of BYTETrack is simple - keep non-background low score boxes for a secondary association step between previous frame and next frame based on their similarities with tracklets. This helps to improve tracking consistency by keeping relevant bounding boxes which otherwise would have been discarded due to low confidence score (due to occlusion or appearance changes).*

We thinking that this helps with the common issue of occlusion chose this as a base and went through with documenting the code that was in the repository and we found the most useful part of the code being the Kalmen Filter code.

ByteTrack itself at that time was facing issues with its compatibility for Cythonbbox a repository that we were going to need if the project was to continue with ByteTrack and the latest NumPy version had removed a key component used for the code of Cythonbbox. “np.float” was discontinued because to NumPy it was a redundancy but it was the function used in Cythonbbox leading to NumPy being a dead end otherwise.

Kalmen Filter code itself was very neatly documented and explained by Shashank and the documentation is as follows.

Kalmen Filter documentation

-Imports

-numpy as np (it is used for matrix operations)
-scipy.linalg **(**It is used for doing linear algebra functions**)**

-Constant:

-chi2inv95 **(**it represents degree of freedom from 1-9 as a list for 95 quantile**)**

-Class Kalmanfilter

It is the main class which combines the sensory data from various sensors with prediction from mathematical model based on the system to the true real state of the system at present)

-Here two constants are declared

1. **ndim =4** **(**ndim corresponds to the dimensionality of our system**)**
2. dt = 1 **(**dt corresponds to the time step**)**

-Various matrices are declared in the class

1. **self._motion_mat = np.eye(2*ndim, 2*ndim)** **(**this is a 4x4 identity matrix that will store the state estimate for uncertinity in motion and observation)

This matrix show it incoporates changes and differences in estimates and uncertainity in data and with every iteration in time step it evolves the parameters of the kalman filter model

-for loop

-def ini

this loop update the matrix for the values of x,y,a,h and Vx, Vy, Va, Vh
it also initiates tge values of starting values of the starting point
it also sets the weights for position and velocity

-def initiate

it creates track from unassocaited measurement.
In this part the standard deviation and covariance values are calculated.
Then mean position, mean velocity and covariance is returned.

-def predict

here standard position and velocity value combined with motion_covariance are returned

-def project

The code projects the state's distribution on a measurement space

-def multipredict

It calculates std_pos and std_vel values for each row in the dataframe.
It returns the mean vector and covariance matrix of the predicted state.
Unobserved velocities are initialized to 0 mean.

-def update

It runs the Kalman filter correction step
The inputs mean, covariance, measurement are used to calucate kalman gain
Then innovation variable to calculated and using its value the new_mean and new_covariance are returned

-def gating_distance

Here the code calculates the distance between a state distribution and the set of measurements.
Measurement which is a list containing measurements to be compared to the state distribution.
Only_position is a boolean flag tells weather to use the position information for distance calculation.
The mahalanobis distance is used to incoporate both location and shape of probability distribution.
Earlier the cholesky decompsition is used for efficient computation of mahalanobis distance.
It returns an array of length N, where the i-th element contains the squared Mahalanobis distance between (mean, covariance) and measurements[i]
If the metric used is guassian then the sum of squares is calculated for each dimension of data instead of Mahalannobis distance.
In summary the code calculates distance between the mean projected of the specified used metrics (that can be gaussian or maha) using properties of dimensionality reduction (refer to projection) for efficiency and utilizing Chlesky decomposition for optimization

Chapter III

VoidTrack

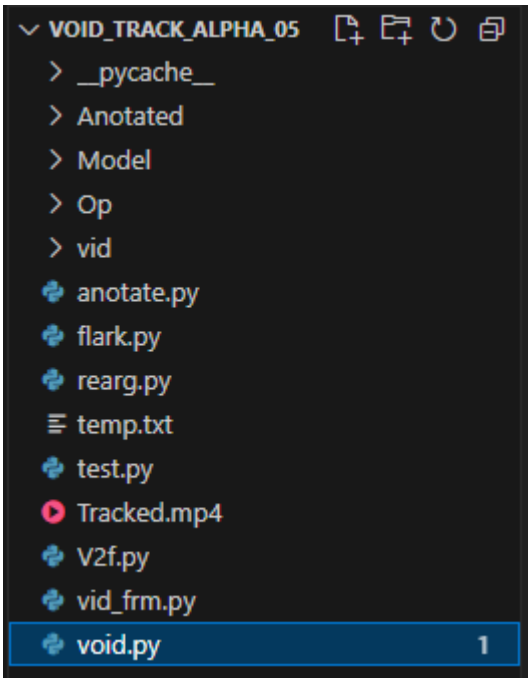
VoidTrack is the result of the work put into the project and is a tracker that can track a group of identical objects, tested for three at once by the proximity of the previous frame's positioning of the predicted location to the once of the new frame's.

