**BÁO CÁO CÔNG VIỆC**

**Công việc số:** 68

**Mô tả công việc:** Chuyển mã nguồn tính tốc độ của phương tiện giao thông qua video từ python qua Csharp

**Người thực hiện:** Hạ Quang Dũng

**Ngày bắt đầu:** 22/01/2025

**Ngày kết thúc:** 06/02/2025

***Ghi chú:*** *Tạm dừng từ ngày 25/01/2025, tiếp tục ngày 03/02/2025 (09 ngày) nghỉ tết*

**NỘI DUNG TÀI LIỆU**

[**1. Giới thiệu 2**](#_ye3e2aq1n8fk)

[**2. Công thức tính vận tốc 2**](#_75o3eamjnwxo)

[**3. Quá trình tính vận tốc trong mã nguồn 2**](#_hy3ommxvk54p)

[3.1. Hiệu chuẩn tỉ lệ mét trên pixel 2](#_19x0dtdpt1oz)

[3.2. Xác định vị trí của phương tiện 2](#_ltq5krc7i21x)

[3.3. Xác định phương tiện và theo dõi vị trí 3](#_3zrg63l60asg)

[3.4. Tính vận tốc 3](#_tyb8661670ng)

[**4. Hiển thị kết quả 4**](#_f1v78v4curka)

[**5. Tổng kết 4**](#_2um2aymbudia)

[**6. Mã nguồn chi tiết 5**](#_3cn95c3vpavo)

[6.1. Lớp VehicleTrackingInfo 5](#_igvq2zqt52yh)

[6.2. MainForm 5](#_bxvmfzn0j711)

# 

# 1. Giới thiệu

Trong mã nguồn C#, việc tính toán vận tốc của phương tiện dựa trên việc theo dõi vị trí của bounding box của phương tiện trong các khung hình liên tiếp. Vận tốc được tính bằng cách đo khoảng cách di chuyển của phương tiện trong một khoảng thời gian nhất định và quy đổi sang đơn vị km/h.

# 2. Công thức tính vận tốc

Quá trình tính toán vận tốc sử dụng công thức:

Trong đó:

* distancePixels: Khoảng cách di chuyển của phương tiện theo đơn vị pixel trong ảnh.
* metersPerPixel: Tỷ lệ chuyển đổi từ pixel sang mét (được hiệu chuẩn dựa trên chiều rộng làn đường).
* timeDiff: Khoảng thời gian giữa hai khung hình (tính theo giây).
* speedMps: Vận tốc theo đơn vị mét/giây (m/s).
* speedKmh: Vận tốc theo đơn vị km/h.

# 3. Quá trình tính vận tốc trong mã nguồn

## **3.1. Hiệu chuẩn tỉ lệ mét trên pixel**

Trước khi tính vận tốc, hệ thống cần hiệu chuẩn để xác định tỷ lệ metersPerPixel\text{metersPerPixel}:

| if (!\_calibrationDone) {  \_metersPerPixel = \_laneWidthMeters / 243;  \_calibrationDone = true; } |
| --- |

Ở đây, \_laneWidthMeters là chiều rộng làn đường (giả định 3.5m), còn 243 là giá trị cố định (có thể là số pixel đại diện cho chiều rộng làn đường trong ảnh).

## 3.2. Xác định vị trí của phương tiện

Tọa độ trung tâm của bounding box của phương tiện được tính như sau:

| float x\_center = box.x + box.w / 2; float y\_center = box.y + box.h / 2; var currentPos = new PointF(x\_center, y\_center); |
| --- |

## 3.3. Xác định phương tiện và theo dõi vị trí

Hệ thống kiểm tra khoảng cách giữa vị trí hiện tại và vị trí trước đó để xác định phương tiện có di chuyển hay không:

| foreach (var objId in \_vehicleTracking.Keys) {  var track = \_vehicleTracking[objId];  if (track.Positions.Any())  {  var lastPos = track.Positions.Last();  double dist = Math.Sqrt(Math.Pow(lastPos.X - currentPos.X, 2) + Math.Pow(lastPos.Y - currentPos.Y, 2));  if (dist < minDist && dist < 100)  {  minDist = dist;  matchedId = objId;  }  } } |
| --- |

