BÁO CÁO CÔNG VIỆC

**Công việc số:** 59

**Mô tả công việc:** Tìm hiểu phương án truyền dữ liệu qua socket giữa C# và Python

**Người thực hiện:** Hạ Quang Dũng

**Ngày bắt đầu:** 20/11/2024

**Ngày kết thúc:** 21/11/2024

**NỘI DUNG TÀI LIỆU**

[1. Mục tiêu tổng quan: 2](#_u1v2mb27fajo)

[**2. Kiến trúc tổng quan: 2**](#_h57e9jqbteae)

[**3. Chi tiết mã nguồn 2**](#_xs91s2yvaklu)

[3.1. Phía bên Client - Mô tả hoạt động 2](#_ffkat4a3tjmx)

[3.1.1. Khởi tạo và Cấu hình 3](#_960dmltalckn)

[3.1.2. Chức năng chính 3](#_8bfdfjj70e24)

[3.1.2.1. Chọn video 3](#_oc7rnfe0dd9f)

[3.1.2.2. Trích xuất hình ảnh từ video 3](#_v0pe92ptfkm)

[3.1.2.3. Hiển thị bounding box trên hình ảnh 4](#_k1bync94zvd8)

[3.1.2.4. Cập nhật kết quả giao diện 4](#_6bct5hs2pg7v)

[3.1.3. Quy trình hoạt động 4](#_cjz15ita38d4)

[3.2. Phía bên Server - Mô tả hoạt động 9](#_2ubn2uohc43z)

[3.2.1. Khởi tạo và tải mô hình YOLO 9](#_d2bucjwunadj)

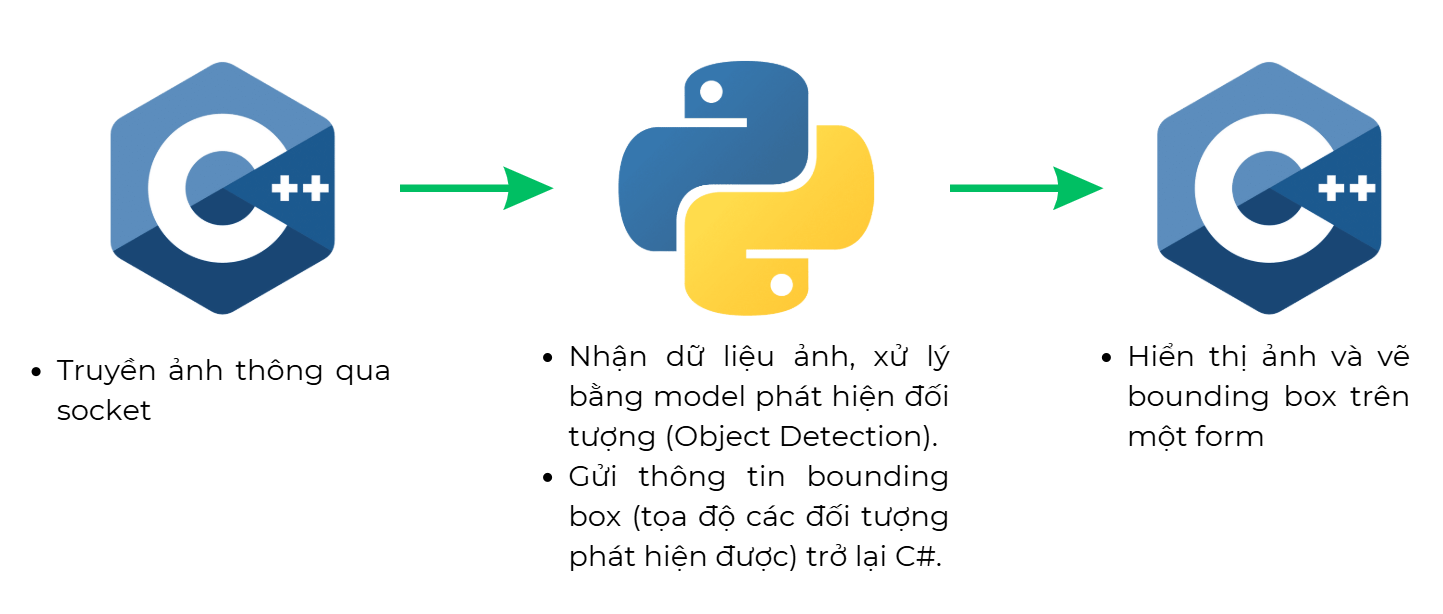
[3.2.2. Phát hiện đối tượng trong ảnh 9](#_3rief13j0hj8)

