Sensoriamento por imagens na análise da utilização de equipamentos urbanos:

Playgrounds em praças públicas

Universidade Federal de Santa Catarina - UFSC

Computação e Programação Aplicada ao Processo de Projeto

Professor: Carlos Eduardo Verzola Vaz

Aluno: Jaceguay Zukoski

- Objetivo
- Metodologia
- Estudos
- Conclusão

Objetivo:

Levantar dados sobre a utilização de equipamentos urbanos, especificamente playgrounds em praças públicas.

O tratamento dos dados armazenados baseia-se na correlação da densidade de usuários ao longo do tempo, utilizando-se de técnicas de processamento de imagem.

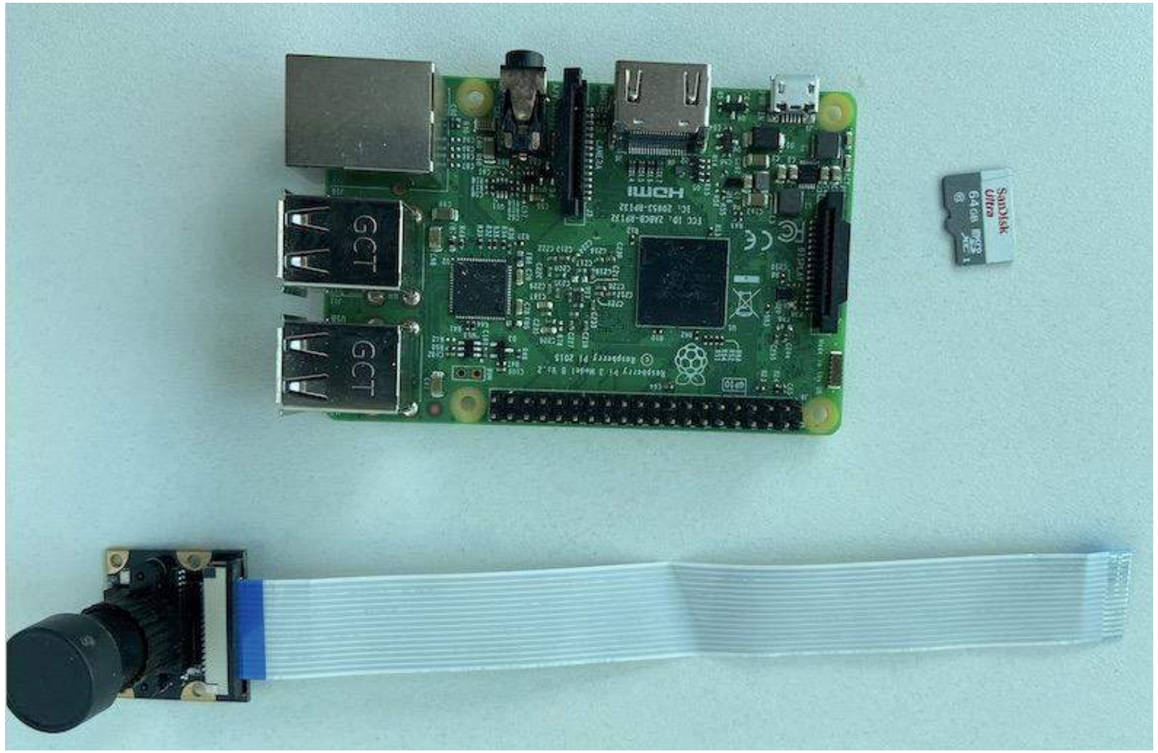
Tanto a captura quanto o processamento devem ocorrer in-loco, deste modo lidando com restrições sobre o armazenamento das imagens ou banda para transmissão de dados.

Metodologia:

Metodologia:

Equipamentos:

