NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme "Data Science and Business Analytics"

Research Project Report

on the topic

Long-Range Camera Guidance for Person Recognition in Public Spaces.

Fulfilled by the Student:

Student of the group БПАД213
Timchenko Daniil Gennadievich _____ (signature) _____ (date)

02.06.2024

Checked by the Project Supervisor:

Khelvas Alexander Valerievich
COO JSC COS&HT

_____ (signature) _____ (date)

Moscow 2024

Содержание

1 Description of the problem setting for the course work

1.1 Terminology

1.1.1 Focal Length

Focal length is a distance between "nodal point"(that is where light converges in a lens) and a camera sensor[2]. Cameras have a base focal length (max), but some cameras provide with a possibility to vary focal length by increasing/decreasing length of a camera lens. Thus a range of focal length ($F = (F_{min}, F_{max})$) is of interest, as applications imply usages with long focus lenses.

1.1.2 Image sensor

An image sensor refers to the electronic component in a digital camera that captures and converts light into digital signals, ultimately creating a digital image. The image sensor plays a crucial role in digital photography by replacing the traditional film used in film cameras. $U = (u_1, u_2)$ - sensor size of a camera in pixels represents number of pixels along x and y axes respectively, total image might have upmost $u_1 * u_2$ pixels, given that photo is RGB, it can be calculated, that on an 3-dimensional tensor with shape $(u_1, u_2, 3)$ the whole image can be stored, and on 4-dimensional tensor with shape $(u_1, u_2, 3, \textbf{frames})$ a whole video may be stored frome such camera without audio-stream, where frames - is the amount of frames taken from that camera consequently.

1.1.3 Angular Velocity

An angular velocity is the speed of rotation for an object that can be stated as $\omega = \dfrac{d\varphi}{dt}$.

1.1.4 Elevation

Vertical angle of an observed object over true horizon. Elevation combined with azimuth are used for obtaining the direction to an object. Elevation

1.1.5 Azimuth

Horizontal angle evaluated from predefined direction (for example north) and direction of an observed object.

1.1.6 PTZ Camera

is also known as pan-tilt-zoom camera is a type of cameras that is able to rotate across all 3 axis of rotation.

1.2 Work Plan

1) Familiarize oneself with the literature.
2) Formulate the goal and objectives of the work.
3) Transition from a camera view to a top view (Projective Geometry).
4) Solve the TSP problem for navigating players statically positioned on the field.
5) Develop a simulation for debugging and testing the long-focus camera control algorithm.
    а) Develop simulations of camera projection onto the field (3D).
    б) Develop an API to control the camera in the simulation.
    в) Integrate object behavior simulation with camera simulation.
6) Develop a metric that fairly evaluates the algorithm's performance.
7) Develop a camera control algorithm with the best metrics.
    а) Implement field traversal through points most visited by each player without predicting player movement (find the most frequent point for each player).
    б) Implement player traversal considering predictions of player movement.

8) Implement a PTZ camera model considering speed and acceleration constraints imposed on it. In the simplest case, neglect oscillations.

9) Summarize the results.

1.3 Assumptions

1. The input data is well-labeled and accurately reflects the true locations of objects, with IDs not getting mixed up.

2. One camera has a view of the entire field, while the other is a telephoto camera.

3. If a face is turned within ± 45 degrees towards the camera, it is considered possible to photograph (so positions of players are known at a given moment).

4. The height of the players is 170 cm, with the possibility of setting individual values.

5. If the face is captured within 150 pixels under the condition of point 3 and is not obstructed by other players, the player is photographed.

1.4 Problem Statement

Develop a mechanism for controlling a long-focus camera for efficient player detection and tracking within the field of view. A cyclic process is proposed, within which the camera automatically switches between players, adjusting the viewing angle and shooting distance. At the beginning of each iteration, the camera is focused on a new player, and the distance between them and the camera is optimized until the player's silhouette or face occupies a specified percentage of screen space - n% (with a tolerance). After that, the camera remains fixed on the player for a certain number of frames k before moving to the next player for the next iteration of the process.

