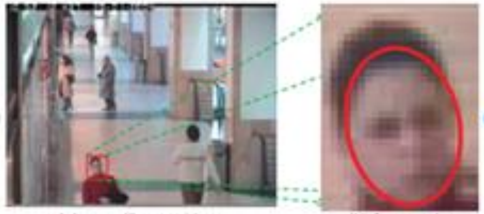
Smart Surveillance Video Control System

by selective super resolution and video auto analysis

201533308 이찬하
201735824 나민수
201735838 방준석
201835425 김수빈

CONTENT

Project Remind – Idea Proposal



(a) surveillance video (b) face region

Need : High - resolution image to use various social problem

→ if quality low : can't specify object

→ if higher : more storage needed ⇒ high maintenance cost

In field : Storing video clip for certain period & discard in the end of it

→ hard to find some videos when we want to

NTSC: Recording Variable: 30fps			Surveillance Hard Drive Capacity		
			1TB	2TB	3TB
176 x 120	Low Quality	# Days	230	460	690
352 x 240	↓	# Days	88	176	264
704 x 480		# Days	28	56	84
1280 x 1024	High Quality	# Days	8	16	24

⇒ efficiency & video quality of storage space should be coordinated

Project Remind – Idea Proposal

As Is

To Be

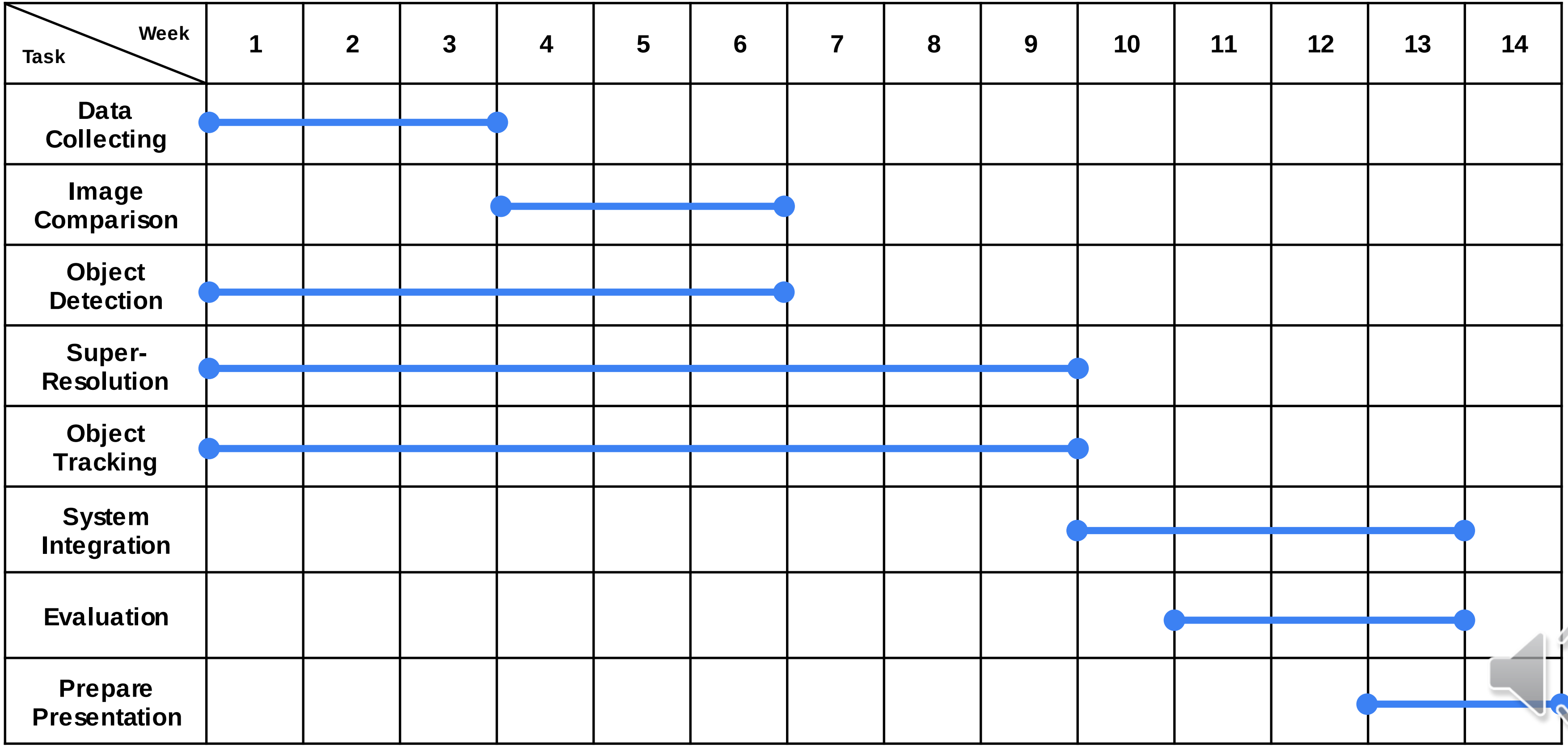
- **Increased fatigue** of administrators
- **High failure** of proper detection
- **Lots of cameras** assigned per administrator
- Save video **quality supported by cameras** in system