[3.2.3. Server Socket 9](#_hdo4nx2p03gt)

[3.2.4. Giao tiếp giữa client và server 10](#_esafq1hfmimh)

**Tìm hiểu phương án truyền dữ liệu qua socket giữa C# và Python**

# 1. Mục tiêu tổng quan:

****

# 2. Kiến trúc tổng quan:

1. **Socket Communication**:
   * **C# (Client)**: Gửi ảnh.
   * **Python (Server)**: Nhận ảnh, xử lý, và trả kết quả.
2. **Processing Flow**:
   * Ảnh → (C# gửi) → Python → (Object Detection) → Kết quả → (Python gửi lại) → C#.
3. **Data Formats**:
   * Ảnh: Byte stream (PNG/JPG).
   * Bounding box: JSON hoặc định dạng nhị phân.

# 3. Chi tiết mã nguồn

## 3.1. Phía bên Client - Mô tả hoạt động

Đoạn mã này là một phần của ứng dụng **Vehicle Detection** được phát triển bằng **C#**, hoạt động như một client để thực hiện trích xuất hình ảnh từ video và gửi dữ liệu đến một server Python xử lý phát hiện đối tượng. Sau đây là mô tả chi tiết các thành phần và hoạt động chính của client:

### 3.1.1. Khởi tạo và Cấu hình

**Thuộc tính chính**

* **\_imageExtractor**: Đối tượng để trích xuất hình ảnh từ video.
* **\_rootDir**: Thư mục gốc của dự án.
* **\_extractImageFolder**: Đường dẫn lưu trữ ảnh trích xuất từ video.
* **\_videoPath**: Đường dẫn file video người dùng chọn.
* **\_frameSkipQuantity**: Số khung hình được bỏ qua giữa các lần trích xuất (tùy chỉnh được thông qua giao diện).
* **frameTimeExecute**: Dictionary lưu kết quả phát hiện phương tiện từ server.
* **host & port**: Địa chỉ IP và cổng kết nối đến server Python.
* **stopwatch**: Bộ đếm thời gian thực hiện trích xuất và phát hiện phương tiện.

**Khởi tạo Form**

* **MainForm**: Phương thức khởi tạo Form và thiết lập các thư mục liên quan.

### 3.1.2. Chức năng chính

#### 3.1.2.1. Chọn video

* **btnSelectFile\_Click**:
  + Cho phép người dùng chọn file video thông qua hộp thoại (OpenFileDialog).
  + Cập nhật đường dẫn video và phát video trên giao diện sử dụng Windows Media Player (wmpVideo).

#### 3.1.2.2. Trích xuất hình ảnh từ video

* **btnExtractImages\_Click**:
  + Kiểm tra người dùng đã chọn video hay chưa.
  + Sử dụng đối tượng ImageExtractor để trích xuất các khung hình từ video vào thư mục đã định nghĩa (\_extractImageFolder).
  + Xử lý các khung hình và gửi chúng đến server Python để phát hiện phương tiện.
* **ProcessImage**:
  + Trích xuất ảnh từ video.
  + Lọc các file ảnh dựa trên định dạng (jpg, png, bmp, v.v.).
  + Kết nối tới server Python qua giao thức TCP:
    - Gửi kích thước và dữ liệu ảnh.
    - Nhận phản hồi từ server (bounding boxes và labels).
  + Hiển thị ảnh với bounding box và thông tin trên giao diện.
  + Cập nhật kết quả phát hiện vào bảng hiển thị (DataGridView).