When using the term 'efficiently' in this context, it implies that the algorithm should perform optimally on various datasets, such as video recordings of football matches with player labels and their positions in each frame. An algorithm tested on a subset taken from the population of football matches should perform traversal with shooting in the shortest time possible.

1.4.1 Coordinate Transformation

A projectivity from one projective plane to another is called a plane-to-plane projectivity, though it is often simply referred to as a projectivity. It operates on and produces a homogeneous 3-vector, thus represented as a 3-by-3 matrix.

To understand how such a projectivity occurs, consider two images taken from different viewpoints of a plane within a scene, as illustrated in Figure 1. The mapping of points to their corresponding points in image 1 is defined by a projectivity. Similarly, the mapping of points to their corresponding points in image 2 is defined by another projectivity. An essential characteristic of projectivities is that they form a group. Consequently, there exists a projectivity that describes the mapping from the image of the plane in image 1 to the image of the plane in image 2 where.

$$R = ST^{-1}$$

Given particular coordinates X, Y a plane-to-plane projective transformation can be done as following:

$$\begin{bmatrix} \tau_i X' \\ \tau_i Y' \\ \tau_i \end{bmatrix} = \underbrace{\begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix}}_{M} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Where a_i are elements of a scaling/rotation matrix, $\begin{bmatrix} b_2 & b_1 \end{bmatrix}^T$ is a translation vector and $\begin{bmatrix} c_1 & c_2 \end{bmatrix}$ is a projection vector.

To find true new coordinates X', Y' resulting vector has to be divided by τ_i that is the scaling factor.

1.4.1.1 Code implementation

Given source image field corner coordinates in a list corner_src_points, a projective transformation matrix can be calculated. Function cv2.getPerspectiveTransform takes 2 arguments: source (4 coordinates (x,y), resembling corners of the input quadrilateral) and

destination (4 coordinates (x,y), resembling corners of the output quadrilateral). On output the projective transformation matrix $M \in \mathbb{R}^{3\times3}$ described above is obtained.

```
Algorithm: GetProjectiveTransformMatrix
Input: base_size (integer), corner_src_points (list of tuples of integers)
Output: M (2D array)

1: Initialize base_size to 700  # arbitrary size of smallest side of new plane in pixels
2: Initialize whole_stadion_length to 105  # size of actual football stadium in meters
3: Initialize whole_stadion_width to 68
4: Calculate stadion_ratio as (whole_stadion_length / 2) / whole_stadion_width  # length to width ratio

Function GetProjectiveTransformMatrix(base_size: integer, corner_src_points: list of tuples of integers) -> 2D array:
    5: Set d1 to (0, 0)
    6: Set d2 to (0, base_size)
    7: Set d3 to (base_size * stadion_ratio, 0)
    8: Set d4 to (base_size * stadion_ratio, base_size)

    9: Set corner_dest_points to [d1, d2, d3, d4]

    10: Convert corner_src_points to a float32 2D array named source
    11: Convert corner_dest_points to a float32 2D array named dest
    12: Calculate the projective transformation matrix M using cv2.getPerspectiveTransform(source, dest)
    13: Return M
End Function
```

```
Algorithm: TransformCoordinates
Input: file_name (string, optional), player_labels_dataset_path (string, optional)
Output: new_coords (dataframe)

1: Define default values for file_name as 'track_df_new_coords.csv' and player_labels_dataset_path as 'yantar-230722-02_track.csv'

2: Function DownscaleDataFrame(df):
    # Implementation omitted for downscaling dataframe

3: Function ApplyProjectiveTransform(M, XY):
    # XY is an array of shape (3,), XY = [x, y, 1]
    # M is a matrix of shape (3, 3)
    3.1: XY_transformed <- M @ XY
    3.2: scaling_factor <- XY_transformed[2]
    3.3: unscaled_XY_transformed <- XY_transformed / scaling_factor
    3.4: Return unscaled_XY_transformed

4: Function GetPositionTransformed(frame):
    4.1: Initialize XY_init, id_count, and ones
    4.2: Concatenate XY_init with ones
    4.3: Initialize XY_transformed_frame as an empty list
    4.4: For i from 0 to length of id_count - 1:
        4.4.1: XY_trans <- ApplyProjectiveTransform(M, XY_init[i])
        4.4.2: Append XY_trans to XY_transformed_frame
    4.5: Return XY_transformed_frame

5: Load track_df from player_labels_dataset_path using CSV read function
6: Initialize track_df_transformed as an empty list
7: Compute shift as M @ [0, 0, 1]
8: Downscale track_df using DownscaleDataFrame function
9: Filter track_df to obtain rows where "frame" equals 1
10: Apply GetPositionTransformed to the filtered dataframe to get track_df_transformed

11: Modify new_coords:
    11.1: Subtract shift[0] from new_coords['x']
    11.2: Subtract shift[1] from new_coords['y']

12: Save new_coords to a CSV file with the given file_name
13: Return new_coords

End Algorithm
```

