CAMBOARD

Low-Price Interactive Board

Who am I ?

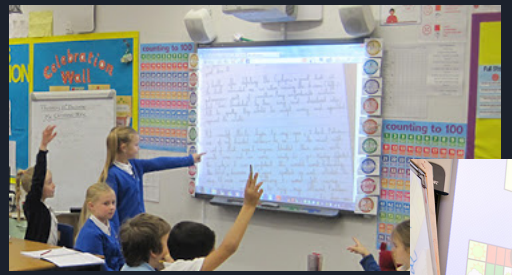
- Yaron Hay, 20 years old
- From Rosh Ha'Ayin
- Computer Science B.Sc.
  - I started my B.Sc. In the 11th grade
  - B.Sc. Student at Bar-Ilan University
- Last year's projects lab
  - Partnered HILMA

CS@BIU
המחלקה למדעי המחשב
אוניברסיטת בר-אילן

Bar-Ilan University
אוניברסיטת בר־אילן

Looking for an Idea

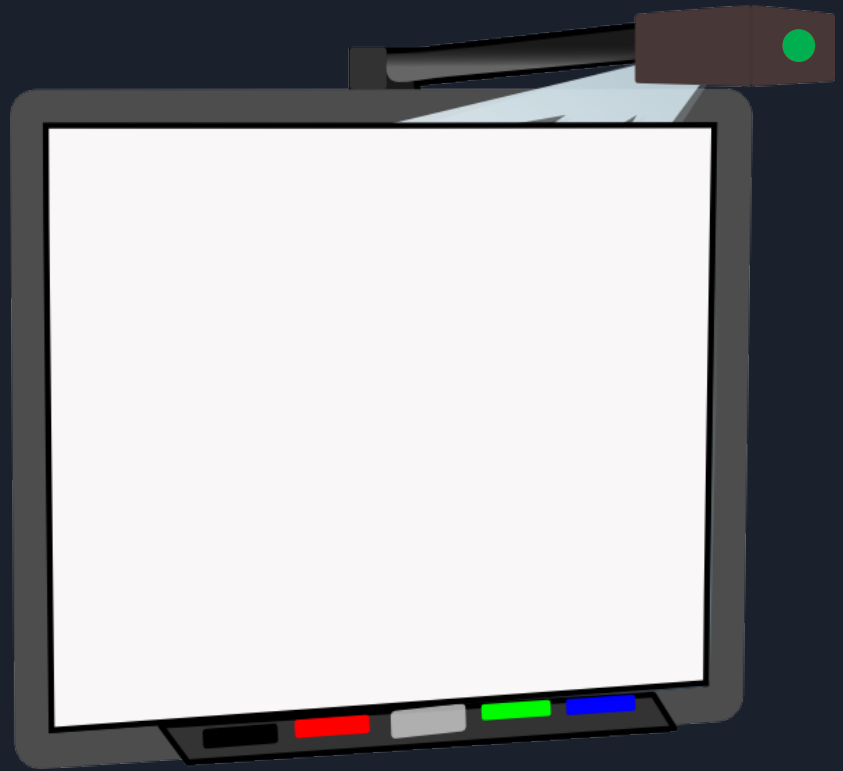
- School classrooms lack any modern technology
- Smart Boards are
  - Expensive
  - Hard to install without a technician
  - Requires special hardware
- Using two+ basic webcams
  - Exploits the existing projection system
  - Cheap
  - Easy to install