Now one would ask why would this alone work when there is a lot more effort put into the trackers used for any other application and that is true, the quality of the results is not as pristine as it could be but the main problem is that the other rigmaroles that we would do in any other tracker would not work here.

A normal tracking algorithm works on three components to predict a score of what the objects of the new frame correspond the older frame. These three are proximity, velocity and features. As we can tell from the type of object in this case a specific insect the approach of telling an insect A from insect B from the same species is nigh impossible. The other approach that we reject is not by choice but by the input provided to us. The issue is with an extremely bad frame rate to a broken continuity of frames we cannot employ the approach of velocity either. So, the only thing we're left with is the approach of proximity and thus it is our only olive branch which we must optimise.

The code has many different components which come together to form a cohesive unit that takes in an input that was constructed from 8 frames that happen to be roughly consecutive that have been pre-stitched into a video that is assumed to be done outside of the code's domain.

We take that input and break it down into the 8 frames we have and then add different color boxes on top of the predicted location of the bees and we try to keep the same-colored box on the same bee as obvious to our human eyes.



Void.py

This is essentially the “run.py” or the “main” code of the custom repository and is code that calls on the others to make the cohesive unit. The code starts with calling the two major libraries of “OS” and “torch”. OS is used for path manipulation so this code can run anywhere on any computer without the need to alter the paths in the code itself. And Torch is used to call upon the Yolo.V5 model that has been pre-trained on this by another team working with us in the larger project and here is treated as another point of input.

We then import the codes of video formation (vid_frm.py) and video breakdown (v2f.py) for the purpose of breaking an input video and forming an output video after the processing is done. This is the same code used for the last project in cp301 which came in handy for this term project as well so it is used directly with some minor changes done to them to support the independence from the specific path but just in case the changes are not enough the preferred location of this repository would be: D:\I.Research\Void_Track_alpha_05\. It should work regardless but just in case it does not that is the exact repository path it was made under.

After that we call upon the next major code written for the repository, before moving on to that let us show case the code for void.py

void.py > ...

```python
import torch
import os
current_directory = os.path.dirname(os.path.abspath(__file__))
model_path = os.path.join(current_directory, 'model/model.pt')
model = torch.hub.load('ultralytics/yolov5', 'custom', path=model_path)
from anotate import ant
from vid_frm import form
import V2f
file = open("temp.txt","r+")
file.truncate(0)
file.close()
no_frm = V2f.video()
print ("no of fromes :"+str(no_frm))
idx = 0
while ( 1==1 ):
    print("#"*150)
    image_path = 'Op/Frame%05d.png'%idx
    image_path = os.path.join(current_directory, image_path)
    out_path = 'Anotated/Anotated%05d.png'%idx
    out_path = os.path.join(current_directory, out_path)
    if os.path.isfile(image_path) == False:
            break
    print("given frame:"+str(idx+1))
    inptx = ant(image_path,out_path,idx,model)
    idx = idx + 1
form(os.path.join(current_directory, 'Anotated'))
```

Anotate.py

This is a relatively simple code all it does is take the predicted values for the midpoint of the box predicted as the insect and we give the first prediction the first label in the form of a color code, that is the purpose of the RGB values in the color array. We currently only support a limited number of color codes but the array can be given as many color codes as we need. This does however give us a limit to number that can be tracked without us running into an out of bounds error which can be solved by us adopting a different method for color allocation but that is the one we ended up using for this experimental repository.