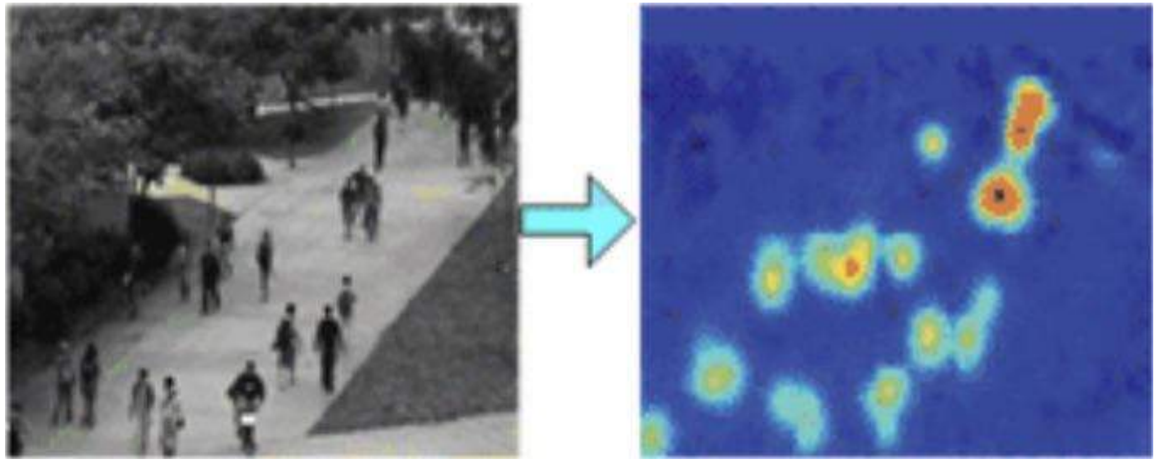
- Raspberry Pie Model 3B
- Módulo câmera Ov5647 5mp 75° 3.6mm 1080p
- Cartão SD 32Gb



As imagens coletadas são processadas diretamente pela unidade, o resultado é gravado em um mapa de pontos como um arquivo .csv contendo as coordenadas do ponto, seu rótulo e tempo em segundos da sua captura:

x	y	classe	segundo	nomeclasse
488	271	13	16	person
463	318	13	907	person
503	263	13	907	person
263	315	13	907	person

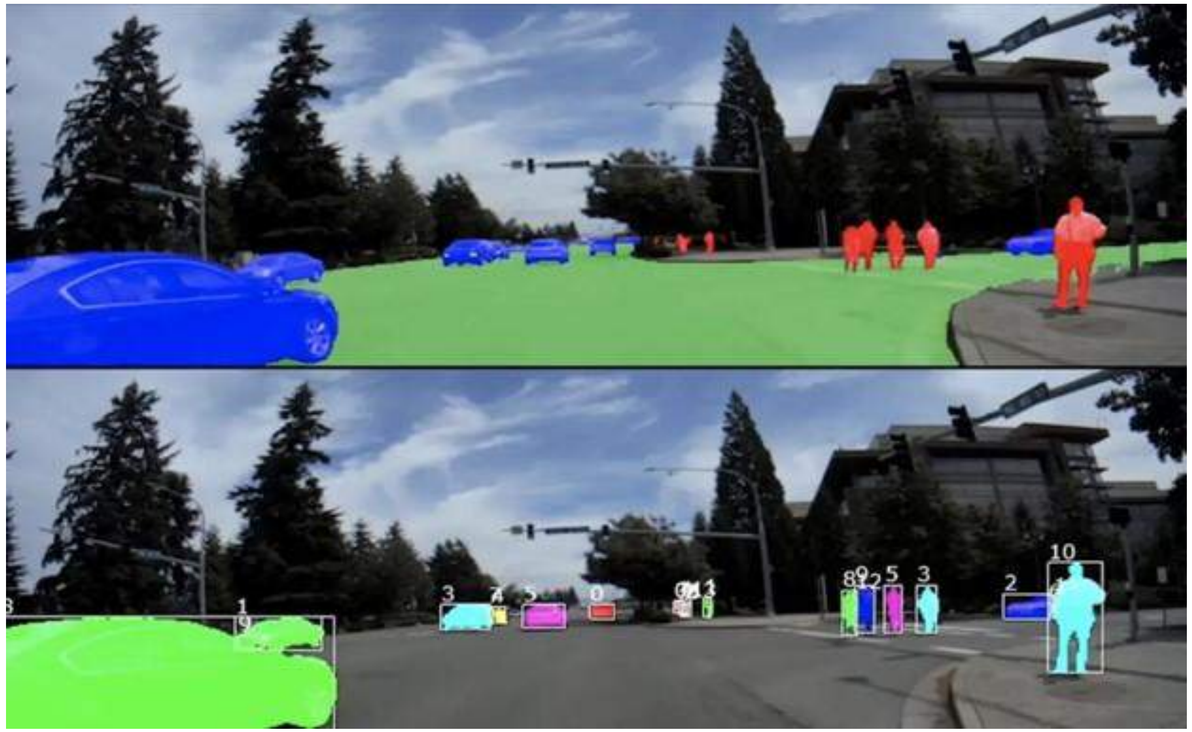
O processamento prevê a visualização da densidade do uso do espaço através da soma destes pontos.