Nếu không tìm thấy phương tiện khớp, hệ thống tạo ID mới:

| if (matchedId == -1) {  matchedId = \_nextId++;  \_vehicleTracking[matchedId] = new VehicleTrackingInfo { ... }; } |
| --- |

## 3.4. Tính vận tốc

Sau khi xác định phương tiện, hệ thống tính vận tốc dựa trên khoảng cách di chuyển giữa hai vị trí:

| var prevPos = vehicleTrack.Positions[vehicleTrack.Positions.Count - 2]; var prevTime = vehicleTrack.Timestamps[vehicleTrack.Timestamps.Count - 2]; var currentTime = vehicleTrack.Timestamps.Last();  // Sử dụng FPS để tính thời gian giữa hai khung hình double fps = 29.0; double timeDiff = (\_frameSkipQuantity + 1) / fps;  speed = CalculateSpeed(prevPos, currentPos, timeDiff); |
| --- |

Hàm CalculateSpeed thực hiện tính toán vận tốc:

| private double CalculateSpeed(PointF pos1, PointF pos2, double timeDiff) {  if (timeDiff < 0.001)  return 0.0;   double dx = pos2.X - pos1.X;  double dy = pos2.Y - pos1.Y;  double distancePixels = Math.Sqrt(dx \* dx + dy \* dy);  double distanceMeters = distancePixels \* \_metersPerPixel.Value;  double speedMps = distanceMeters / timeDiff;  double speedKmh = Math.Min(Math.Max(speedMps \* 3.6, 0), 200);    return speedKmh; } |
| --- |

Trong đó:

* dx, dy là sự thay đổi tọa độ của phương tiện.
* distancePixels là khoảng cách Euclidean giữa hai vị trí trong không gian pixel.
* distanceMeters chuyển đổi từ pixel sang mét.
* speedMps tính vận tốc theo m/s.
* speedKmh chuyển đổi sang km/h với giới hạn từ 0 đến 200 km/h để tránh giá trị không hợp lý.

# 4. Hiển thị kết quả

Vận tốc được vẽ lên ảnh:

| g.DrawString($"{CalculateSpeed(box)} km/h", new Font("Arial", 10), new SolidBrush(ColorTranslator.FromHtml("#ff2020")), x + 40, y - 24); |
| --- |

Điều này đảm bảo vận tốc được hiển thị rõ ràng cùng với bounding box.

# 5. Tổng kết

* Hệ thống theo dõi phương tiện bằng cách ghi nhận vị trí bounding box.
* Sử dụng khoảng cách Euclidean để tính toán sự di chuyển.
* Sử dụng thông tin FPS để xác định khoảng thời gian giữa hai khung hình.
* Chuyển đổi đơn vị từ pixel sang mét và từ m/s sang km/h.
* Giới hạn vận tốc trong khoảng hợp lý.
* Hiển thị kết quả trên giao diện đồ họa.

Với cách tiếp cận này, hệ thống có thể tính toán vận tốc phương tiện một cách tương đối chính xác dựa trên video đầu vào.

# 6. Mã nguồn chi tiết

## 6.1. Lớp VehicleTrackingInfo

| using System; using System.Collections.Generic; using System.Linq; using System.Security.Cryptography; using System.Text; using System.Threading.Tasks;  namespace VehicleDetection\_8.\_0\_.src.CSharp.Models {  public class VehicleTrackingInfo  {  public string Type { get; set; }  public List<PointF> Positions { get; set; }  public List<double> Timestamps { get; set; }  public List<double> Velocities { get; set; }  } } |
| --- |