System Design

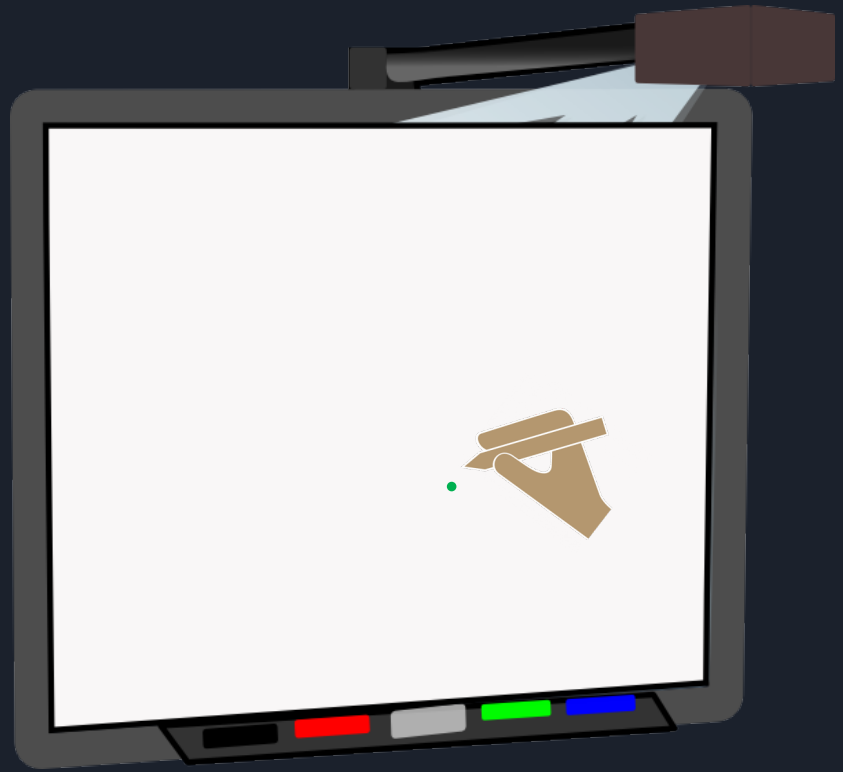
Sub-Problems

Track Contact with Board



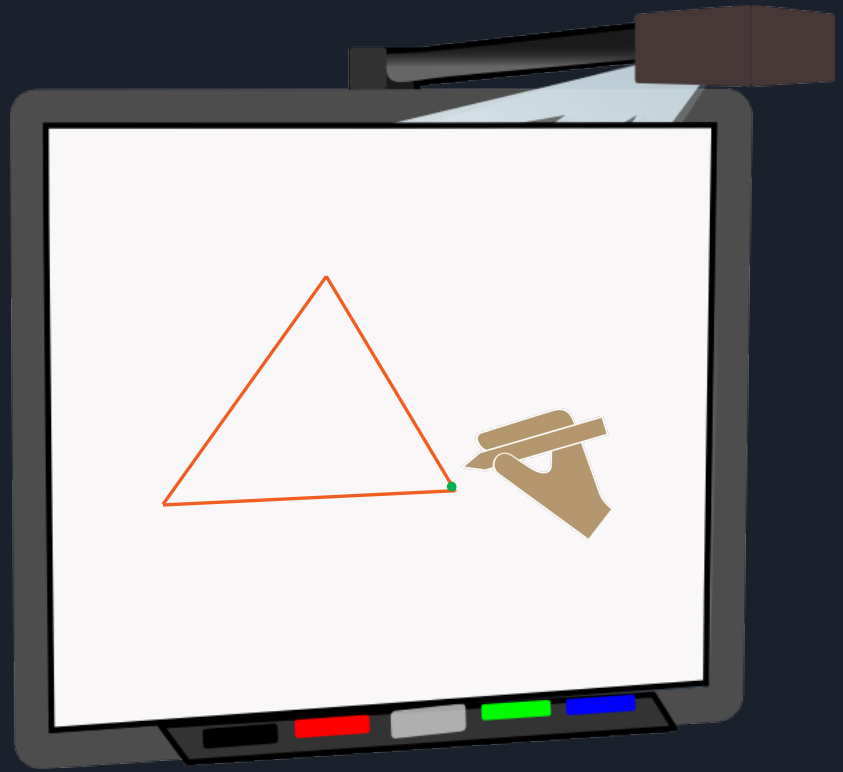
Sub-Problems

Track Pen Movement



Sub-Problems

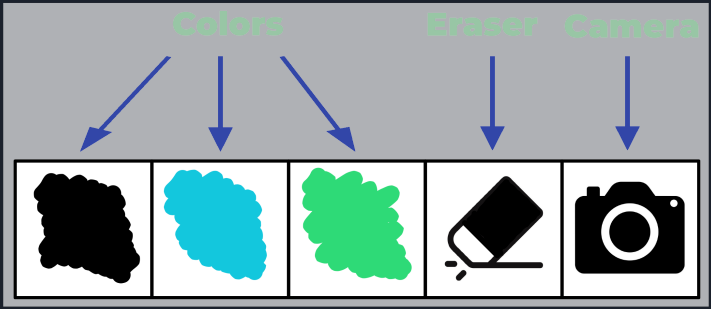
Translate Pen Movement to Shapes or other functionality



Sub-Problems

Changing Color & Erasing

- Use a menu with virtual buttons
  - Can create other functionality
- Utilizes the existing layout
  - No need for pen color detection
- Menu itself can be virtual
  - Present a dynamic menu with adapting buttons



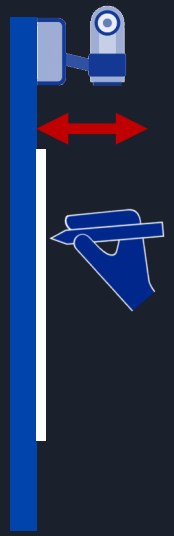
Camera Placement & Calibration

- Both cameras are fixed therefore, there is a possibility not to use automatic calibration
  - One-time manual calibration should be sufficient

System Implementation

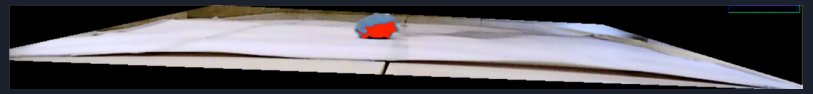
Touch Detection

- Position one camera at the top of the board and use it to follow when the pen touches it
- When the pen touches the board the camera will signal the system with a “board touched” event
- Utilizes color detection with contour scanning and area calculation
    - Choose some small distance $\Delta > 0$ from the board.
    - When the distance between the pen and the board $< \Delta$, then the pen is treated as if it is touching the board



