Jeremy Ralls

2 March 2022

Collins Aerospace Tracking Algorithm Documentation

This document explains the execution of the object tracking algorithm through pseudo code and mathematical equations. 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.

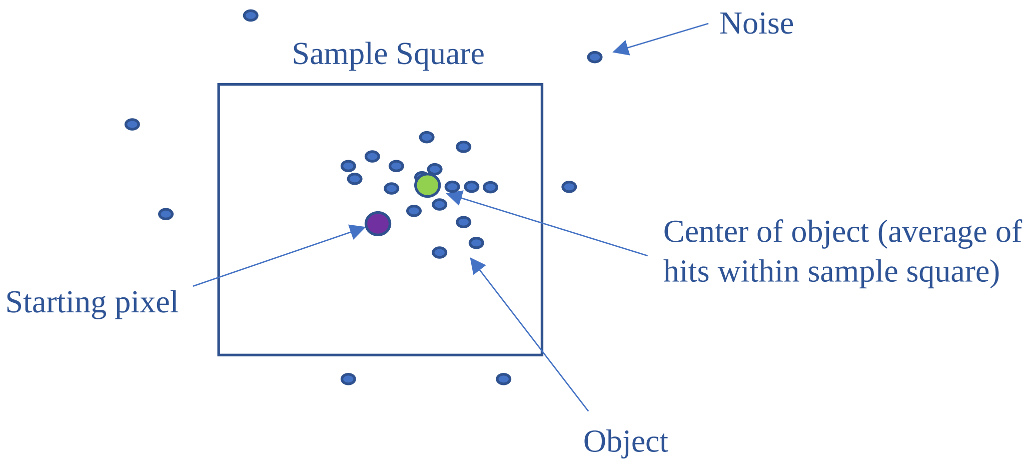
Candidate: the current hit being observed and evaluated as to whether or not it will become the bound.

tracking.py:

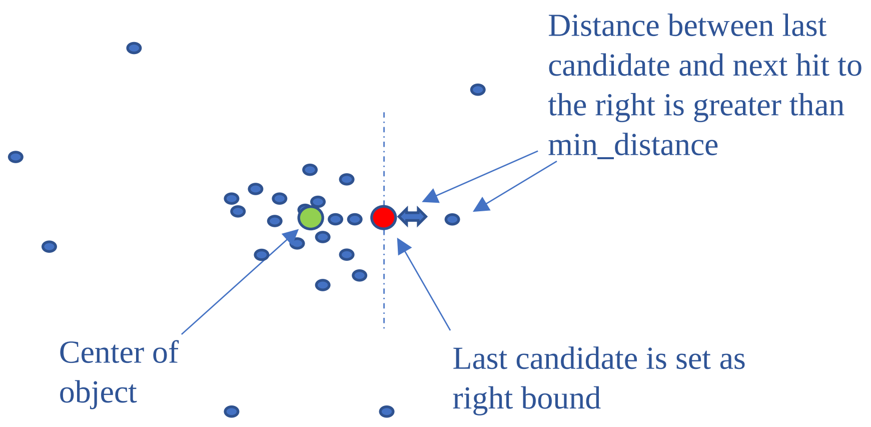
Description:

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.

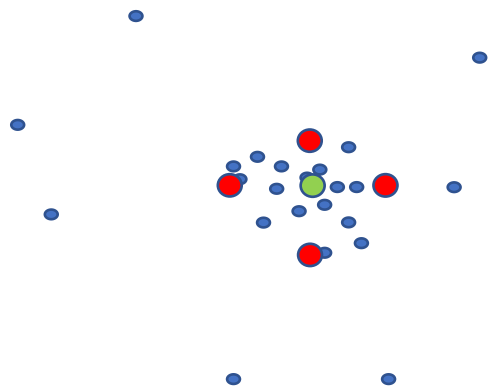
This function begins by taking the first hit in the provided arrays of hits and drawing a square around it with that first hit at the center. This is the sample square whose size is user defined. All hits within the sample square are averaged. This allows us to approximate the center of the object within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


Distance between last candidate and next hit to the right is greater than min_distance
Center of object
Last candidate is set as right bound

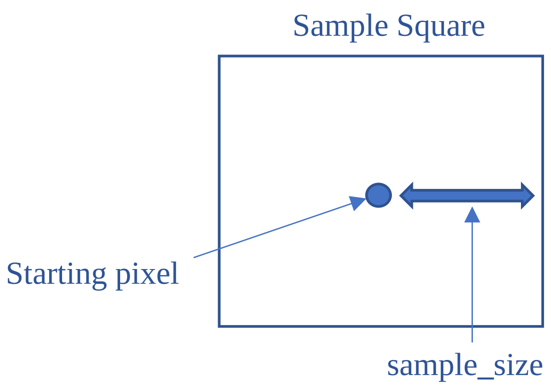
This is repeated for all bounds so that the edge for each side has been identified. Each of the red and green dots below are outputted as X and Y coordinates for the next function to use to draw a box.



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.

<u>Inputs:</u>

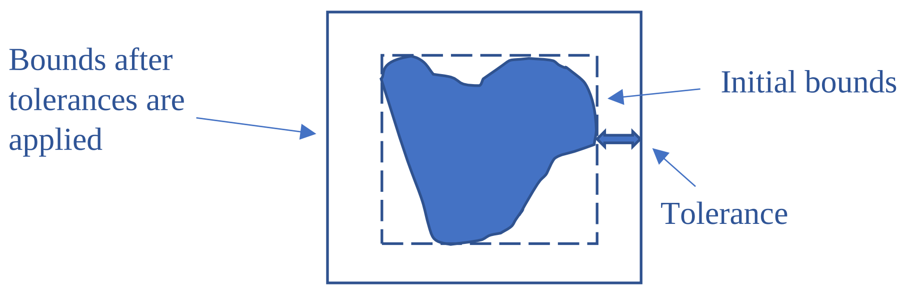
- sample_size (int): sets the size of the sample square from which the average is taken. This is the number of pixels from the starting pixel to the right, left, top, and bottom of the sample square. This equals half of one side of the square.



- 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.



- 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
```