#### 3.1.2.3. Hiển thị bounding box trên hình ảnh

* **DisplayImageWithBoundingBoxes**:
  + Đọc file ảnh và vẽ các bounding box lên hình.
  + Mỗi bounding box được hiển thị cùng nhãn loại phương tiện phát hiện (label).
  + Hiển thị ảnh đã xử lý trên PictureBox.

#### 3.1.2.4. Cập nhật kết quả giao diện

* **DisplayDetectionResult**:
  + Hiển thị tổng thời gian xử lý, tổng số phương tiện và kết quả chi tiết về từng loại phương tiện trong bảng (DataGridView).
* **UpdateDataGridView**:
  + Cập nhật bảng kết quả khi có thay đổi dữ liệu từ quá trình phát hiện.

### 3.1.3. Quy trình hoạt động

1. **Người dùng chọn file video**:
   * Hệ thống sẽ hiển thị video trong giao diện.
2. **Trích xuất khung hình từ video**:
   * Ứng dụng phân tích video và trích xuất các khung hình theo giá trị frameSkipQuantity.
   * Các khung hình được gửi lần lượt đến server Python qua socket.
3. **Kết nối server Python và nhận kết quả**:
   * Server Python xử lý ảnh, phát hiện phương tiện và trả về thông tin bounding box (tọa độ, loại phương tiện).
   * Kết quả được hiển thị trực quan trên giao diện người dùng.
4. **Hiển thị kết quả**:
   * Tổng số phương tiện và thời gian xử lý được hiển thị.
   * Các bounding box được vẽ lên hình ảnh để minh họa.