## 6.2. MainForm

| using VehicleDetection.src.CSharp.Models; using VehicleDetection.src.CSharp.Services; using System.Text; using Newtonsoft.Json; using System.Diagnostics; using Grpc.Net.Client; using VehicleDetection\_8.\_0\_.src.CSharp.Models; using static System.Runtime.InteropServices.JavaScript.JSType;  namespace VehicleDetection\_8.\_0\_ {  public partial class MainForm : Form  {  #region Thuộc tính  private PythonExecutor \_pythonExecutor;  private ImageExtractor \_imageExtractor;  private int \_frameSkipQuantity;  private Dictionary<string, DetectionResult> \_frameTimeExecute = new();  private Stopwatch \_stopwatch = new();  private Dictionary<string, (double X, double Y)> \_previousPositions = new();    // Đường dẫn thư mục và file  private string \_rootDir;  private string \_extractImageFolder;  private string \_logFilePath;  private string \_modelPath;  private string \_videoPath;   // Thuộc tính liên quan tính vận tốc  private double \_laneWidthMeters = 3.5;  private Dictionary<int, VehicleTrackingInfo> \_vehicleTracking;  private bool \_calibrationDone;  private double? \_metersPerPixel;  private int \_nextId;   #endregion   #region Nhóm hàm khởi tạo  public MainForm()  {  InitializeComponent();  InitializePaths();  \_vehicleTracking = new Dictionary<int, VehicleTrackingInfo>();  }  private void InitializePaths()  {  \_rootDir = Path.GetFullPath(Path.Combine("..", "..", ".."));  \_extractImageFolder = Path.Combine(\_rootDir, "resources", "Image", "ExtractFromVideo");  \_logFilePath = Path.Combine(\_rootDir, "resources", "Logs", "Log.txt");  \_modelPath = Path.Combine(\_rootDir, "model", "yolov8n.pt");  \_imageExtractor = new ImageExtractor(\_extractImageFolder);  \_frameSkipQuantity = (int)nmrframeSkip.Value;  }  #endregion   #region Nhóm hàm sự kiện  private async void Form1\_Load(object sender, EventArgs e)  {  StartPythonGRPCServer();  }  private void MainForm\_FormClosing(object sender, FormClosingEventArgs e)  {  //PortKiller.KillAllProcessesAndConnectionsOnPort(50051);  }  private void btnSelectFile\_Click(object sender, EventArgs e)  {  SelectVideoFile();  }  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 nmrframeSkip\_ValueChanged(object sender, EventArgs e)  {  \_frameSkipQuantity = (int)nmrframeSkip.Value;  }   #endregion   #region Nhóm hàm xử lý chính  private void SelectVideoFile()  {  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();  }  }  private void StartPythonGRPCServer()  {  string scriptPath = Path.Combine(\_rootDir, "src", "Python", "GRPCServer.py");   \_pythonExecutor = new PythonExecutor("python", scriptPath);  \_pythonExecutor.Execute(string.Empty);  }  #endregion   #region Trích xuất Hình ảnh  private async Task ProcessImage()  {  ImageExtractor imageExtractor = new ImageExtractor(\_extractImageFolder);  imageExtractor.ExtractImages(\_videoPath, \_frameSkipQuantity);  try  {  DetectionResult detectionResult = new DetectionResult();  string[] allFiles = Directory.GetFiles(\_extractImageFolder);   foreach (string file in allFiles)  {  \_stopwatch = Stopwatch.StartNew();  \_stopwatch.Start();   var results = await gRpc(file);   \_stopwatch.Stop();   foreach (var box in results)  {  string label = box.label.ToString();  if (detectionResult.VehicleCounts.ContainsKey(label))  {  detectionResult.VehicleCounts[label]++;  }  else  {  detectionResult.VehicleCounts[label] = 1;  }   //// Tính toán vận tốc  //CalculateSpeed(box);  }   Invoke(new Action(() =>  {  pictureBox1.Image = DrawBoundingBoxes(file, results);  pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;  pictureBox1.Refresh();  }  )  );   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($"Có lỗi xảy ra: {ex.Message}");  }    }  private Bitmap DrawBoundingBoxes(string filePath, dynamic boundingBoxes)  {  using (Image img = Image.FromFile(filePath))  {  Bitmap bitmap = new Bitmap(img);  using (Graphics g = Graphics.FromImage(bitmap))  {  Color customColor = ColorTranslator.FromHtml("#33FF66");  using (Pen pen = new Pen(customColor, 2)) // Độ dày là 2  {  foreach (var box in boundingBoxes)  {  int x = box.x, y = box.y, w = box.w, h = box.h;  g.DrawRectangle(pen, x, y, w, h);  g.DrawString(box.label.ToString(), new Font("Arial", 10), new SolidBrush(ColorTranslator.FromHtml("#33FF66")), x, y - 24);  g.DrawString($"{CalculateSpeed(box)} km/h", new Font("Arial", 10), new SolidBrush(ColorTranslator.FromHtml("#ff2020")), x+ 40, y - 24);  }  }  }  return bitmap;  }  }   private double CalculateSpeed(dynamic box)  {  if (!