- Take videos with low-definition cameras but store them by **improving quality** video
- **Auto-detection** of specific targets

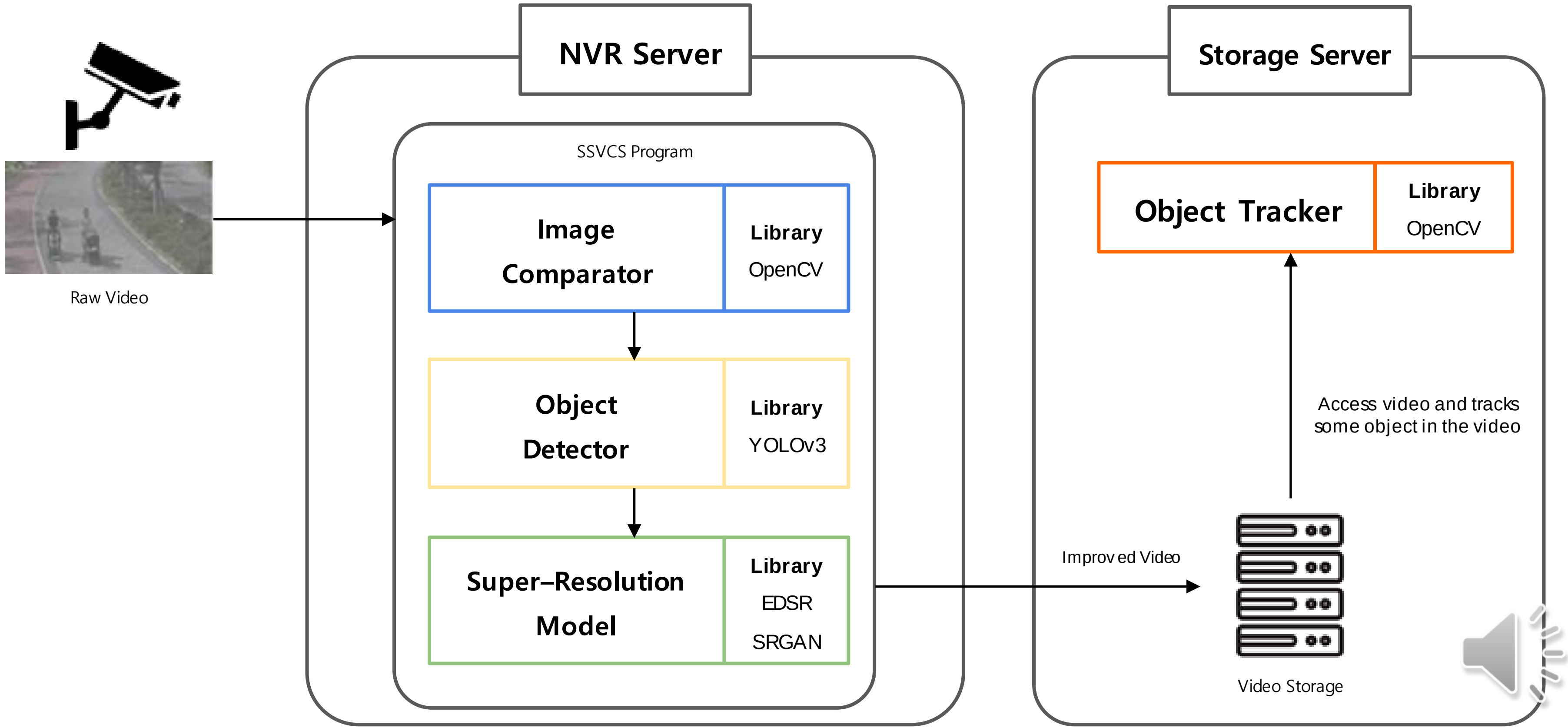
Project Remind - Member Roles

<table>
<tr><th>Name</th><th colspan="2">Role</th></tr>
<tr><td>나민수</td><td>Object Tracking
Presentation</td><td rowspan="2">System Integration</td></tr>
<tr><td>방준석</td><td>Object Detection
Super-resolution</td></tr>
<tr><td>김수빈</td><td>Data Collecting
Image Comparison</td><td rowspan="2">Evaluation & Analysis
Prepare Presentation</td></tr>
<tr><td>이찬하</td><td>Super-resolution</td></tr>
</table>

Project Remind - Timeline



Implementation - System Flow



Implementation – Image Comparison

- **Implementation goal:**

  To figure out the change in video scene

- **Opensource Library that is used:**

  OpenCV

- **Main Function Description**

  1. Split input video to frames
  2. Read criteria image
  3. Calculate MSE and SSIM
  4. Check threshold and collect high-difference frames
  5. Send these frames' data to Object Detection Module

Algorithm 1: Image Comparison

Input: Video file *video*, threshold hyperparameters T_{MSE}, T_{SSIM}

Output: data of high-difference frames $F = \{index, MSE, SSIM, image\}$

n <- the number of frames
for *(i = 0; i < n; i++)* **do**
  write image 'frame *i*'
origin <- frame 0
for *(i = 0; i < n; i++)* **do**
  $MSE_i = \frac{1}{n_{pixel}} \sum_{j=0}^{n_{pixel}} (frame_{i,j} - origin_{i,j})$
  $SSIM_i = calculate\ SSIM$
  if $(MSE_i > T_{MSE}$ and $SSIM_i < T_{SSIM})$ **do**