Now by using this function and warpPerspective function from opencv library, transformation can be done:

```
corrected_image = cv2.warpPerspective(image, M, (width, height),
    borderValue=(255,255,255))
transform_coordinates(file_name="unscaled_track_df_new_coords.csv")
```

1.5 Structure of the data

Given a football field on which there are players and a ball, specified by coordinates $\vec{X}$, as functions of time t, in the reference system associated with the field. So we have an input array X_i^{nm}, where

n - player number;

m - describes one of the coordinates of the player's position;

i - describes a moment in time.

It is necessary to build a high-resolution camera axis control function (reserve k for the camera number) with the given characteristics:

$F = (F_{min},F_{max})$ - focal length range;

$U = (u_1,u_2)$ - camera matrix size in pixels;

p - matrix pixel size in meters (real world size of an image sensor's pixel);

$\Omega = (\omega_1, \omega_2)$ - maximum angular velocity in elevation and azimuth;

$\frac{dF}{dt}_{max}$ - maximum rate of change of focal length over time.

Camera coordinates in the reference system associated with the far left corner of the field

$$W = \{w_1,w_2,w_3\}$$

1.6 The simplest task of controlling a high-resolution camera

It is necessary to propose an algorithm for bypassing all players on the field, starting from the center of the field.

As a result we should get:

$\vec{\psi}(t)$ is a vector describing the elevation angle and azimuth of the camera sighting as a function of time.

At the same time, we must ensure that the player's image is obtained in the camera's field of view during the time ΔT corresponding to R frames.

We consider the movement of the players to be a prior unknown.

1) The first step is to bypass stationary players

2) Second step - bypassing moving players

1.7 Approximations and limitations

Moving the camera angle up/down left/right and focusing are independent of each other and can be done in parallel. (The metric being optimized depends on this). We plan to have a different number of facilities and much greater than the number of football players. (The choice of algorithm depends on this - since the problem is NP hard, for a small amount it can be solved head-on - the 22 hypothesis is more optimal). The camera should return to the starting point (center by default).

2 Formal problem statement

2.1 The problem of traversing a set of moving points on a surface

The problem of fast traversal of dynamic systems is a critical problem in the context of surveillance, safety, logistics and other fields. In our concrete case, we aim to develop an algorithm, able to traverse with long-focus camera those dynamic agents in the shortest time possible (or close to it). Advancements in this research could help across different fields, such as human surveillance, sport stadiums, city safety and etc. Thus provided with the purpose let us dive into the formal statement of a problem.