Touch Detection

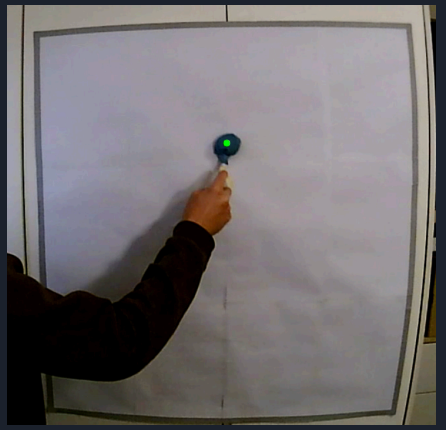
- Cutout only the section with the relevant distance
- Filter out all colors except the one chosen for the detection of the pen
- Find all contours to calculate the area of any detected objects, then filter out any noise to reduce the rate of false alarms
- Consider the pen touching board, if any large enough object is detected
- Problem: Camera must be precisely placed above the board



```
def detect_object_presence(
  frame, color_limits, thresh):

    mask = mask_by_color(
          frame, color_limits)
    contours, _ = cv2.findContours(
          mask, cv2.RETR_EXTERNAL,
          cv2.CHAIN_APPROX_SIMPLE)
    filtered = []

    for contour in contours:
        area = cv2.contourArea(contour)
        if area > thresh:
            filtered.append(contour)

    return len(filtered) > 0, filtered
```

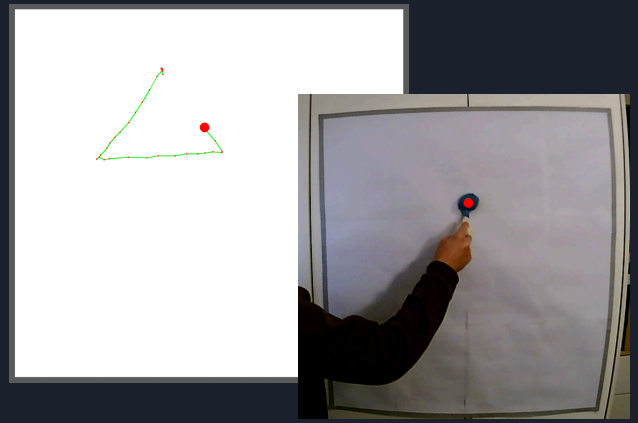
Pen Location Tracking

- Cutout only the section with the relevant area
- Filter out all colors except the one chosen for the detection of the pen
- Use moments calculation to estimate the location of the object
- Problem: Only ONE pen can present on the board in any given time



```
def detect_object_location(
          frame, color_limits):
    mask = mask_by_color(
          frame, color_limits)
    try:
       M = cv2.moments(mask)
       cX = int(M["m10"] / M["m00"])
       cY = int(M["m01"] / M["m00"])
       return True, (cX, cY)

    except ZeroDivisionError:
       return False, None
```

Pen Location Mapping



Assume:

- The detected pen center was at location (x, y) on the board
- The Board dims are $b_w \times b_h$
- The display dims are $d_w \times d_h$

The center will be mapped to a point on the display with the same relative location.

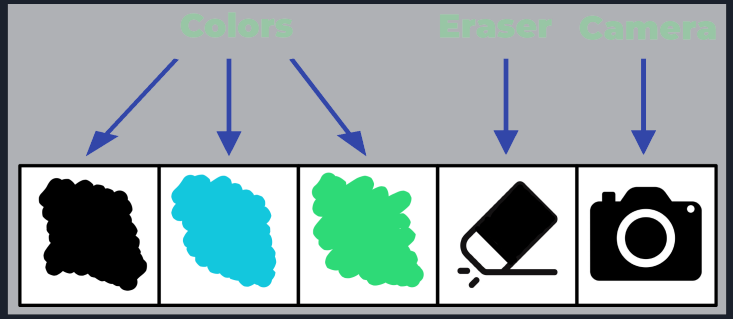
$$\left(\left\lfloor d_w \cdot \frac{x}{b_w} \right\rfloor, \left\lfloor d_h \cdot \frac{y}{b_h} \right\rfloor\right)$$

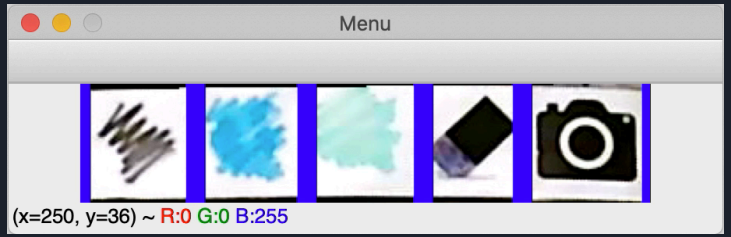
```
class Display:
  def __init__(self, size):
    self.__board = np.full(
        (1080, 1920, 3), 0xFF, np.uint8)
    cv2.rectangle(self.__board,
        (0, 0), (1920, 1080),
        GRAY, thickness=70)
    cv2.rotate(self.__board, cv2.ROTATE_90_CLOCKWISE,
        self.__board)
    self.f_width, self.f_height = size
    self.b_height, self.b_width, _ = self.board.shape

def place_point(self, x, y):
    new_x = self.__b_width * (x / self.__f_width)
    new_y = self.__b_height * (y / self.__f_height)
    return int(new_x), int(new_y)
```