    $F_i = \{i, MSE_i, SSIM_i, image_i\}$
return F

Implementation – Object Detection

- **Implementation goal:**

  To find object's location for selective super-resolution

- **Opensource Library that is used:**

  YOLOv3

- **Main Function Description**

  1. Read frame image
  2. Detect objects using yolo
  3. Visualize objects' boundary
  4. Calculate objects' location coordinates

Algorithm 2: Object Detection

Input: Image data img
Output: Image data with box img, Object location $xy = \{ x_\alpha, y_\alpha, x_\beta, y_\beta \}$; α is top left, β is bottom right.

img <- $resized\ image\ from\ img$
$outs = Detection\ output\ from\ YOLO$
$boxes$ <- empty list
for $each\ out\ in\ outs$ **do**
  $w = out.width$
  $h = out.height$
  $x = \frac{out.x_{center}}{2}$
  $y = \frac{out.y_{center}}{2}$
  $boxes_{out} = [x, y, w, h]$
  img <- $Draw\ image\ with\ boxes$
  $xy_{out} = \{x, y, x + w, y + h\}$
return img, xy

Implementation – Super-Resolution

- **Implementation goal:**

  To improve video resolution on location of objects that is found

- **Opensource Library that is used:**

  EDSR, SRGAN

- **Main Function Description**

  1. Cut the image to background and object
  2. Improve resolution of object's image
  3. Combine background and improved image
  4. Generate improved video

Algorithm 3: Super-Resolution

Input: Image data $frames$, objects' location $xy = \{x_\alpha, y_\alpha, x_\beta, y_\beta\}$; α is top left, β is bottom right, SR deep learning model $model$
Output: Improved Image $frame_{improved}$