Other than that, we call upon the two other major code components from the repository which are flark.py and rearrange.py. The first just extracts the values we need from the model's output and well running it before getting that output.

flark.py > ...

```python
import cv2
import pandas as pd
##########################################################################################################
def darknet(results):
    df_list = pd.DataFrame()
    df_list['x_centre'] = (results.pandas().xyxy[0]['xmin'] + results.pandas().xyxy[0]['xmax']) / 2
    df_list['y_centre'] = (results.pandas().xyxy[0]['ymin'] + results.pandas().xyxy[0]['ymax']) / 2
    return df_list
##########################################################################################################
def main(image_path,model):
    #inital current directory location
    input_image = cv2.imread(image_path)[..., ::-1]
    #inital model placement
    results = model(input_image)
    df = darknet(results)
    df = df.to_string(header=False, index=False)
    return df
##########################################################################################################
```

The other code we call upon deserves an explanation of its own so let us showcase annotate.py.

anotate.py > ...

```python
from PIL import Image
from flark import main
from rearg import rearange
##########################################################################################################
def ant(image_path,output_path,idx,model):
    df = main(image_path,model)
    mid_points = df.splitlines()
    lenght = len(mid_points)
    mid_points = rearange(mid_points, lenght, idx)
    input_image = Image.open(image_path)
    pixel_map = input_image.load()
    width, height = input_image.size
    wd = int(width/100)
    c = 0
    Flag = 0
    color = [(255,20,20), (20,255,20), (20,20,255), (255, 255, 255), (34, 67, 19), (255, 255, 255), (56, 88, 199), (3, 46
```

```python
for i in range(lenght):
    start = mid_points[i].split()
    if (Flag == 1):
        Flag = 0
        c = c + 1
        continue
    if (i < lenght-1):
        if(mid_points[i]==mid_points[i+1]):
            Flag = 1
```

```
        for j in range(wd):
            for k in range(wd):
                pixel_map[int(float(start[0]))-j,int(float(start[1]))-k] = color[c]
                pixel_map[int(float(start[0]))+j,int(float(start[1]))+k] = color[c]
                pixel_map[int(float(start[0]))-k,int(float(start[1]))+j] = color[c]
                pixel_map[int(float(start[0]))+k,int(float(start[1]))-j] = color[c]
        c = c + 1
    input_image.save(output_path, format="png")
    return mid_points
##########################################################################################
```

Reagr.py

This is the key of the code, we here change the inherent order in which the yolo.v5 model is detecting the objects, the order on in its own is mostly random and since that is what we're basing our labels we would also get a random label allocation and obviously that is bad so we rearrange the numbers that we get from model with the order rearranged we also have the labels rearranged to the correct ones.

The rearrangement for some reason that we have not identified yet was overwriting the other labels in certain cases so to solve that problem we have added a leftovers technique that allows us to run the allocation again just in case there is any overlap which would leave us with a leftover label that did not get allocated. At the moment we are limited to just one left over but this can be extrapolated to an array of leftovers if need be.

Other than that, all this code does is give the proper order to for which the labels are then given out.

```
import numpy as np
def rearange(Pn, length, idx):
    file = open("temp.txt","r")
    Crn_1 = file.readline()
    Crn_1 = Crn_1.replace("'","")
    Crn_1 = Crn_1.replace("[","")
    Crn_1 = Crn_1.replace("]","")
    Crn_1 = Crn_1.split(",")
    file.close()
    Crn = Pn
    lnth = len(Crn_1)
    print("Pn = "+str(Pn))
    print("Crn_1 = "+str(Crn_1))
    if(len(Crn_1)>len(Pn)):
        Crn_1[len(Crn_1)-1] = ""
        lnth = lnth - 1
    if (len(Pn)>len(Crn_1)):
        Crn_1.extend('0 0')
    if (idx != 0):
        Crn = Crn_1
        for i in range(lnth):
            Thr = 100000
            Pn_copy = Pn
            Crn_1_split = Crn_1[i].split()
            for j in range(length):
                Pn_split = Pn[j].split()
                dif = np.sqrt(np.square(int(float(Pn_split[0]))-int(float(Crn_1_split[0]))) + np.squar
                #print("dif ="+str(dif))
```

```
+ np.square(int(float(Pn_split[1]))-int(float(Crn_1_split[1]))))
```

```
                #print("dif ="+str(dif))
                #print("unupadted Thr = "+str(Thr))
                if (  dif < Thr ):
                    Crn[i] = Pn[j]
                    #Pn[j] = '100000 100000', this is a really bad idea don't do this please
                    Thr = dif
                    #print("Thr updated = "+str(Thr))
            Pn = Pn_copy
    #print("Crn = "+str(Crn))
    for i in range (len(Pn)):
        Flag = 0
        for j in range (len(Crn)):
            if (Pn[i] != Crn[j]):
                Flag = Flag + 1
                #print("Flag = "+str(Flag))
        if (Flag == len(Pn)+1):
            left_over = Pn[i]
            print (left_over)
    for i in range(len(Crn)):
        for j in range(len(Crn)):
            if ((i!=j) & (Crn[i]==Crn[j])):
                Crn[j] = left_over
```

```
                Crn[j] = left_over

    file = open("temp.txt", "w")
    L = str(Crn)
    file.writelines(L)
    print("Crn fixed = "+str(Crn))
    return Crn
```

Video formation and breakdown

The code for these two is extremely simple and taken from the last years term project with very minor changes to make it work with any file location as long as everything insider the repository is placed correctly.