Using a Menu

- A menu is an example for other types of functionality, that can be achieved while using the pen touch & location detection mechanism

- Each button of the menu is a region where the pen can be placed
- Pressing a button is performed when the pen touches the board on the specific region of that button





```
ALGORITHM
IF pen ON board:
  IF eraser mode IS OFF:
     APPEND pen.quards TO current_path
  ELSE:
     FOR path IN paths:
        FOR point IN path:
           IF point IS CLOSE TO path:
              REMOVE path

ELSE IF pen ON menu:
  selected := buttons.match(pen.quards)
  SWITCH selected {
     CASE eraser:
        SET eraser mode TO ON
     CASE snapshot_button:
        send(TAKE SCREENSHOT)
     CASE selected IN colors:
        SET paint_color TO COLOR(selected)
        SET eraser mode OFF
  }

ELSE:
  IF LEN(current_path) > 1:
     SUBMIT current_path
  ELSE:
     CLEAR current_path
```

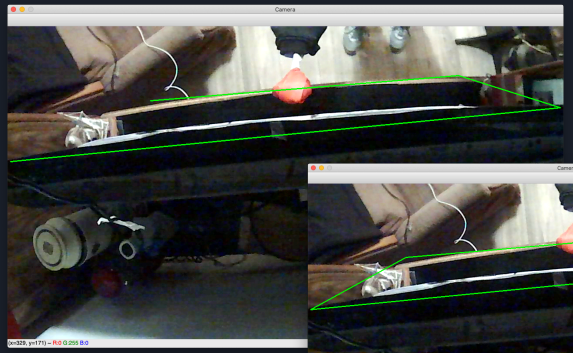
```
FN buttons::match(point) -> color|eraser {
  FOR border IN borders:
      IF point IN border:
         YIELD border
}
```

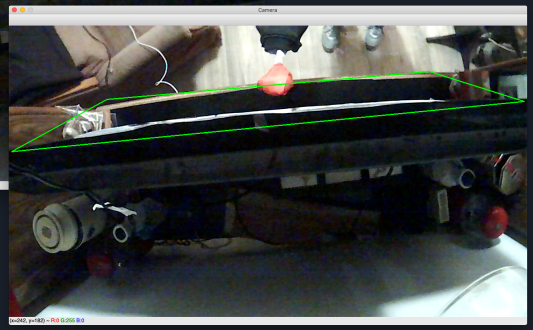
Core Algorithm

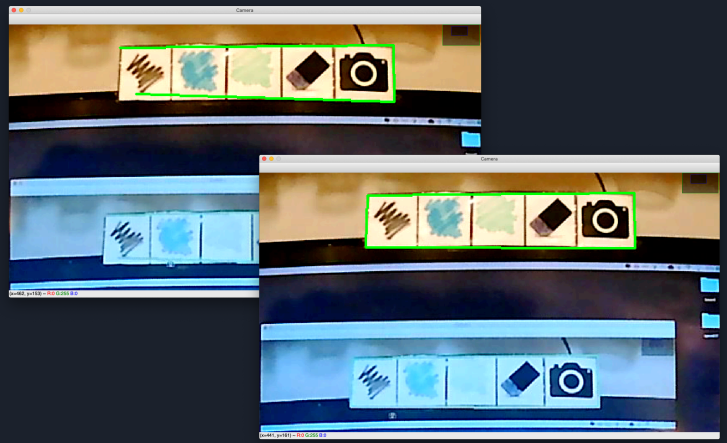

TOP_FILTERED
=88) ~ R:0 G:0 B:0
Board
67 B:39

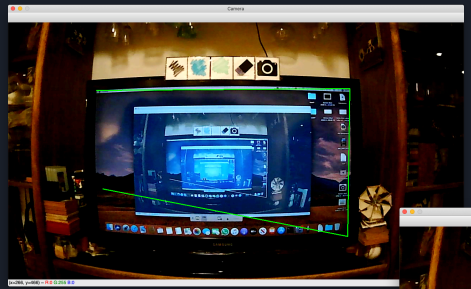
Calibration

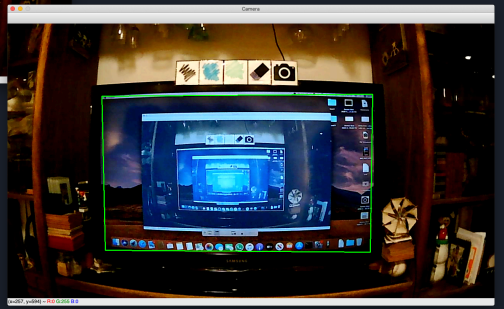
- The calibration process requires a manual cutout of the relevant part of the cameras' view for each of the board front, menu and board profile