$model$ <- pre-assigned model(EDSR or SRGAN)
for $each\ frame\ in\ frames$ **do**
  $objects_{frame} =$ $extract\ objects'image\ from\ frame$
  $objects\_SR_{frame} =$ $Improve\ object\ image's\ resolution\ using\ model$
  $frame_{improved} =$ $Combine\ background\ and\ improved\ image$
return $frame_{improved}$

Implementation – Object Tracking

- **Implementation goal:**

  To track the movement of selected object for video analysis

- **Opensource Library that is used:**

  OpenCV

- **Main Function Description**

  1. Select video in storage
  2. Select time to search an object
  3. Select object to track
  4. Visualize optical flow track line

Algorithm 4: Object Tracking

Input: Video file $video$, starting time T_{start}
Output: Tracking video $V_{tracking}$

$video.time$ <- T_{start}
$frame_{start} = Read\ first\ frame$
$boundary$ <- boundary from user's mouse input
$points = generate\ points\ on\ boundary$
$frame_{old}$ <- $frame_{start}$
while $video\ is\ opened$ **do**
  $frame_{new} = Read\ current\ frame$
  for $each\ point\ in\ points$ **do**
    $vector_{point} = point_{old} - point_{new}$
    $frame_{new}$ <- $Draw\ frame\ with\ vector$
  $V_{tracking}$ <- $Add\ new\ frame\ in\ result\ video$
  $frame_{old} = frame_{new}$
return $V_{tracking}$

Implementation – System Protocol

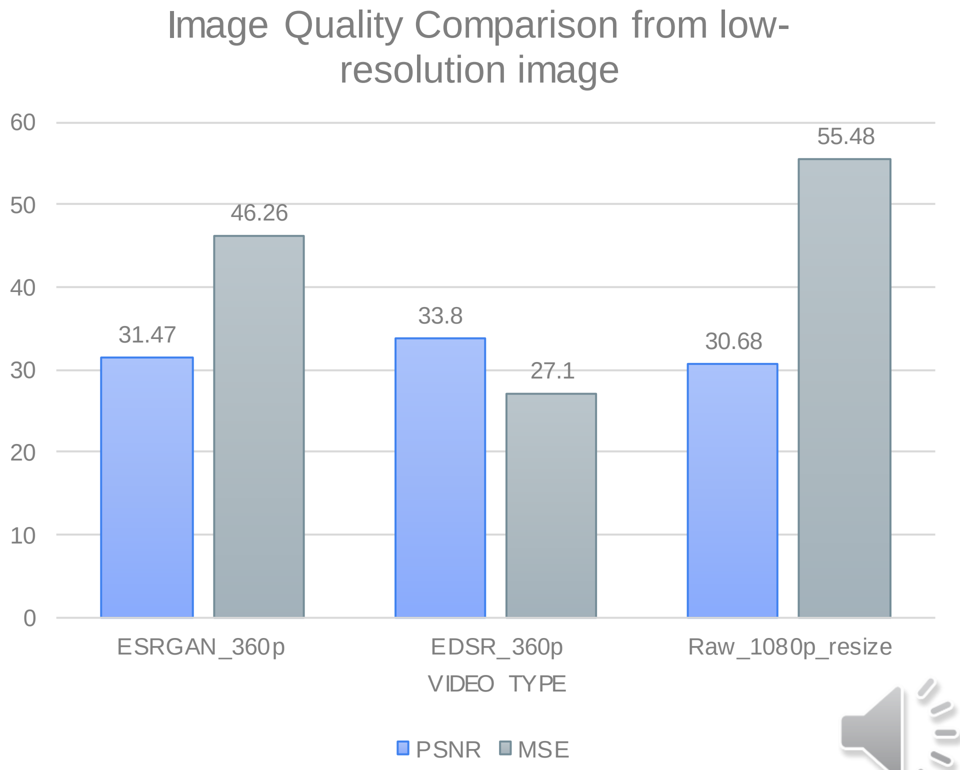
Message Block	Message Description
VIB	Describe target video information
FIB1	Describe basic frame information
FIB2	Describe frame information including MSE and SSIM
FIB3	Describe frame information including objects' location
FIB4	Describe frame information after SR
FIB5	Describe frame information including optical flow vector
DIB	Describe Deep learning model information

Implementation - Project Demo

Evaluation - Resolution Analysis

- The right chart shows the average PSNR value and MSE value for each image quality compared to the same resized low-resolution image.