| using Emgu.CV;  using VehicleDetection.src.CSharp.Models;  using VehicleDetection.src.CSharp.Services;  using System.Net.Sockets;  using System.Text;  using Newtonsoft.Json;  using System.Diagnostics;  namespace VehicleDetection\_8.\_0\_  {  public partial class MainForm : Form  {  #region Thuộc tính và Khởi tạo  private readonly ImageExtractor \_imageExtractor;  private readonly string \_rootDir;  private readonly string \_extractImageFolder;  private string \_videoPath;  private int \_frameSkipQuantity = 1;  private Dictionary<string, DetectionResult> frameTimeExecute = new Dictionary<string, DetectionResult>();  private string host = "127.0.0.1";  private int port = 5000;  private Stopwatch stopwatch = new Stopwatch();  public MainForm()  {  InitializeComponent();  \_rootDir = Path.GetFullPath(Path.Combine("..", "..", ".."));  \_extractImageFolder = Path.Combine(\_rootDir, "resources", "Image", "ExtractFromVideo");  \_imageExtractor = new ImageExtractor(\_extractImageFolder);  }  #endregion  private async void Form1\_Load(object sender, EventArgs e)  {  string scriptPath = Path.Combine(\_rootDir, "src", "Python", "SocketServer.py");  string modelPath = Path.Combine(\_rootDir, "model", "yolov8n.pt");  string outputPath = Path.Combine(\_rootDir, "resources", "Image", "OutputDetection", "vehicle\_image\_detected.jpg");  var pythonExecutor = new PythonExecutor("python", scriptPath);  string arr = "";  var result = pythonExecutor.Execute(arr);  }  #region Chọn File Video  private void btnSelectFile\_Click(object sender, EventArgs e)  {  using (var openFileDialog = new OpenFileDialog  {  Title = "Chọn File Video",  Filter = "Video Files|\*.mp4;\*.avi;\*.mov;\*.mkv|All Files|\*.\*"  })  {  if (openFileDialog.ShowDialog() == DialogResult.OK)  {  \_videoPath = openFileDialog.FileName;  wmpVideo.URL = \_videoPath;  wmpVideo.Ctlcontrols.play();  }  }  }  #endregion  #region Trích xuất Hình ảnh  private void btnExtractImages\_Click(object sender, EventArgs e)  {  if (\_videoPath == null)  {  MessageBox.Show("Vui lòng chọn Video trước khi trích xuất!");  return;  }  stopwatch = Stopwatch.StartNew();  Task.Factory.StartNew(() => ProcessImage());  }  private void ProcessImage()  {  ImageExtractor imageExtractor = new ImageExtractor(\_extractImageFolder);  imageExtractor.ExtractImages(\_videoPath, \_frameSkipQuantity);  // Các phần mở rộng của file ảnh mà bạn muốn lấy  string[] imageExtensions = { ".jpg", ".jpeg", ".png", ".bmp", ".gif", ".tiff" };  try  {  DetectionResult detectionResult = new DetectionResult();  // Lấy danh sách file từ thư mục  string[] allFiles = Directory.GetFiles(\_extractImageFolder);  // Lọc file ảnh dựa trên phần mở rộng  foreach (string file in allFiles)  {  stopwatch = Stopwatch.StartNew();  stopwatch.Start();  if (Array.Exists(imageExtensions, ext => ext.Equals(Path.GetExtension(file), StringComparison.OrdinalIgnoreCase)))  {  // Kết nối đến Python server  using (TcpClient client = new TcpClient(host, port))  using (NetworkStream stream = client.GetStream())  {  // Đọc ảnh và chuyển sang byte[]  byte[] imageBytes = File.ReadAllBytes(file);  byte[] sizeBytes = Encoding.UTF8.GetBytes(imageBytes.Length.ToString().PadLeft(16, '0'));  // Gửi kích thước ảnh  stream.Write(sizeBytes, 0, sizeBytes.Length);  // Gửi ảnh  stream.Write(imageBytes, 0, imageBytes.Length);  // Nhận kết quả bounding box  using (MemoryStream ms = new MemoryStream())  {  byte[] buffer = new byte[4096];  int bytesRead;  while ((bytesRead = stream.Read(buffer, 0, buffer.Length)) > 0)  {  ms.Write(buffer, 0, bytesRead);  }  string response = Encoding.UTF8.GetString(ms.ToArray());  var boundingBoxes = JsonConvert.DeserializeObject<dynamic>(response);  // Cập nhật kết quả vào dictionary  foreach (var box in boundingBoxes)  {  string label = box.label.ToString();  if (detectionResult.VehicleCounts.ContainsKey(label))  {  detectionResult.VehicleCounts[label]++;  }  else  {  detectionResult.VehicleCounts[label] = 1;  }  }  Invoke(new Action(() => DisplayImageWithBoundingBoxes(file, boundingBoxes)));  }  Invoke(new Action(() => pictureBox1.Refresh()));  }  }  stopwatch.Stop();  detectionResult.TotalTime = (double)stopwatch.Elapsed.TotalSeconds;  detectionResult.TotalVehicles = (int)detectionResult.VehicleCounts.Values.Sum();  UpdateDataGridView(detectionResult.VehicleCounts);  File.Delete(file);  Invoke(new Action(() => DisplayDetectionResult(detectionResult)));  detectionResult.VehicleCounts.Clear();  }  }  catch (Exception ex)  {  Console.WriteLine($"Có lỗi xảy ra: {ex.Message}");  }  }  private void DisplayImageWithBoundingBoxes(string filePath, dynamic boundingBoxes)  {  using (Image img = Image.FromFile(filePath))  {  Bitmap bitmap = new Bitmap(img);  using (Graphics g = Graphics.FromImage(bitmap))  {  Color customColor = ColorTranslator.FromHtml("#33FF66");  using (Pen pen = new Pen(customColor, 2)) // Độ dày là 2  {  foreach (var box in boundingBoxes)  {  int x = box.x, y = box.y, w = box.w, h = box.h;  g.DrawRectangle(pen, x, y, w, h);  g.DrawString(box.label.ToString(), new Font("Arial", 12), new SolidBrush(ColorTranslator.FromHtml("#33FF66")), x, y - 24);  }  }  }  pictureBox1.Image = bitmap;  pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;  }  }  private void nmrframeSkip\_ValueChanged(object sender, EventArgs e)  {  \_frameSkipQuantity = (int)nmrframeSkip.Value;  }  #endregion  #region Cập nhật Giao diện Người dùng (UI)  private void DisplayDetectionResult(DetectionResult detectionResult)  {  if (detectionResult.VehicleCounts == null) return;  lbTotalTime.Text = $"Tổng thời gian thực hiện: {detectionResult.TotalTime} giây";  lbTotalVehicles.Text = $"Tổng số phương tiện: {detectionResult.TotalVehicles}";  dataGridView1.Columns.Clear();  dataGridView1.Columns.Add("VehicleType", "Loại Phương Tiện");  dataGridView1.Columns.Add("Count", "Số Lượng");  dataGridView1.Rows.Clear();  foreach (var result in detectionResult.VehicleCounts)  {  dataGridView1.Rows.Add(result.Key, result.Value);  }  }  private async void UpdateDataGridView(Dictionary<string, int> results)  {  Invoke((Action)(() =>  {  dataGridView1.Columns.Clear();  dataGridView1.Columns.Add("VehicleType", "Loại Phương Tiện");  dataGridView1.Columns.Add("Count", "Số Lượng");  dataGridView1.Rows.Clear();  foreach (var result in results)  {  dataGridView1.Rows.Add(result.Key, result.Value);  }  }));  }  #endregion  }  } |
| --- |