Let $\mathbb{R}^3$ be the vector space, and $P_t = \{(x_1^{(t)},y_1^{(t)}), \ldots, (x_n^{(t)},y_n^{(t)})\}$ would be a set of observed objects existing in this vector space, that lie on a plane $z = 0$ in moment of time $t \in \mathbb{N}$ (x,$y \in \mathbb{R}$). Let $\mathcal{P}$

$$\mathcal{P}_t(\hat{x}, \hat{y}, \hat{z}, \phi_t, \psi_t, \theta_t,) \to \mathcal{V}$$

be the projection function that calculates corner points of FOV projection $\mathcal{V}_t \in \mathbb{R}^{3\times4}$ on $z = 0$ in the moment t. Here $\hat{x}, \hat{y}, \hat{z}$ - coordinates of a camera, ϕ_t, ψ_t, θ_t - yaw (azimuth), pitch (elevation) and roll in the time moment t (rotation coordinates).

Let $\mathcal{C}$

$$\mathcal{C}(P_t, \mathcal{V}_t) \to \begin{bmatrix}\Delta\phi_t & \Delta\psi_t & \Delta\theta_t\end{bmatrix}^T$$

be the controller function, that makes a decision on the controlling of camera direction and zoom for the time-step t+1. Camera rotations then are updated as following:

$$\begin{bmatrix}\phi_{t+1}\\ \psi_{t+1}\\ \theta_{t+1}\end{bmatrix} = \begin{bmatrix}\phi_t\\ \psi_t\\ \theta_t\end{bmatrix} + \begin{bmatrix}\Delta\phi_t\\ \Delta\psi_t\\ \Delta\theta_t\end{bmatrix}$$

Let $\mathcal{I}$

$$\mathcal{I}_t(\mathcal{P}_\sqcup, \mathcal{V}_t, p_1, p_2) \to a \in \{0,1\}$$

Be the indicator function, concluding if an observed object is taking up from p_1 to p_2 portion of space on a viewfinder AND never was observed in proper ratio before ($p_1 \leqslant p \leqslant p2$). a in this case is an indicator (True or False).

Then the constrained optimization problem looks as following:

$$\begin{cases} \sum\limits_{t=1}^{T} \mathcal{I}_t(\mathcal{P}_{\sqcup}, \mathcal{V}_t, p_1, p_2) \geqslant n \\ T \to \min\limits_{\mathcal{C}} \end{cases}$$

2.2 An alternative formulation of DTSP in the general case

DTSP is defined on a complete bidirectional graph $G = (V, E)$, where V is the set of vertices of size n and E is the set of edges. V consists of a depot 0 and a set of potential agents. We consider an asymmetric distance in DTSP. Thus, E includes edges in both directions. The agents to be visited are placed in a pool of agents C of size c, where C is a subset of V.

The salesman starts his journey from depot 0 at the beginning of time (t = 0). He must service each agent in the pool C exactly once and then return to the depot. The travel time from vertex i to vertex j depends on a time-dependent function $g_{ij}(t)$, where t is the time to visit vertex i. We assume that the seller does not wait at a vertex. This is true when the FIFO (First-In-First-Out) constraint is satisfied, i.e., it is guaranteed that if a vehicle leaves vertex i for vertex j at a certain time, any identical vehicle leaving vertex i for vertex j at a later time will arrive later at vertex j.

Let x_{ij} be a binary decision variable that equals 1 if the seller travels from vertex i to vertex j, and 0 otherwise.