- The average PSNR values of EDSR and SRGAN are 33.8 and 31.47, respectively, higher than 30.68 of a simple high-resolution image. The higher the PSNR value, the smaller the difference from the original image, so it shows that our project improves the quality close to the high-resolution image.



Evaluation - File Capacity Analysis

- The chart on the right shows the difference in file capacity between images of different image quality having the same length and same frame size.

- Each of EDSR and SRGAN has a file size of 14.9 and 26.9, which is close to 27.5, the size of a low-resolution image file, and is significantly different from the high-resolution image of 48.7. This can be seen as meaning that the method of changing only a part of it to high-resolution sufficiently increases the efficiency of storage space.

Future work

- **What we need to improve on our project**:
    1. Detect only person and car
    2. Improve just saved video(not apply real-time video)
    3. Image Resize Issue
    4. Incomplete video quality and storage efficiency improvement
    5. Incomplete Program UI

- **Project Improvement Plan**:
    1. Improve the video quality close to original and solve image resize issue.
    2. Improve the storage efficiency using video compression methods.
    3. Develop real-time super-resolution method and apply to real-time video.
    4. Expand the range of object types that can be detected.
    5. Decide SR deep learning model that will be applied to our project.
    6. Develop GUI module.

Appendix: Code Capture

```python
def compare_image(video):
    frame = []
    mse_val = []
    ssim_val = []
    m = []
    s = []

    origin = cv2.imread('./%s/origin.png' % video)
    origin = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
    for j in range(0, video_cap(video)):
        if j == 0:
            img = cv2.imread('./%s/frame%s.png' % (video, j))
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

            m.append(mse(origin, img))
            s.append(ssim(origin, img))
        else:
            pv_img = cv2.imread('./%s/frame%s.png' % (video, j-1))
            pv_img = cv2.cvtColor(pv_img, cv2.COLOR_BGR2GRAY)

            img = cv2.imread('./%s/frame%s.png' % (video, j))
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

            m.append(mse(pv_img, img))
            s.append(ssim(pv_img, img))

    for j in range(0, video_cap(video)):
        # mse와 ssim 출력

        print("%s | mse : %f    ssim : %f" % (j, m[j], s[j]))
        print()
        # 임의로 정한 mse & ssim 조건
        # mse 는 클수록 오차가 심함.
        # ssim은 작을수록 오차가 심함.
        if (m[j] > 5) and (s[j] < 0.98):
            frame.append(j)
            mse_val.append(m[j])
            ssim_val.append(s[j])

    return frame, mse_val, ssim_val, video_cap(video)
```

1. Image Comparison Algorithm

```python
def yolo(img):
    net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
    with open("coco.names", "r") as f:
        classes = [line.strip() for line in f.readlines()]
    layer_names = net.getLayerNames()
    output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]
    colors = np.random.uniform(0, 255, size=(len(classes), 3))

    # 이미지 가져오기
    img = cv2.resize(img, None, fx=1, fy=1)
    height, width, channels = img.shape

    # Detecting objects
    blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)

    # 정보를 화면에 표시
    class_ids = []
    confidences = []
    boxes = []
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                # Object detected
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                # 좌표
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                if (w*h) >= 1000:
                    boxes.append([x, y, w, h])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)

    # 노이즈 제거
    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

    xy = []
    print("YOLO 좌표")
    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            color = colors[i]
            if label == 'person' or 'car':
                # 박스 그림.
                cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
                xy.append([x, y, x + w, y + h])
                print(x, y, x + w, y + h)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

    return img, xy
```