## 3.2. Phía bên Server - Mô tả hoạt động

Phía server sử dụng Python để triển khai một hệ thống phát hiện đối tượng dựa trên mô hình **YOLO v8**. Server hoạt động như một socket server, nhận ảnh từ client, xử lý phát hiện đối tượng, và gửi kết quả lại dưới dạng JSON. Dưới đây là mô tả chi tiết từng thành phần:

### 3.2.1. Khởi tạo và tải mô hình YOLO

* **Mô hình YOLO v8** được tải một lần khi server khởi chạy. Đây là mô hình nhẹ, được sử dụng để phát hiện các đối tượng trong ảnh như ô tô, xe máy, hoặc các phương tiện khác.
* Server giữ mô hình trong bộ nhớ để giảm thời gian tải lại cho mỗi lần phát hiện.

### 3.2.2. Phát hiện đối tượng trong ảnh

* Ảnh đầu vào từ client được xử lý để phát hiện các bounding box của các đối tượng trong ảnh.
* Mỗi đối tượng phát hiện được chứa các thông tin:
  + **Vị trí**: Tọa độ góc trên trái và kích thước của bounding box.
  + **Nhãn (Label)**: Loại đối tượng, ví dụ: "car", "truck".
  + **Độ tin cậy (Confidence)**: Giá trị xác suất mô hình tự tin về dự đoán, thường từ 0 đến 1.
* Các bounding box được chuyển thành một danh sách dưới dạng JSON.

### 3.2.3. Server Socket

**Quy trình hoạt động**

1. **Khởi tạo server**:
   * Server mở một cổng (mặc định là 5000) và bắt đầu lắng nghe kết nối từ client.
   * Mỗi kết nối từ client được server chấp nhận và xử lý độc lập.
2. **Nhận dữ liệu từ client**:
   * Client gửi kích thước của ảnh trước, sau đó gửi dữ liệu ảnh dưới dạng nhị phân.
   * Server giải mã dữ liệu nhận được thành một ảnh màu để xử lý.
3. **Xử lý phát hiện đối tượng**:
   * Ảnh được đưa vào mô hình YOLO để phát hiện các bounding box.
   * Kết quả phát hiện được đóng gói thành JSON để dễ dàng truyền lại cho client.
4. **Trả kết quả về client**:
   * Kết quả bao gồm danh sách bounding box và thông tin nhãn được gửi trả qua socket.
5. **Xử lý lỗi**:
   * Server có khả năng xử lý lỗi như ảnh không hợp lệ, mất kết nối, hoặc các lỗi trong quá trình phát hiện.