Let s_i be the time when the seller visits vertex i. The objective is to minimise the total travel time to visit all agents, i.e.

$$\min \sum_{i \in \{0\} \cup C} \sum_{j \in \{0\} \cup C} g_{ij}(s_i) x_{ij}$$

Set of constraints:

$$\sum_{j \in \{0\} \cup C} x_{ij} = 1 \quad \forall i \in C \tag{2.1}$$

$$\sum_{i \in \{0\} \cup C} x_{ji} = 1 \quad \forall i \in C \tag{2.2}$$

$$s_0 = 0 \tag{2.3}$$

$$s_i + g_{ij}(s_i)x_{ij} = s_i + (s_j - s_i)x_{ij} \tag{2.4}$$

$$\forall i \in \{0\} \cup C, j \in C \tag{2.5}$$

$$x_{ij} \in \{0,1\} \tag{2.6}$$

Constraints (2) and (3) ensure that there is only one incoming and outgoing vertex for agent i. Constraint (4) is the initial time of the commit-merchant in the depot. Constraints (5) specify that the visit time of agent j depends on the visit time of its predecessor i. This set of constraints also guarantees that the visit time at each vertex increases along the path (provided that $g_{ij}(t) > 0$). Hence, there is no sub-cycle in the solution.

The model expressed by formulas (1)-(6) is essentially a formulation of TDTSP. It is nonlinear because of the time-dependent function $g_{ij}(t)$. Some researchers try to linear-ize the formulation by imposing additional assumptions. In contrast, this formulation describes the most generalized version. Note that the domain $g_{ij}(t)$ is continuous. For ease of data collection, the time space can be discretised into a set T of time steps. Thus, we have the traveling time from vertex i to vertex j around a time step $t \in T$ as input values denoted as $d_{ij}(t)$. Here we call $[d_{ij}(t)]$ the traffic pattern of the graph G. We can then approximate $d_{ij}(t)$ by working with d_{ijt}.

TDTSP assumes that all conditions of the graph dynamics are known in advance. In practice, to cope with the dynamic environment, we introduce a stochastic variable $\phi_{ij}(t)$ in addition to $g_{ij}(t)$ to deal with the uncertainty of the actual traveling time. Then the actual traveling time from vertex i to vertex j at time t, denoted as $f_{ij}(t)$, is $f_{ij}(t) = g_{ij}(t) + \phi_{ij}(t)$.

To address the other uncertainty problem, i.e., changing agent queries in a dynamic environment, we introduce a random operation Ω_k after the camera finishes inspecting the k-th agent, denoted as

$$\Omega_k = \begin{cases} 1, & \text{insert unvisited agent } i \text{ into set } C \\ 0, & \text{do nothing} \\ -1, & \text{remove agent } i \text{ from set } C \end{cases}$$

DTSP is an online optimisation task. Solving it efficiently is very difficult. Considering the scaling problem $n = 40$ of a graph G with invariant location. If $c = 20$, the number of possible instances is also huge. When the two dynamic aspects mentioned above are taken into account, the problem becomes even more difficult.

3 Literature Review

3.1 Description of Relevant Literature

3.1.1 Temporal Graphs

• [3] Each edge $e = (u, v, t) \in E$ is a temporal edge from a vertex u to another vertex v at time t. For any two temporal ages (u,v,t_1) and (u,v,t_2) $t_1 \neq t_2$.

• Each vertex $v \in V$ is active when there is a temporal edge that starts or ends at v.

• $d(u, v)$: the number of temporal edges from u to v in G.

• $E(u, v)$: the set of temporal edges from u to v in G, i.e., $E(u,v) = \{(u,v,t_1),(u,v,t_2), ..., (u,v,t_{d(u,v)})\}$.

• $N_{out}(v)$ or $N_{in}(v)$: the set of out-neighbors or in-neighbors of v in G, i.e., $N_{out}(v) = \{u : (v, u, t) \in E\}$ and $N_{in}(v) = \{u : (u, v, t) \in E\}$.

• $d_{out}(v)$ or $d_{in}(v)$: the temporal out-degree or in-degree of v in G, defined as $d_{out}(v) = \sum_{u \in N_{out(v}} d(v,u)$ and $d_{in}(v) = \sum_{u \in N_{in(v}} d(u,v)$.