2. YOLO Object Detection Algorithm

Appendix: Code Capture

```python
def crop(frame, x_start, y_start, x_end, y_end, IMG_TRIM):
    img_trim = frame[y_start : y_end, x_start : x_end]
    width, height, channel = img_trim.shape
    print("cropped shape")
    print(width, height, channel)

    IMG_TRIM.append(img_trim)
    return IMG_TRIM


def add(checked_frame, SR_img, x_start, y_start, x_end, y_end):
    print("add function")
    print(x_start, y_start, x_end, y_end)  # yolo 박스 좌표

    # 높이, 너비 순서
    h, w, c = SR_img.shape  # 박스 크기
    print(h, w, c)
    checked_frame[y_start: y_end, x_start: x_end] = SR_img

    cv2.waitKey(0)
    cv2.destroyAllWindows()

    return 0


def SR(img_trim, SR_img):
    sr = cv2.dnn_superres.DnnSuperResImpl_create()
    path = "EDSR_x4.pb"
    sr.readModel(path)

    sr.setModel("edsr", 4)
    result = sr.upsample(img_trim)

    result = cv2.resize(result, None, fx=0.25, fy=0.25)
    SR_img.append(result)
    return SR_img
```

3. Super-resolution Algorithm

```python
def tracking_video(vfile):
    global panel, isPlay, timer

    if vfile == "":
        messagebox.showinfo(title="Video Load Error", message="Please select video before tracking.")
        return

    cap = cv.VideoCapture(vfile)
    start_time = timer.get() * 1000
    cap.set(cv.CAP_PROP_POS_MSEC, start_time)
    # Parameters for lucas kanade optical flow
    lk_params = dict(winSize=(30, 30),
                     maxLevel=2,
                     criteria=(cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 0.03))
    # Create some random colors
    color = np.random.randint(0, 255, (100, 3))
    # Take first frame and find corners in it
    ret, old_frame = cap.read()

    (x, y, w, h) = cv.selectROI('Select Window', old_frame, fromCenter=False, showCrosshair=True)
    old_gray = cv.cvtColor(old_frame, cv.COLOR_BGR2GRAY)

    point_list = []
    for _y in range(y + int(0.4 * h), y + int(0.6 * h), 15):
        for _x in range(x + int(0.4 * w), x + int(0.6 * w), 15):
            point_list.append((_x, _y))
    points = np.array(point_list)
    points = np.float32(points[:, np.newaxis, :])
    # Create a mask image for drawing purposes
    mask = np.zeros_like(old_frame)
    vector = np.array([0, 0])
    cv.destroyWindow('Select Window')
    isPlay = True
    while cap.isOpened():
        ret, frame = cap.read()
        frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
        # calculate optical flow
        p1, st, err = cv.calcOpticalFlowPyrLK(old_gray, frame_gray, points, None, **lk_params)
        # Select good points
        good_new = p1[st == 1]
         good_old = points[st == 1]
         # draw the tracks
         for i, (new, old) in enumerate(zip(good_new, good_old)):
             a, b = new.ravel()
             a = int(a)
             b = int(b)
             c, d = old.ravel()
             c = int(c)
             d = int(d)
             vector = vector + np.array([a - c, b - d])
             vector[0] = int(vector[0] / 2)
             vector[1] = int(vector[1] / 2)
             mask = cv.line(mask, (vector[0] + c, vector[1] + d), (c, d), color[i].tolist(), 2)
             frame = cv.circle(frame, (vector[0] + c, vector[1] + d), 5, color[i].tolist(), -1)
         img = cv.add(frame, mask)
         image = cv.cvtColor(img, cv.COLOR_BGR2RGB)
         image = Image.fromarray(image)
         image_tk = ImageTk.PhotoImage(image)

         if panel is None:
             panel = Label(image=image_tk)
             panel.image = image_tk
             panel.pack()
         else:
             panel.configure(image=image_tk)
             panel.image = image_tk

         # Now update the previous frame and previous points
         old_gray = frame_gray.copy()
         points = good_new.reshape(-1, 1, 2)
         if cv.waitKey(30) == 27 or isPlay is False:
             return
```

4. Object Tracking Algorithm

Q & A

End of Presentation