### 3.2.4. Giao tiếp giữa client và server

* **Dữ liệu từ client**:
  + Client gửi ảnh dưới dạng nhị phân qua socket, đảm bảo kích thước và định dạng ảnh đúng chuẩn.
* **Kết quả trả về từ server**:
  + Server trả về danh sách bounding box với thông tin chi tiết về từng đối tượng phát hiện trong ảnh.

| from ultralytics import YOLO  import socket  import cv2  import numpy as np  import json  # Tải mô hình YOLO (tải một lần để tiết kiệm tài nguyên)  model = YOLO('./Model/yolov8n.pt')  # Hàm phát hiện đối tượng  def detect\_objects(image):  """  Phát hiện đối tượng trong ảnh sử dụng YOLO.  Args:  image (numpy.ndarray): Ảnh đầu vào.  Returns:  List[Dict]: Danh sách đối tượng được phát hiện, mỗi đối tượng có định dạng:  {"x": int, "y": int, "w": int, "h": int, "label": str, "confidence": float}.  """  # Thực hiện dự đoán  results = model(image)  # Lưu các kết quả phát hiện  detections = []  for result in results:  for box in result.boxes:  x1, y1, x2, y2 = box.xyxy[0].cpu().numpy() # Góc trên trái và dưới phải của bounding box  confidence = float(box.conf[0].cpu().numpy()) # Độ chính xác của dự đoán  class\_id = int(box.cls[0].cpu().numpy()) # ID của lớp  label = model.names[class\_id] # Tên lớp đối tượng  # Tính toán width và height từ bounding box  w = int(x2 - x1)  h = int(y2 - y1)  # Lưu thông tin bounding box vào danh sách kết quả  detections.append({  "x": int(x1),  "y": int(y1),  "w": w,  "h": h,  "label": label,  "confidence": round(confidence, 2)  })  return detections  # Khởi tạo socket server  def start\_server(host='localhost', port=5000):  """  Khởi động server để nhận ảnh qua socket và trả về bounding box.  Args:  host (str): Địa chỉ host. Mặc định là 'localhost'.  port (int): Cổng của server. Mặc định là 5000.  """  server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  server\_socket.bind((host, port))  server\_socket.listen(1)  print(f"Server đang chạy trên {host}:{port}...")  while True:  conn, addr = server\_socket.accept()  print(f"Kết nối từ: {addr}")  try:  # Nhận kích thước ảnh  img\_size = int(conn.recv(16).decode())  print(f"Kích thước ảnh nhận được: {img\_size}")  # Nhận dữ liệu ảnh  img\_data = b''  while len(img\_data) < img\_size:  packet = conn.recv(4096)  if not packet:  break  img\_data += packet  # Decode ảnh  np\_data = np.frombuffer(img\_data, np.uint8)  image = cv2.imdecode(np\_data, cv2.IMREAD\_COLOR)  if image is None:  raise ValueError("Không thể decode ảnh nhận được.")  print("Ảnh đã nhận thành công.")  # Phát hiện đối tượng  bounding\_boxes = detect\_objects(image)  # Gửi bounding box lại dưới dạng JSON  result = json.dumps(bounding\_boxes).encode()  conn.sendall(result)  print("Kết quả đã gửi.")  except Exception as e:  print(f"Lỗi: {e}")  finally:  conn.close()  if \_\_name\_\_ == "\_\_main\_\_":  try:  start\_server()  except KeyboardInterrupt:  print("Server đã dừng.") |
| --- |