3.1.2 Sub-types of TSP

Based on the book "The Traveling Salesman Problem and Its Variations,"[4] two variations of TSP stand out as most relevant to our problem:

Moving Target TSP: We have a collection $X = \{x_1, x_2, \ldots, x_n\}$ of n objects, each located at positions $\{p_1, p_2, \ldots, p_n\}$. Each object x_i moves from its position p_i in $\mathbb{R}^2$ with velocity v_i. A pursuer starts at the origin and moves at speed v with the objective of intercepting all objects $x_1, x_2, \ldots, x_n$ as quickly as possible. This problem relates to the time-dependent TSP.

Time-Dependent TSP: In this scenario, each arc (i, j) in G has n different costs $c_{ij}^t = 1, 2, \ldots, n$. The cost c_{ij}^t indicates the cost of traveling from city i to city j during time period t. The aim is to determine a tour $(\tau(1), \tau(2), \ldots, \tau(n), \tau(1))$, where $\tau(1) = 1$ is the home location at time period zero, in G such that $\sum_{i=1}^{n} c_{\tau(i)\tau(i+1)}^{t_i}$ is minimized. Here, $n + 1$ is equivalent to 1. If $c_{ij}^t = c_{ji}^t = \ldots = c_{ij}^{t'}$ for all (i, j), the time-dependent TSP reduces to the traditional traveling salesman problem.

In our case, it seems that the problem encompasses both sub-types.

In the article [5], the dynamic TSP with variable weights is explored. The impact of these changes on the problem's fitness landscapes is examined.

The paper addresses key questions about the dynamic TSP, such as "how many solutions are affected by a change?"and "how does the severity of the problem influence the optimal solutions?"

The study includes simulations of the dynamic TSP with weight changes, revealing that the new optimal solutions are usually close to the previous ones.

In [6], a robust algorithm for solving DTSP is introduced. Experimental results demonstrate that this algorithm is highly effective, producing high-quality solutions in very short time steps.

3.1.3 Deep Reinforcement Learning

In 2023, an article [7] was published that formulates a problem similar to the one considered in this work. It turned out that our problem is more related to Time-Dependent TSP (TDTSP) rather than Dynamic TSP (DTSP), as initially assumed. The authors used a Deep Reinforcement Learning approach to solve this problem, introducing an additional complication: new vertices can disappear and appear in the process, which may also be relevant for a football field. Provided the data issue (ensuring a sufficient volume of data for training an RL model with an attention mechanism) is resolved, the authors' methodology can be adapted to our case.

3.1.4 Vehicle Routing Problem (VRP)

Problem Statement of VRP: The Vehicle Routing Problem (VRP) ([8] [9]) is an optimization problem aimed at finding the optimal routes for a fleet of vehicles, taking into account several factors such as time, cost, and load capacity. Formally, in VRP, there is a given number of vehicles, each with constraints on load capacity and working hours. There is also a set of client points that need to be visited, and for each point, the load and service time

requirements are known. The objective of VRP is to optimally allocate the client points among the vehicles to minimize the total costs, considering all constraints.

Difference from TSP: The main difference between TSP and VRP is that in TSP, there is one salesman who must visit a set of cities and return to the starting point, minimizing the total distance. In VRP, there are multiple vehicles, each needing to service a set of client points with specific constraints. Therefore, in VRP, it is necessary to optimize the allocation of client points among the vehicles considering various factors such as load capacity, service time, and throughput, making this problem more complex compared to TSP. The VRP problem statement may be relevant in the case of multiple observing cameras.

3.1.5 CGN (Convolutional Graph Networks) for TSP

The article [10] discusses the use of GNN for optimizing the route when multiple cities need to be visited and returned to the starting point. The basics of the problem, the application of GNN to it, and the implementation details of the model using Graph Transformer and Residual Gated GCN are discussed. It concludes that the model demonstrates the ability to find optimal routes but requires labeled data, and a promising direction for development could be a transition to reinforcement learning. Labeled data are obtained using the Concorde TSP Solver, meaning that optimal paths for graphs need to be calculated first, and then the neural network training can begin.