Metodologia:

Etapas:

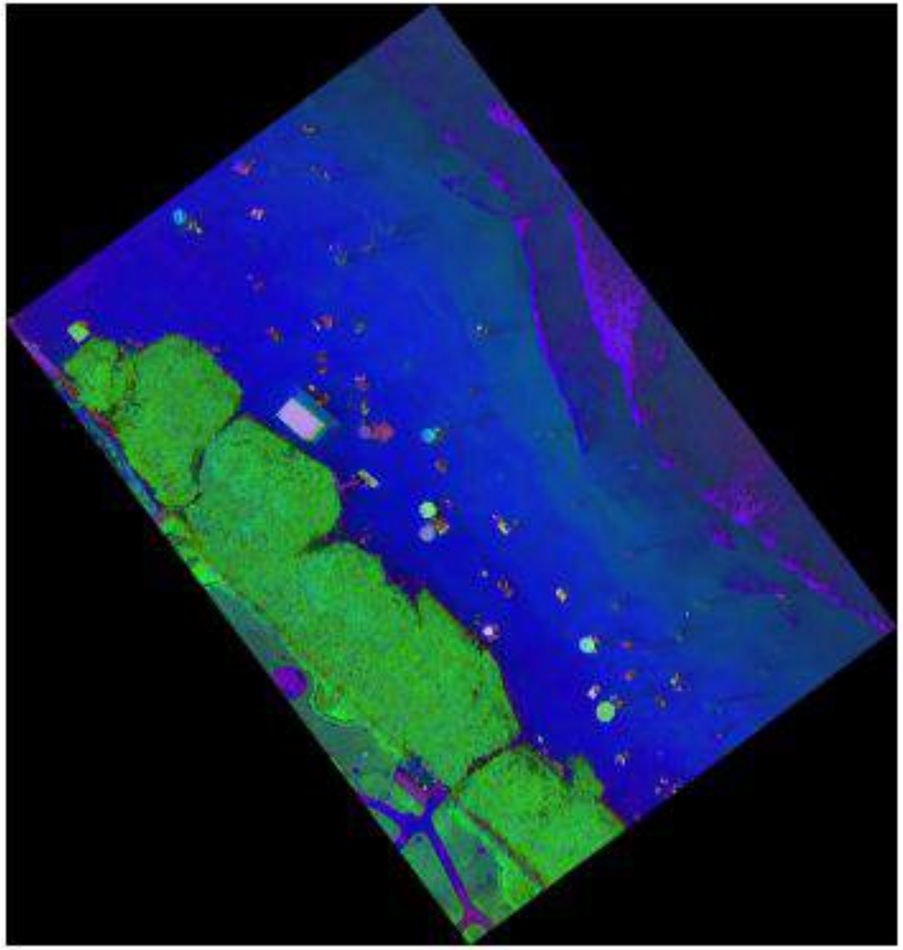
- **Pré-processamento:** remoção de ruído, ajuste de contraste e normalização;
- **Realce:** Filtros a fim de destacar as características básicas (formas, texturas e iluminação);
- **Segmentação:** Separação em partes distintas(objetos) e fundo da imagem;
- **Classificação:** Aplicação de um modelo treinado por aprendizado de máquina para atribuição do significado de cada objeto (rótulo).



segmentação de imagem e seleção de objetos. (fonte: https://blogs.nvidia.com)

Metodologia - exemplo tratamento de baixo nível :

Aplicação de operações primitivas à imagens (ruído, contraste, nitidez, aplicação de filtros…).
São tratados os valores dos pixels e aplicados métodos estatísticos para sua classificação:



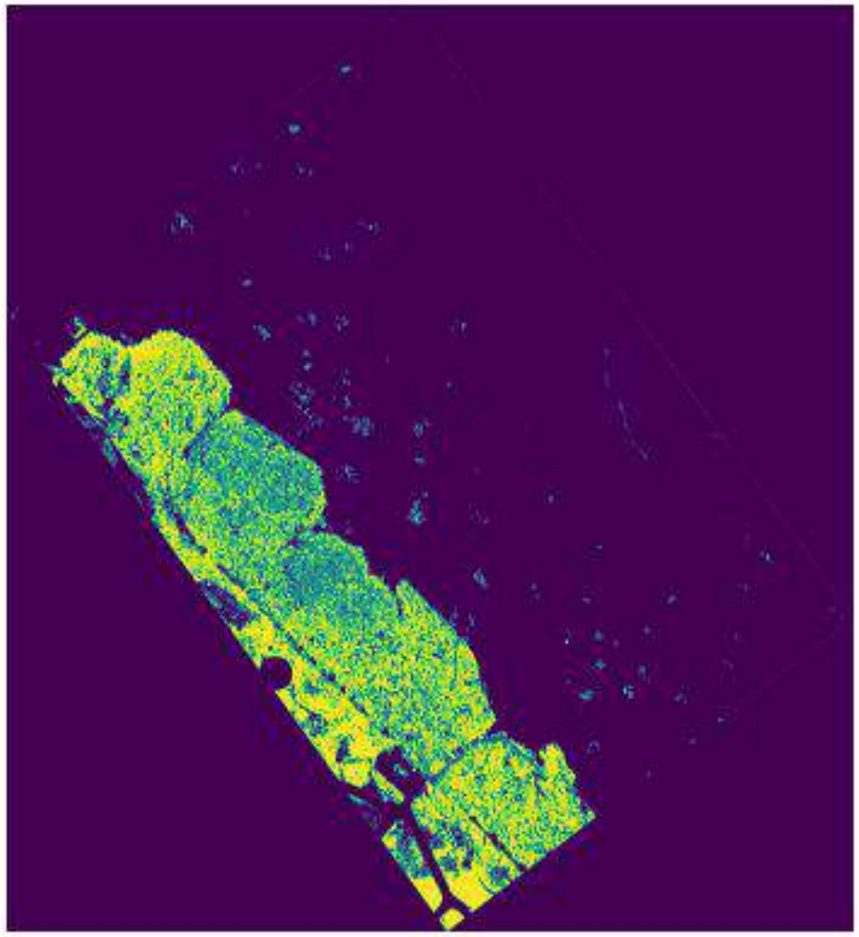
transformação do espaço de cores da combinação RGB(vermelho, verde e azul) em HSV(tom, saturação e valor)

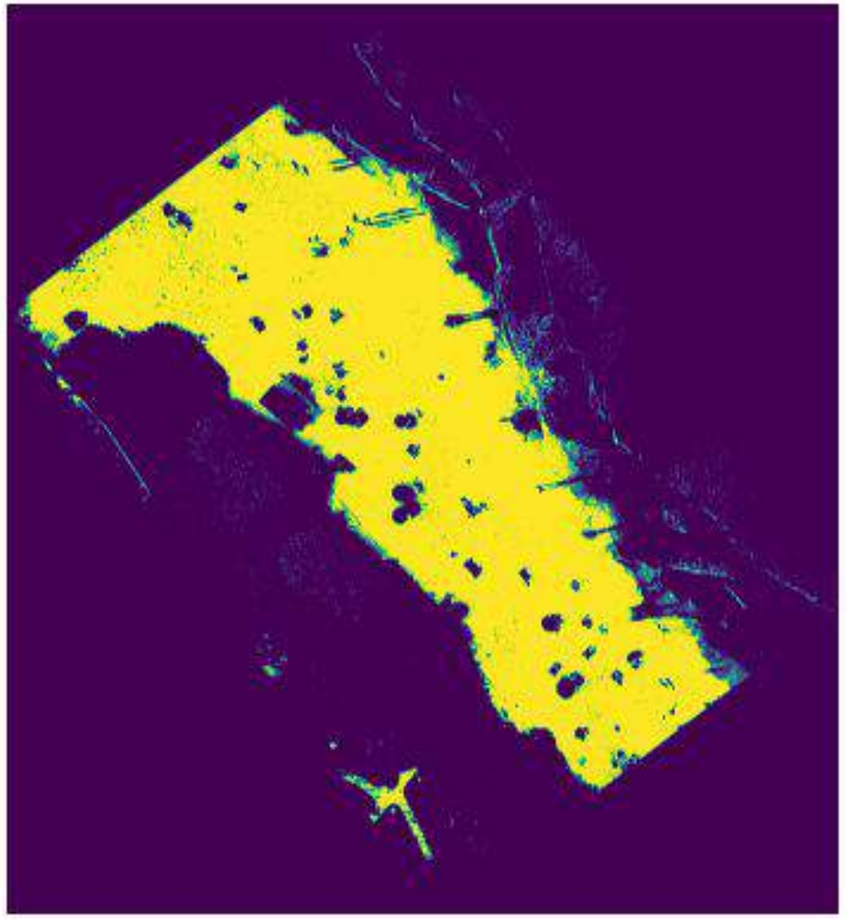
```
cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
```