Problems and Possible Improvements

False Positive Touch, Button Clicking

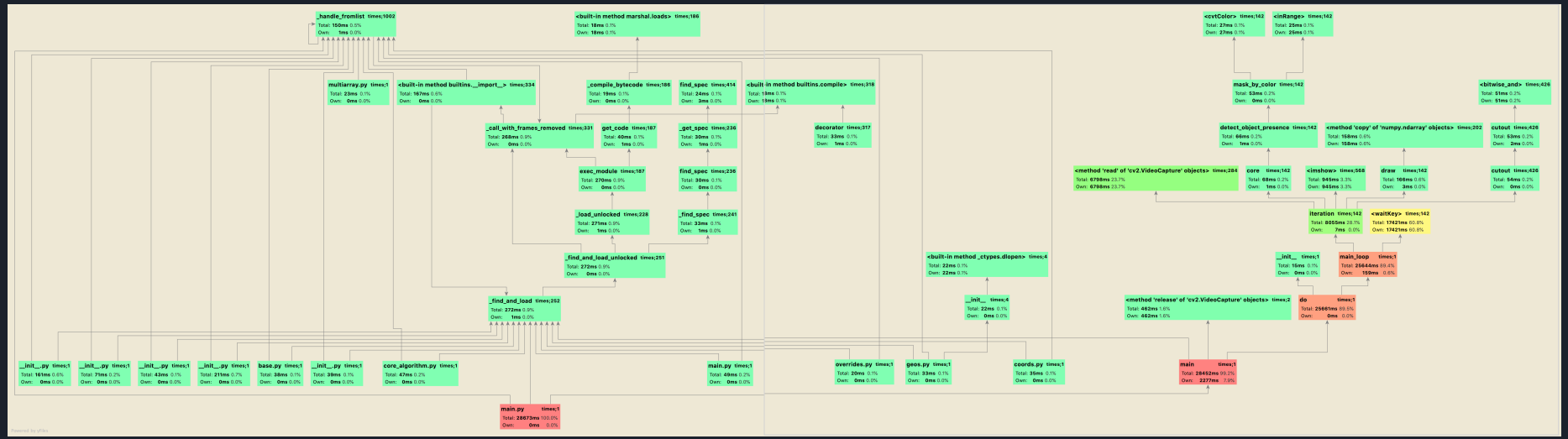
- What happens if in some frame, the pen isn't detected?
- The naive implementation of the menu buttons creates a problem where the "push" of the button causes a redundant repetition of the action
  - This is a problem that occurs when the operation "has" side-effects