We have already had an explanation of these two codes in ‘void.py’ ’s explanation so we’ll just move forward to the code showcase.

vid_frm.py > form

```python
import cv2
import os
import moviepy.editor as mvp
os.system('cls')
current_directory = os.path.dirname(os.path.abspath(__file__))
def form(image_folder):
    video_name = model_path = os.path.join(current_directory, 'temp.avi')
    images = [img for img in os.listdir(image_folder) if img.endswith(".png")]
    frame = cv2.imread(os.path.join(image_folder, images[0]))
    height, width, layers = frame.shape
    video = cv2.VideoWriter(video_name, 0, 10, (width,height))
    for image in images:
        video.write(cv2.imread(os.path.join(image_folder, image)))
    cv2.destroyAllWindows()
    video.release()
    clip=mvp.VideoFileClip(os.path.join(current_directory, 'temp.avi'));
    clip.write_videofile(model_path = os.path.join(current_directory, 'Tracked.mp4'));
    os.remove(os.path.join(current_directory, 'temp.avi'))
```

This is the code for video formation from the given already annotated processed frames.

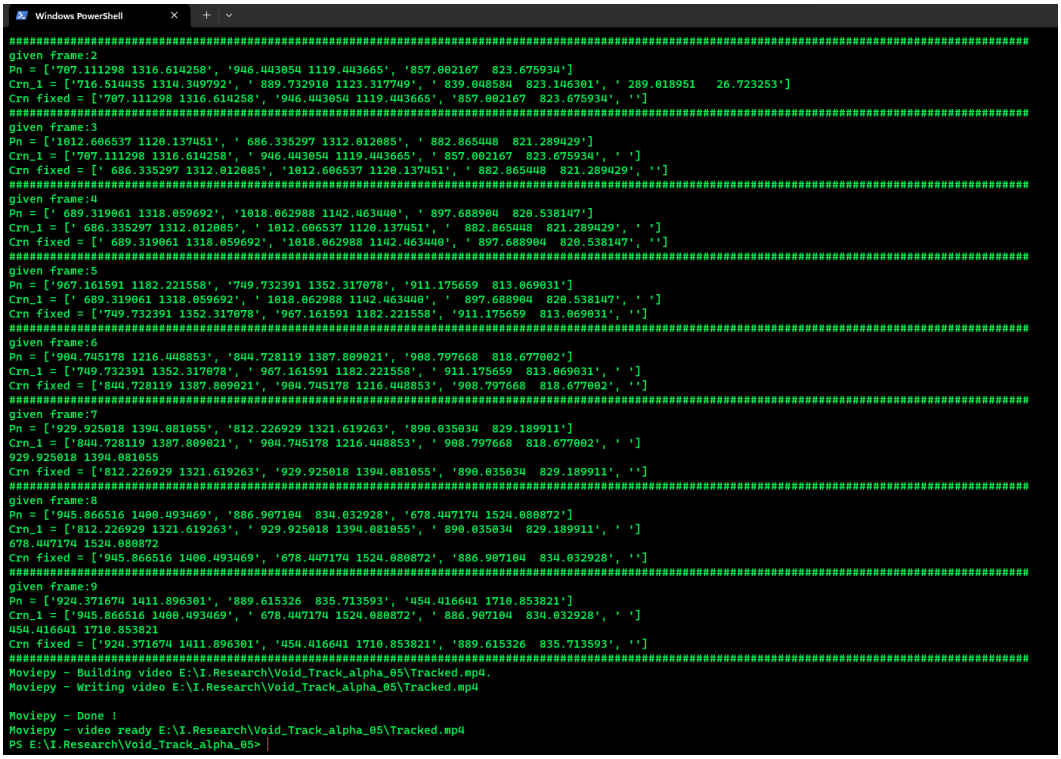
V2f.py > video

```python
import cv2
import os
def video():
    current_directory = os.path.dirname(os.path.abspath(__file__))
    vid_path = os.path.join(current_directory, 'vid/vid.mp4')
    vid = cv2.VideoCapture(vid_path)
    success,image = vid.read()
    c = 0
    while success:
      os.system('cls')
      cv2.imwrite('D:/I.Research/VoidWrk/Op/Frame%05d.png' % c, image)
      success,image = vid.read()
      c = c + 1
    os.system('cls')
    print('Break down done')
    return c
```