Metodologia - exemplo tratamento de baixo nível :

Definição de um intervalo que representa melhor a área de vegetação, criação de uma máscara, o mesmo foi feito com a areia:

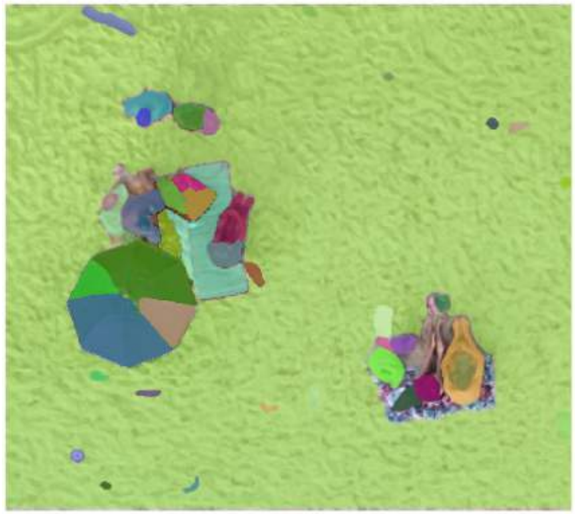
```
lower_veg = np.array([0, 40, 30])
upper_veg = np.array([170, 255, 90])
mask_veg = cv2.inRange(hsv, lower_veg, upper_veg)
```





Metodologia - exemplo tratamento de alto nível :

```
# ler e transformar imagem
image = cv2.imread('DJI_0128.JPG')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# aplicar modelo de segmentação
sam = sam_model_registry["vit_h"](
   checkpoint="sam_vit_h_4b8939.pth")
mask_generator = SamAutomaticMaskGenerator(sam)

masks = mask_generator.generate(image)

# selecionar máscaras com base na área
masks_filtradas = [ann for ann in masks if (ann['area'] > 500) & (ann['area'] < 10000)]

# classificar os objetos encontrados
model = YOLO("yolov8m.pt")

results = model(frame, device="mps")
     result = results[0]
     bboxes = np.array(result.boxes.xyxy.cpu(), dtype="int")
     classes = np.array(result.boxes.cls.cpu(), dtype="int")
```



Metodologia:

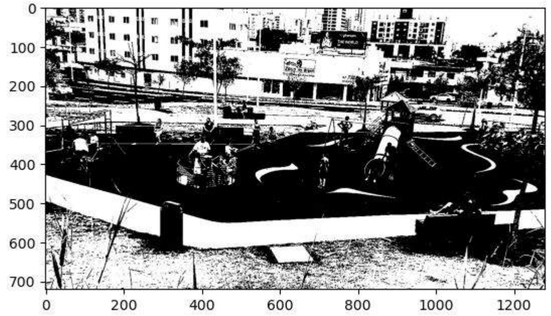
Protótipo:

Estudos:

Estudo 1:

Diferença entre pixels, duas capturas a cada segundo

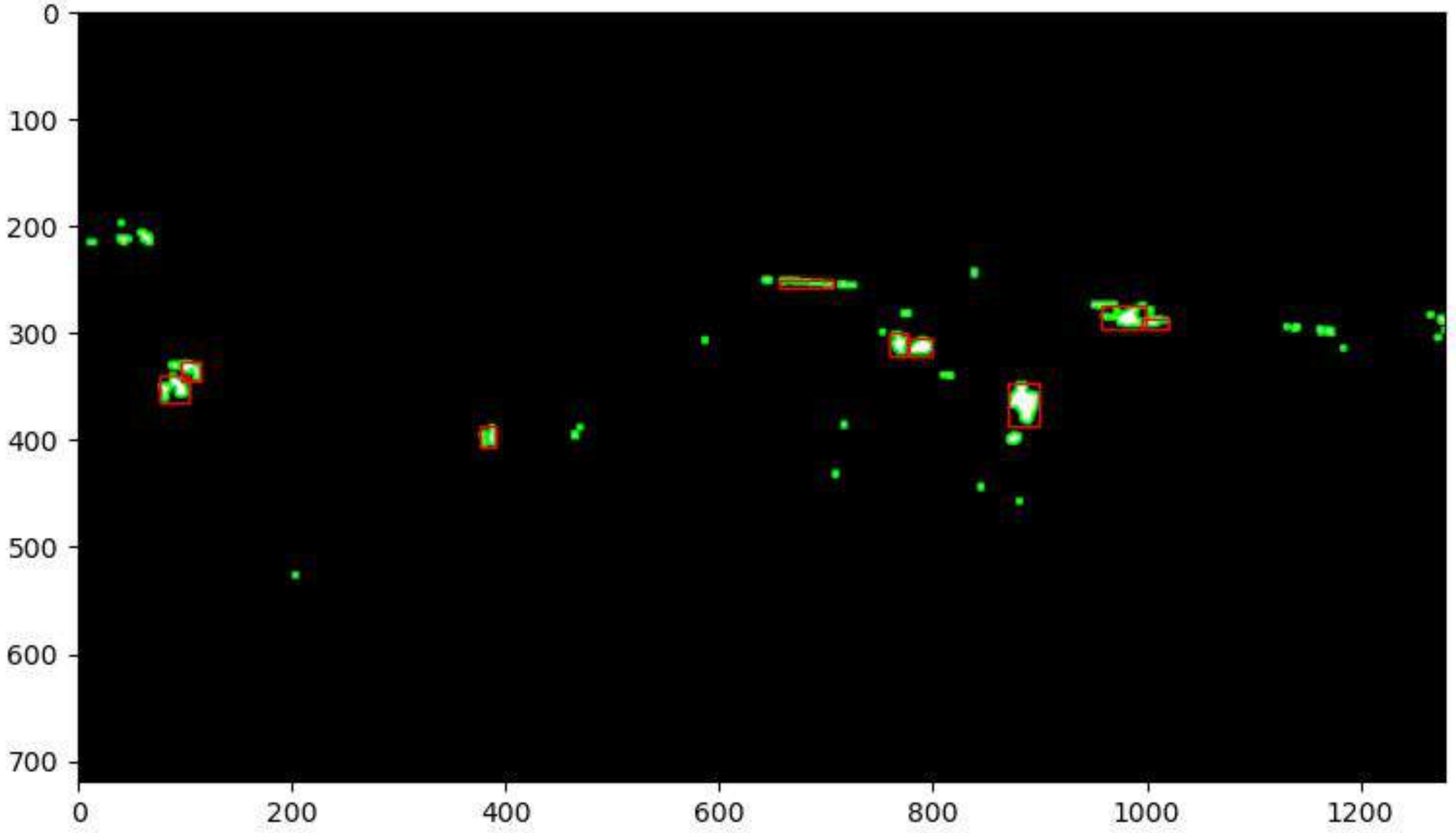
```python
def comparar_imgs(img1, img2):
    # imagens para binário
    img1 = cv2.imread(img1)
    img2 = cv2.imread(img2)
    gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
    (thresh1, binary1) = cv2.threshold(gray1, 127, 255, cv2.THRESH_BINARY)
    (thresh2, binary2) = cv2.threshold(gray2, 127, 255, cv2.THRESH_BINARY)
    # diferença entre as imagens
    diff = cv2.absdiff(binary1, binary2)
    # segmentar o resultado em partes com áreas agrupadas
    kernel = np.ones((5, 5), np.uint8)
    diff = cv2.morphologyEx(diff, cv2.MORPH_OPEN, kernel)
    # encontar os contornos
    contours, hierarchy = cv2.findContours(diff, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    # uma caixa delimitadora para cada área
    boxes = []
    for c in contours:
        (x, y, w, h) = cv2.boundingRect(c)
        boxes.append((x, y, x+w, y+h))
    # resultado da diferença,em vermelho as caixas delimitadoras com área maior que 200 pixels
    fig, ax = plt.subplots(1, 2, figsize=(20, 8))
    ax[0].imshow(diff, cmap='gray')
    ax[1].imshow(cv2.drawContours(cv2.cvtColor(diff, cv2.COLOR_GRAY2RGB), contours, -1, (0, 255, 0), 2))
    for box in boxes:
        if (box[2]-box[0])*(box[3]-box[1]) > 200:
            ax[1].add_patch(plt.Rectangle((box[0], box[1]), box[2]-box[0], box[3]-box[1], fill=False, color='red'))
    plt.show()
```