3.1.6 Graph Neural Networks

Graphs are pervasive in our world, with many real-world entities being defined by their connections to others. These collections of objects and their interconnections can be naturally represented as graphs. For more than a decade, researchers have been developing neural networks tailored to graph data, known as graph neural networks (GNNs). Recent advancements have significantly enhanced their capabilities and expressive potential. Today,

GNNs are finding practical applications in diverse fields such as antibacterial discovery, physics simulations, fake news detection, traffic prediction, and recommendation systems.

This article [11] delves into the intricacies of modern graph neural networks. We structure our discussion into four sections. Initially, we examine the types of data that are most appropriately represented as graphs, along with some typical examples. Next, we discuss what sets graphs apart from other data types and the unique considerations required when working with them. In the third section, we construct a modern GNN, detailing each component of the model and tracing the evolution of key innovations in the field. We progress from a basic implementation to a cutting-edge GNN model. Finally, we offer a GNN playground, allowing you to experiment with a real-world task and dataset to better understand how each element of a GNN model influences its predictions.

3.1.7 Linearization of TDTSP (integer programming) [1]

Papers [1] and [12] showed the possibility of formulating the time-dependent traveling salesman problem, as an integer programming problem, diving deeper on the first of those papers here. Given the Time-Dependent Traveling Salesman Problem (TDTSP), the task is to find a Hamiltonian tour with the shortest total duration, where traversal times between vertices vary over time. This paper makes two main contributions. First, it introduces a lower and upper bounding procedure that involves solving a simpler, though still NP-hard, asymmetric traveling salesman problem (ATSP). Additionally, it is demonstrated that this ATSP solution is optimal for the TDTSP when all arcs exhibit a common congestion pattern. Second, the paper formulates the TDTSP as an integer linear programming model and develops valid inequalities for this model. These inequalities are incorporated into a branch-and-cut algorithm capable of solving instances with up to 40 vertices.

3.1.8 Genetic ant colony algorithms

The Dynamic Traveling Salesman Problem (DTSP) is a complex optimization challenge that traditional methods struggle to solve. Numerous approaches have been proposed in the literature, each with its own strengths and weaknesses. Among these, Genetic Algorithms (GA) and Ant Colony Optimization (ACO) have proven effective for tackling the DTSP. This paper [13] introduces a novel hybrid algorithm that combines GA and ACO to provide an improved solution for the DTSP. The hybrid algorithm focuses on the suitability of the method and the travel distance for the DTSP. The results indicate that the hybrid algorithm avoids easily settling into local optima and demonstrates a good convergence speed towards an optimal solution.

4 Solution Description

4.1 Coordinate Systems Used

At this stage, two coordinate systems are used: the first is related to the football field, where the top-left corner is considered the point (0, 0), and the second is related to the camera coordinates, in a reference system linked to the far left corner of the field. For given coordinates X, Y, the transition from one plane to another is carried out according to the following law:

$$\begin{bmatrix} \tau_i X' \\ \tau_i Y' \\ \tau_i \end{bmatrix} = \underbrace{\begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix}}_{M} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix},$$

where a_i are scaling/rotation elements, $\begin{bmatrix} b_2 & b_1 \end{bmatrix}^T$ is the shift vector, and $\begin{bmatrix} c_1 & c_2 \end{bmatrix}$ is the projection vector.

To find the true values of X', Y', the resulting vector must be divided by the coefficient τ_i, which is the scaling factor.