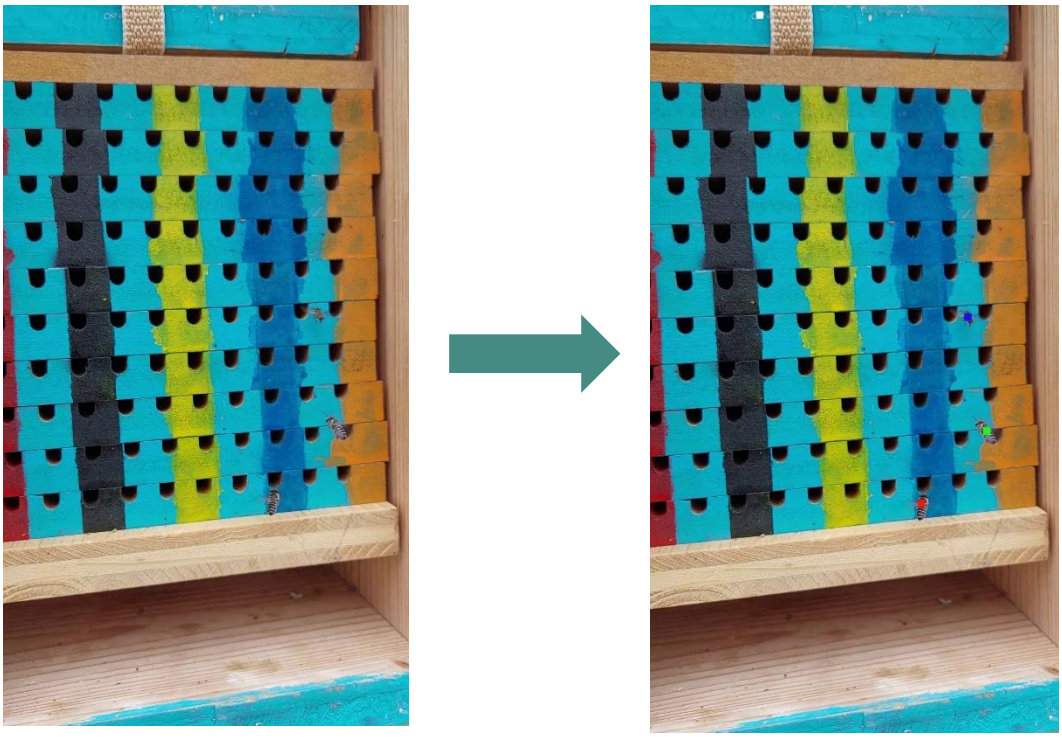
This is the code to break the input video to frames that can be then processed.