Estudo 1:

Diferença entre pixels, duas capturas a cada segundo

resultado da diferença entre as duas imagens convertidas em imagens monocromáticas, as caixas delimitadoras com área maior que 200 pixels demarcadas em vermelho:



Estudo 1:

Diferença entre pixels, duas capturas a cada segundo

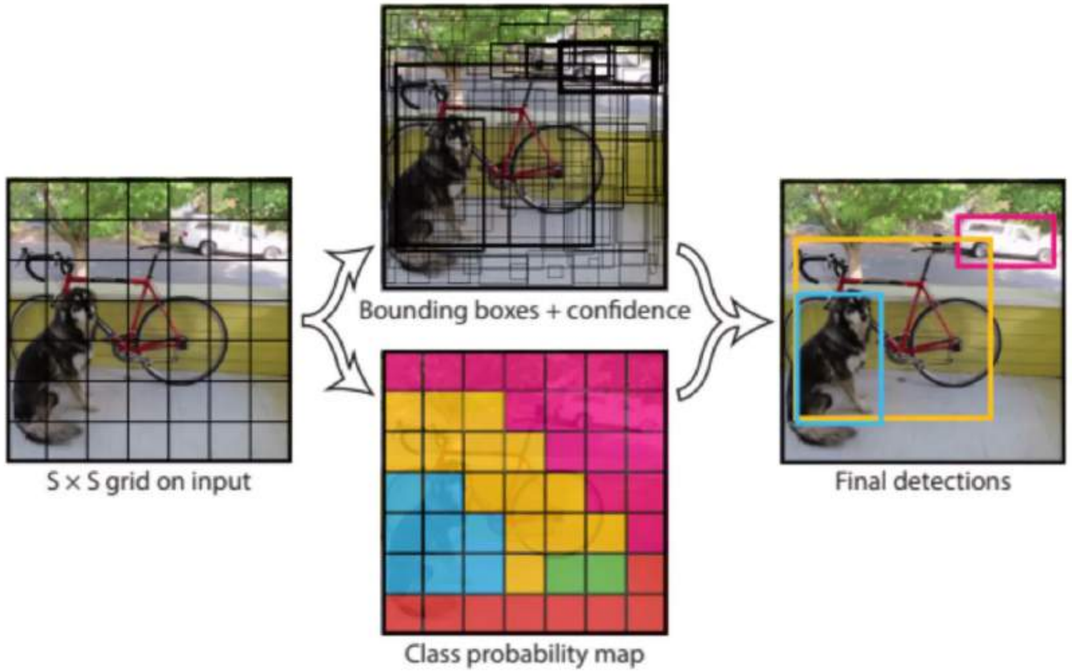
Estudo 2:

Modelo de classificação de objetos

You only look once (YOLO)

é um método de detecção de objetos em tempo real que é baseado em uma única rede neural convolucional (CNN). Esta é responsável em dividir e classificar a imagem.

A imagem é dividida em regiões iguais que são avaliadas por sua probabilidade em serem parte de uma classe, incluindo a probabilidade da posição da caixa delimitadora da classe com base na célula.



fonte: https://pjreddie.com/darknet/yolo/

```python
if quadro % 60 == 0:
    results = model(frame, device="mps")
    result = results[0]
    bboxes = np.array(result.boxes.xyxy.cpu(), dtype="int")
    classes = np.array(result.boxes.cls.cpu(), dtype="int")


# columnas: coordenada x, y do centróide, classe, quadro e segundo
    for cls, bbox in zip(classes, bboxes):
        (x, y, x2, y2) = bbox
        x_centroide = (x + x2) / 2
        y_centroide = (y + y2) / 2
        df = pd.DataFrame(
            {
                "x_centroide": [str(int(x_centroide))],
                "y_centroide": [str(int(y_centroide))],
                "classe": [str(int(cls))],
                "quadro": [str(int(quadro))],
                "segundo": [str(int(quadro / 60))],
            }
        )
        df.to_csv(f"centroids/{str(int(quadro))}.csv",
                  header=False, mode="a", index=False)
```

Estudo 2:

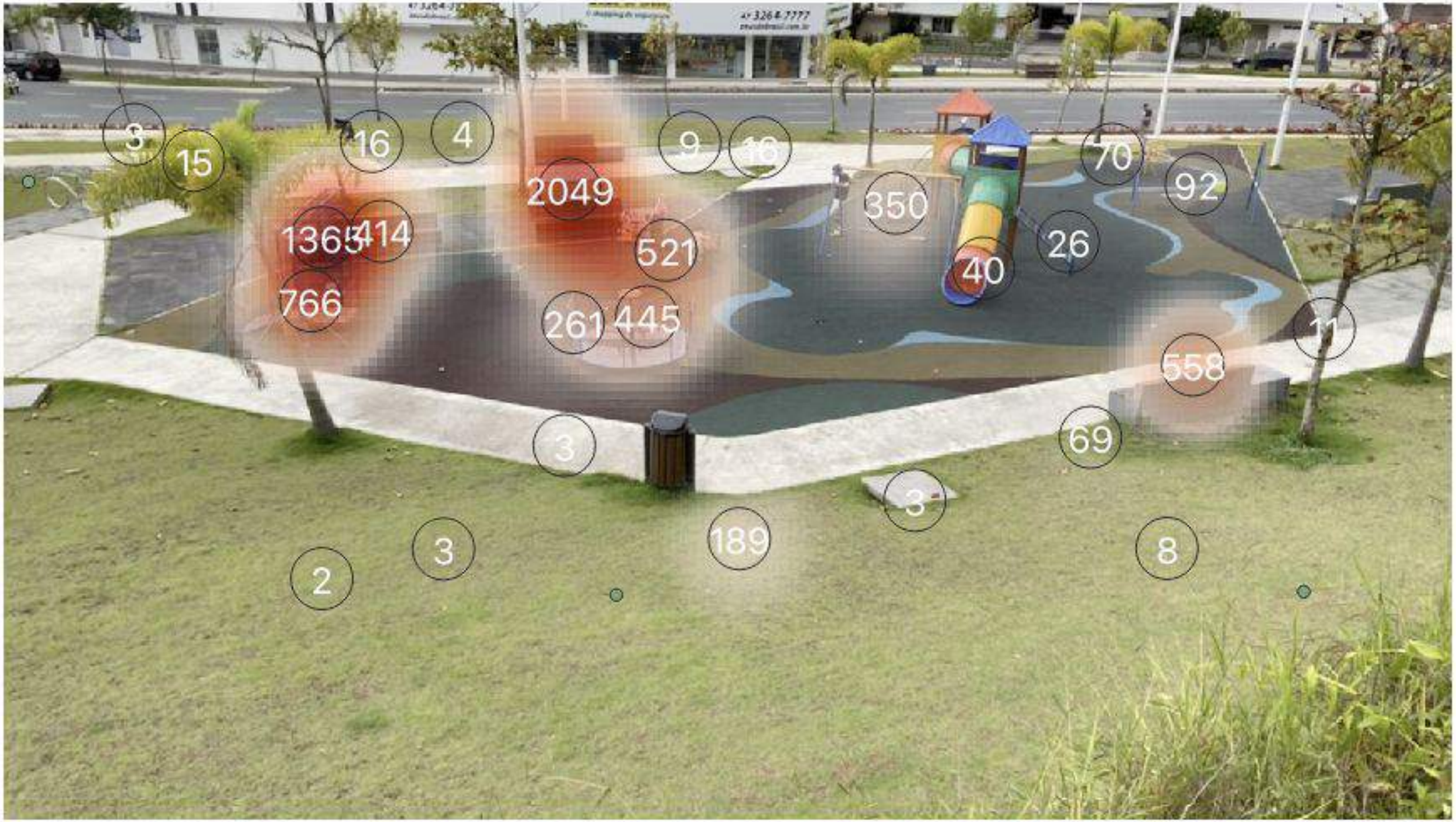
Modelo de classificação de objetos

Estudo 2:

Modelo de classificação de objetos

Estudo 2:

Soma dos pontos detectados ao longo de 45 minutos classe pessoas



Conclusão:

O modelo de baixo nível, possui como vantagens o baixo requisito de processamento e principalmente a capacidade de funcionar em imagens de baixa resolução. Já o modelo de alto nível pode capturar o uso do espaço de modo mais abrangente.

A partir dos resultados obtidos, é possível concluir que a metodologia utilizada com o modelo de detecção de objetos é eficaz para coletar e tratar dados sobre o uso dos playgrounds em praças públicas.

É contudo necessário acrescentar uma etapa de contextualização para cada cenário, de acordo com o local de monitoramento. Para que efetivamente possam ser comparadas às áreas de lazer infantil, estares e circulação.