\_calibrationDone)  {  // Hiệu chuẩn nếu chưa thực hiện  \_metersPerPixel = \_laneWidthMeters / 243;  \_calibrationDone = true;  }  // Tính tọa độ trung tâm của bounding box  float x\_center = box.x + box.w / 2;  float y\_center = box.y + box.h / 2;  var currentPos = new PointF(x\_center, y\_center);    int matchedId = -1;  double minDist = double.MaxValue;   // Tìm phương tiện phù hợp dựa trên khoảng cách  foreach (var objId in \_vehicleTracking.Keys)  {  var track = \_vehicleTracking[objId];  if (track.Positions.Any())  {  var lastPos = track.Positions.Last();  double dist = Math.Sqrt(Math.Pow(lastPos.X - currentPos.X, 2) + Math.Pow(lastPos.Y - currentPos.Y, 2));  if (dist < minDist && dist < 100) // Giới hạn khoảng cách tối đa để khớp  {  minDist = dist;  matchedId = objId;  }  }  }   // Nếu không tìm thấy phương tiện phù hợp, tạo ID mới  if (matchedId == -1)  {  matchedId = \_nextId++;  \_vehicleTracking[matchedId] = new VehicleTrackingInfo  {  Type = box.Label,  Positions = new List<PointF>(),  Timestamps = new List<double>(),  Velocities = new List<double>()  };  }   // Cập nhật thông tin theo dõi  var vehicleTrack = \_vehicleTracking[matchedId];  vehicleTrack.Positions.Add(currentPos);  vehicleTrack.Timestamps.Add(\_stopwatch.Elapsed.TotalSeconds);   double speed = 0.0;  // Tính vận tốc nếu có đủ dữ liệu  if (vehicleTrack.Positions.Count >= 2)  {  var prevPos = vehicleTrack.Positions[vehicleTrack.Positions.Count - 2];  var prevTime = vehicleTrack.Timestamps[vehicleTrack.Timestamps.Count - 2];  var currentTime = vehicleTrack.Timestamps.Last();  //var timeDiff = currentTime - prevTime;  double fps = 29.0; // Nếu bạn biết FPS của video  var timeDiff = (\_frameSkipQuantity + 1) / fps;    speed = CalculateSpeed(prevPos, currentPos, timeDiff);  vehicleTrack.Velocities.Add(speed);   var speedText = $"Vehicle ID: {matchedId}, Type: {box.label}, Speed: {speed} km/h";  }  return Math.Round(speed, 2);  }  private double CalculateSpeed(PointF pos1, PointF pos2, double timeDiff)  {  if (timeDiff < 0.001)  return 0.0;   double dx = pos2.X - pos1.X;  double dy = pos2.Y - pos1.Y;  double distancePixels = Math.Sqrt(dx \* dx + dy \* dy);  double distanceMeters = distancePixels \* \_metersPerPixel.Value;  double speedMps = distanceMeters / timeDiff;  double speedKmh = Math.Min(Math.Max(speedMps \* 3.6, 0), 200);   return speedKmh;  }  private async Task<List<dynamic>> gRpc(string imagePath)  {  try  {  // Địa chỉ server gRPC  using var channel = GrpcChannel.ForAddress("http://localhost:50051");   // Tạo client  var client = new ImageTransfer.ImageTransferClient(channel);   // Gửi yêu cầu tới server  var request = new ImageRequest { Path = imagePath, ModelPath = \_modelPath };  var response = await client.SendImageAsync(request);   // Xử lý phản hồi  if (response.Success)  {  Console.WriteLine("Message: " + response.Message);   // Phân tích JSON thành danh sách BoundingBox  List<dynamic> detections = JsonConvert.DeserializeObject<List<dynamic>>(response.Data);   // Trả về danh sách phát hiện  return detections;  }  else  {  Console.WriteLine("Error: " + response.Message);  return new List<dynamic>(); // Trả về danh sách rỗng nếu có lỗi  }  }  catch (Exception ex)  {  Console.WriteLine($"An error occurred: {ex.Message}");  return new List<dynamic>(); // Trả về danh sách rỗng nếu có ngoại lệ  }  }   #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  } } |
| --- |