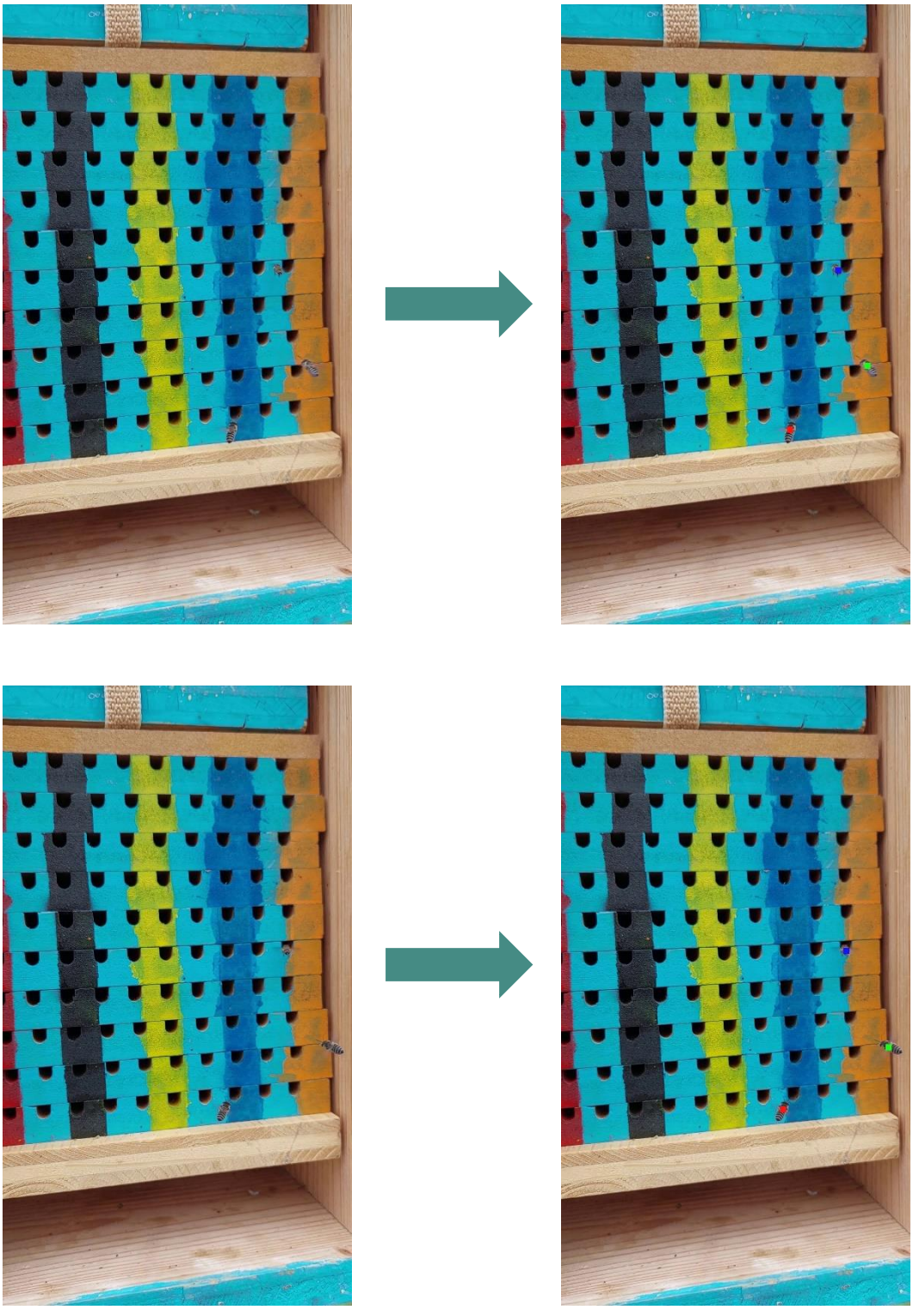
Running the code

Code Running:



As we can see the code runs properly using the windows PowerShell terminal and now let us look at the results.





As we can see the color labels follow the same bees in the first 3 frames. We could call this a success but we do run into a lot of trouble because of the low frame rate however this is how far we were able to reach by the end of the semester still looking for the how to proceed further with the problems we are facing with this iteration of void track.

Chapter IV

Future prospects

We still were not able to solve the issues caused by the low frame rate. For this we propose we were looking into the prospects of filling in the gaps in the frames using generative adversarial networks or GANs for short. Our plan thus far was to create the background of the frames free from insects using GANs and then implant the insects in the new frames without any insects by predicting their path using the Kalman filter which is why all the way at the start of the project we documented the Kalman filter from the ByteTrack repository which we intended to use as a base the missing frames filling code on.

We still have no answer for how to identify bees that leave and then re-enter the frames other than the earlier rudimentary hypothesis of using a probability model for the purposes of re-identifying the bee however we were not able to test that so it remains something that still needs a pre-alpha version and then so on.

Conclusion

To conclude this report the major outputs of this project turned out to be the documentation of the Kalman filter code from ByteTrack and the creating of Void track as an alpha which most certainly works for at least 3 identical objects in front of a variable background.

This is by no means a final product that would be used for industry application but this is definitely a good base to work on further to get to a beta version and then maybe an official release which can then be worked on by far far more talented people in the field of Object tracking.

References

[1] [Page 1] : https://github.com/ifzhang/ByteTrack

[2] [Page 2] : https://www.datature.io/blog/introduction-to-bytetrack-multi-object-tracking-by-associating-every-detection-box#:~:text=The%20main%20idea%20of%20BYTETrack,on%20their%20similarities%20with%20tracklets