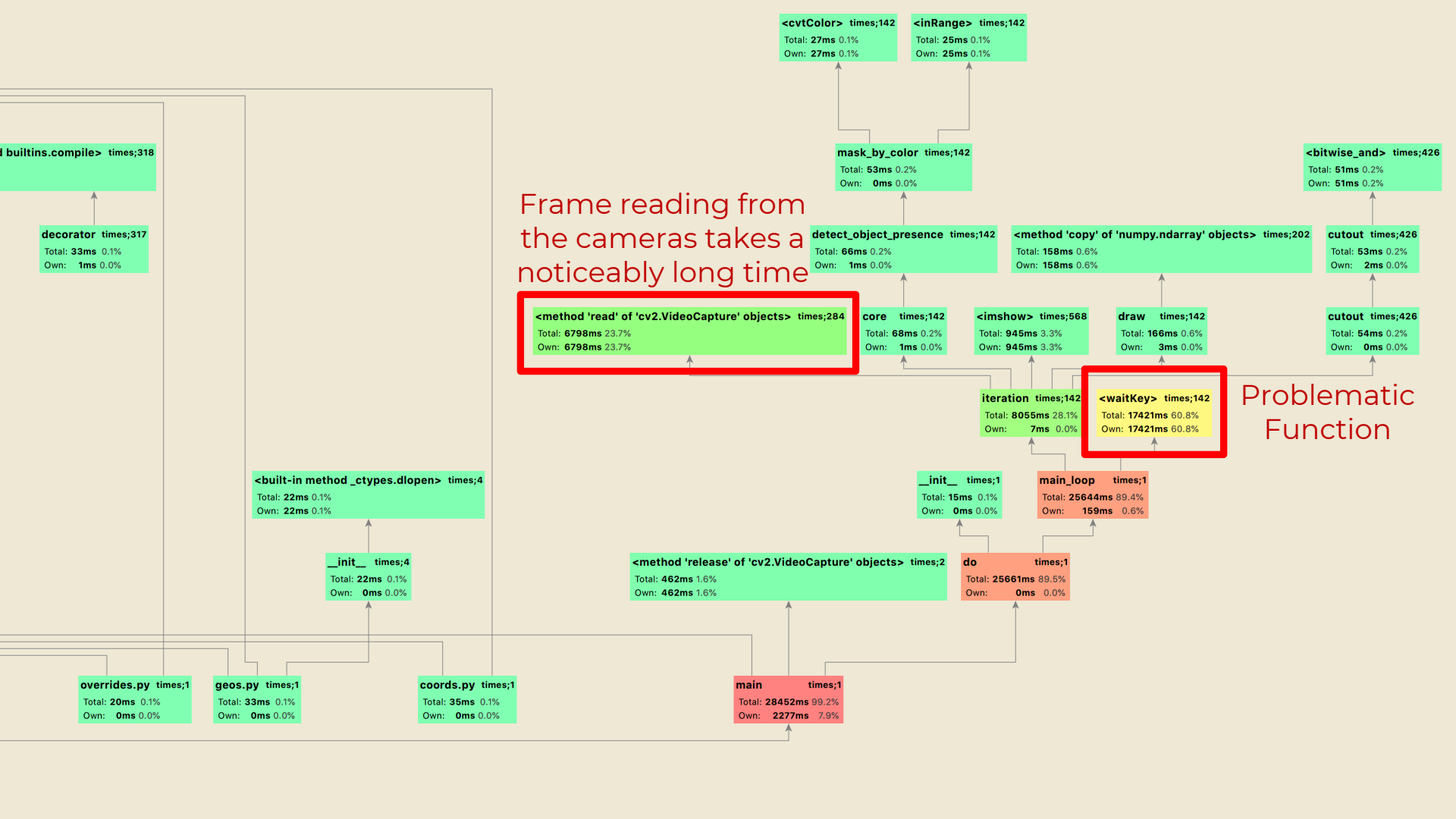
```
THRESHOLD = 0
COLOR_LIMITS = detector.RED_LIM
EPS = 5
BUTTON_MIN_TIME_DELTA = timedelta(seconds=3)
PEN_MIN_TIME_DELTA = timedelta(milliseconds=500)

screenshot_time_stamp = datetime.now()
board_pen_time_stamp = datetime.now()


def find_button(x, y, menu_buttons):
    for limits in menu_buttons:
        if x in range(*limits):
            return menu_buttons[limits]


def core(front_c, top_c, menu_c, displayer, state):
    global screenshot_time_stamp, board_pen_time_stamp
    ph: PointHolder = state.ph
    color = state.color
    is_eraser = color is None
    board_location, menu_location = None, None
    ##
    # Determine if object is touching
    ##
    touching, top_contours = detector.detect_object_presence(top_c, COLOR_LIMITS, THRESHOLD)


    if touching:
        on_board, board_location = detector.detect_object_location(front_c, COLOR_LIMITS)

        if on_board:
            if is_eraser:
                to_erase = []
                point = np.asarray(board_location)
                for path in ph.paths[:-1]:
                    points = [np.asarray(p) for p in path['points']]
                    if is_point_close_to_path(points, point, eps=EPS):
                        to_erase.append(path)
                ph.remove_paths(to_erase)

            else:
                board_pen_time_stamp = datetime.now()
                ph.add_to_path(board_location)

        else:
            on_menu, menu_location = detector.detect_object_location(menu_c, COLOR_LIMITS)
            if on_menu:
                button = find_button(*menu_location, state.menu['buttons'])
                if button == 'camera':
                    now = datetime.now()
                    delta = now - screenshot_time_stamp
                    screenshot_time_stamp = now

                    if delta > BUTTON_MIN_TIME_DELTA:
                        print("in")
                        subject = 'Your Boardshot'
                        content = 'Here is you boardshot - the screenshot of your smart board, ' \
                                  f'taken on {now.strftime("%c")}:'
                        address = 'yaron.hay@live.biu.ac.il'
                        file_name = f"Boardshot{now.strftime('%Y_%m_%d_%H_%M_%S')}.png"

                        os.system(f"screencapture {file_name}")
                        print(content)
                        os.system(f"./core/email.sh {address} \"{subject}\" \"{content}\" {file_name}")
                else:
                    state.setcolor(button)
```

Other

- Color detection
- The run time depends on the number of points drawn
- Low frame rate → Low resolution

Profiling

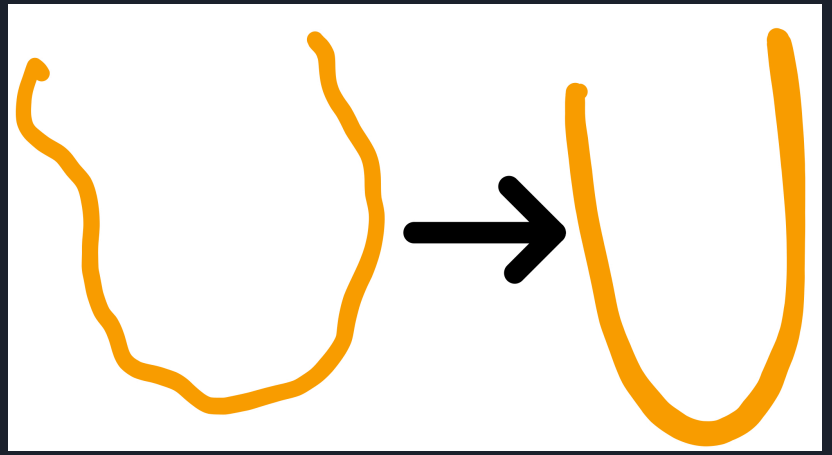

_handle_fromlist times;1002
Total: 150ms 0.5%
Own: 1ms 0.0%
<built-in method marshal.loads> times;186
Total: 18ms 0.1%
Own: 18ms 0.1%
<cvtColor> times;142
Total: 27ms 0.1%
Own: 27ms 0.1%
<inRange> times;142
Total: 25ms 0.1%
Own: 25ms 0.1%
multiarray.py times;1
Total: 23ms 0.1%
Own: 0ms 0.0%
<built-in method builtins.__import__> times;334
Total: 167ms 0.6%
Own: 0ms 0.0%
_compile_bytecode times;186
Total: 19ms 0.1%
Own: 0ms 0.0%
find_spec times;414
Total: 24ms 0.1%
Own: 3ms 0.0%
<built in method builtins.compile> times;318
Total: 13ms 0.1%
Own: 13ms 0.1%
mask_by_color times;142
Total: 53ms 0.2%
Own: 0ms 0.0%
<bitwise_and> times;626
Total: 51ms 0.2%
Own: 51ms 0.2%
_call_with_frames_removed times;331
Total: 268ms 0.9%
Own: 0ms 0.0%
get_code times;187
Total: 40ms 0.1%
Own: 1ms 0.0%
_get_spec times;236
Total: 30ms 0.1%
Own: 1ms 0.0%
decorator times;317
Total: 33ms 0.1%
Own: 1ms 0.0%
detect_object_presence times;142
Total: 66ms 0.2%
Own: 1ms 0.0%
<method 'copy' of 'numpy.ndarray' objects> times;202
Total: 168ms 0.6%
Own: 158ms 0.6%
cutout times;426
Total: 53ms 0.2%
Own: 2ms 0.0%
exec_module times;187
Total: 270ms 0.9%
Own: 0ms 0.0%
find_spec times;236
Total: 30ms 0.1%
Own: 0ms 0.0%
<method 'read' of 'cv2.VideoCapture' objects> times;284
Total: 6798ms 23.7%
Own: 6798ms 23.7%
core times;142
Total: 68ms 0.2%
Own: 1ms 0.0%
<imshow> times;568
Total: 945ms 3.3%
Own: 945ms 3.3%
draw times;142
Total: 166ms 0.6%
Own: 3ms 0.0%
cutout times;626
Total: 54ms 0.2%
Own: 0ms 0.0%
_load_unlocked times;228
Total: 271ms 0.9%
Own: 1ms 0.0%
_find_spec times;241
Total: 33ms 0.1%
Own: 1ms 0.0%
iteration times;142
Total: 8055ms 28.1%
Own: 7ms 0.0%
<waitKey> times;142
Total: 17421ms 60.8%
Own: 17421ms 60.8%
_find_and_load_unlocked times;251
Total: 272ms 0.9%
Own: 0ms 0.0%
<built-in method _ctypes.dlopen> times;4
Total: 22ms 0.1%
Own: 22ms 0.1%
__init__ times;1
Total: 15ms 0.1%
Own: 0ms 0.0%
main_loop times;1
Total: 26644ms 89.4%
Own: 159ms 0.6%
_find_and_load times;262
Total: 272ms 0.9%
Own: 1ms 0.0%
__init__ times;4
Total: 22ms 0.1%
Own: 0ms 0.0%
<method 'release' of 'cv2.VideoCapture' objects> times;2
Total: 462ms 1.6%
Own: 462ms 1.6%
do times;1
Total: 25661ms 89.9%
Own: 0ms 0.0%
__init__.py times;1
Total: 161ms 0.6%
Own: 0ms 0.0%
__init__.py times;1
Total: 71ms 0.2%
Own: 0ms 0.0%
__init__.py times;1
Total: 43ms 0.1%
Own: 0ms 0.0%
__init__.py times;1
Total: 211ms 0.7%
Own: 0ms 0.0%
base.py times;1
Total: 38ms 0.1%
Own: 0ms 0.0%
__init__.py times;1
Total: 39ms 0.1%
Own: 0ms 0.0%
core_algorithm.py times;1
Total: 47ms 0.2%
Own: 0ms 0.0%
main.py times;1
Total: 49ms 0.2%
Own: 0ms 0.0%
overrides.py times;1
Total: 20ms 0.1%
Own: 0ms 0.0%
geos.py times;1
Total: 33ms 0.1%
Own: 0ms 0.0%
coords.py times;1
Total: 35ms 0.1%
Own: 0ms 0.0%
main times;1
Total: 28452ms 99.2%
Own: 2277ms 7.9%
main.py times;1
Total: 28673ms 100.0%
Own: 0ms 0.0%


<cvtColor> times;142
Total: 27ms 0.1%
Own: 27ms 0.1%
<inRange> times;142
Total: 25ms 0.1%
Own: 25ms 0.1%
mask_by_color times;142
Total: 53ms 0.2%
Own: 0ms 0.0%
<bitwise_and> times;426
Total: 51ms 0.2%
Own: 51ms 0.2%
Frame reading from the cameras takes a noticeably long time
detect_object_presence times;142
Total: 66ms 0.2%
Own: 1ms 0.0%
<method 'copy' of 'numpy.ndarray' objects> times;202
Total: 158ms 0.6%
Own: 158ms 0.6%
cutout times;426
Total: 53ms 0.2%
Own: 2ms 0.0%
<method 'read' of 'cv2.VideoCapture' objects> times;284
Total: 6798ms 23.7%
Own: 6798ms 23.7%
core times;142
Total: 68ms 0.2%
Own: 1ms 0.0%
<imshow> times;568
Total: 945ms 3.3%
Own: 945ms 3.3%
draw times;142
Total: 166ms 0.6%
Own: 3ms 0.0%
cutout times;426
Total: 54ms 0.2%
Own: 0ms 0.0%
iteration times;142
Total: 8055ms 28.1%
Own: 7ms 0.0%
<waitKey> times;142
Total: 17421ms 60.8%
Own: 17421ms 60.8%
Problematic Function
__init__ times;1
Total: 15ms 0.1%
Own: 0ms 0.0%
main_loop times;1
Total: 25644ms 89.4%
Own: 159ms 0.6%
<built-in method _ctypes.dlopen> times;4
Total: 22ms 0.1%
Own: 22ms 0.1%
__init__ times;4
Total: 22ms 0.1%
Own: 0ms 0.0%
<method 'release' of 'cv2.VideoCapture' objects> times;2
Total: 462ms 1.6%
Own: 462ms 1.6%
do times;1
Total: 25661ms 89.5%
Own: 0ms 0.0%
overrides.py times;1
Total: 20ms 0.1%
Own: 0ms 0.0%
geos.py times;1
Total: 33ms 0.1%
Own: 0ms 0.0%
coords.py times;1
Total: 35ms 0.1%
Own: 0ms 0.0%
main times;1
Total: 28452ms 99.2%
Own: 2277ms 7.9%

Suggested Features

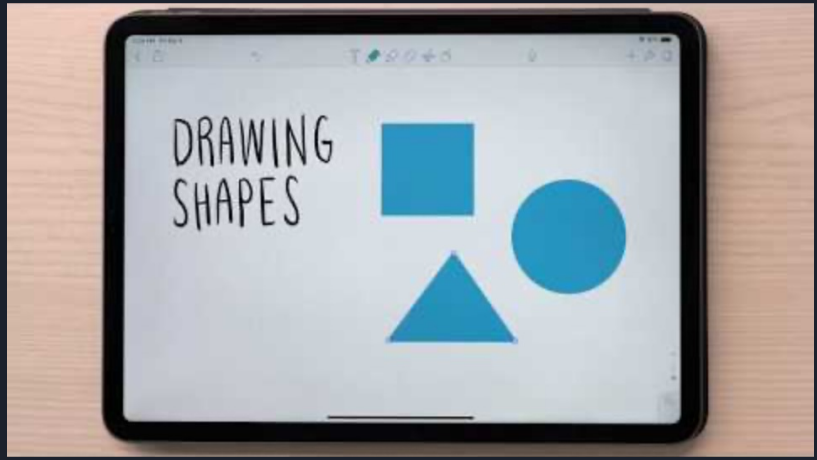
Line Smoothing

- The lines that were drawn, don't always look very good and points aren't distributed uniformly
  - Process the path drawn and create a similar path with its points more evenly distributed
  - Use a different interpolation than linear interpolations



Shape Detection

- After drawing the contour of the shape hold the pen in place for a moment
  - When held in place, transfer the path of pen location samples to a classifier to detect the shape as shape type, location, size, and color
  - After the classifier successfully detects the shape, replace the path of pen location samples with a new drawing of the shape



Auto Calibration

- Manual Calibration can be tiresome
- Cropping the frames using a quadrilateral cutout may not be as accurate due to the angle of the camera‘s position
  - The rectangular shape may be distorted

- Idea: use the projector to display some unique colors / patterns so that the all pixels that are located on the board can be identified one by one
  - By finding all of them, one may create a mask, instead of using a quadrilateral shaped mask

Adapt the point mapping strategy:

- Board dims are $\mathrm{w_b}(i) \times h_b(j)$
- Display dims are $d_w \times d_h$

If the center is at row i and column j, then it will be mapped to

$$\left(\left\lfloor d_w \cdot \frac{x}{\mathrm{w_b}(i)} \right\rfloor, \left\lfloor d_h \cdot \frac{y}{h_b(j)} \right\rfloor\right)$$

Thank You!