As a result, an algorithm was developed using the cv2 library, which takes two arguments as input: four initial coordinates (x, y), which indicate the initial corners of the rectangle of the transition area, and the second argument - four coordinates that indicate the desired corner coordinates for the output image. The algorithm produces a transformation matrix $M \in \mathbb{R}^{3\times3}$, after which the algorithm is sequentially applied to each frame from the initial dataset. As a result, a new dataset is obtained in which for each id, i.e., the recognized object in the frame, new coordinates (X', Y') are obtained. An example visualizing the algorithm's work:

4.2 Camera Simulation

4.2.1 Line - Plane Intersection

Intersection of a plane with a line: Let p_0 be the camera vector, and v_0 be the direction vector of the light ray passing through the camera vector (center of the focal lens).

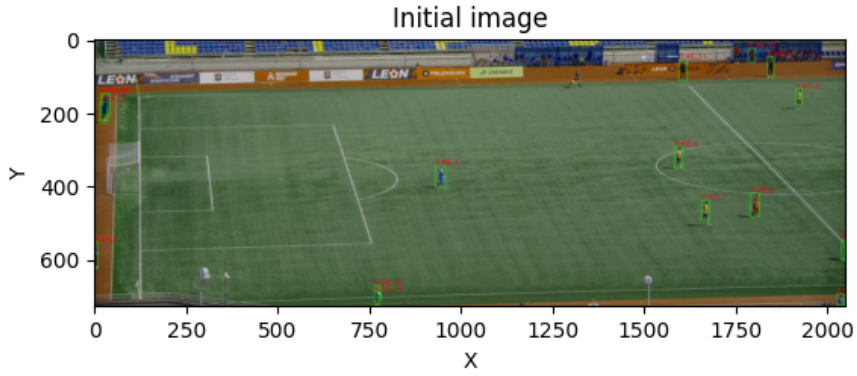

Рисунок 4.1 — Image before transformation of coordinates

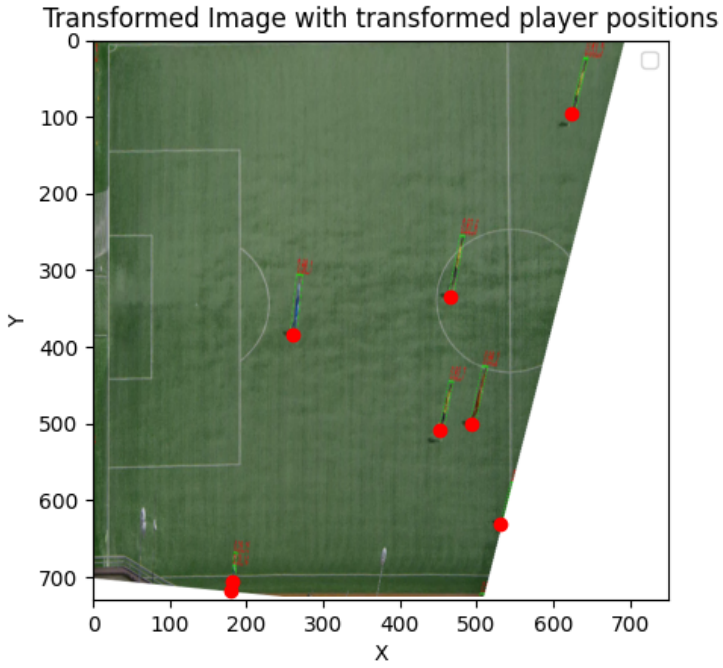

Рисунок 4.2 — Image after transformation of coordinates

Then the parametric equation of the line can be defined as:

$$P(t) = p_0 + t \cdot v_0 \qquad (P(t) \text{ is a point on the line})$$

We need to find such a t that $P(t)$ lies on a given plane. Substituting this into the plane equation, we get:

$$a(P0_x + tV0_x) + b(P0_y + tV0_y) + c(P0_z + tV0_z) = d$$

Solving for t (n is the normal vector of the plane):

$$t = \frac{D - nP0}{n \cdot V0}$$

We are interested in $t < 0$ since positive values correspond to the wrong light direction.

4.2.2 Field of View and Angle of View

For truthful simulation of field of view and physics of long-focus cameras, work of Matvey Gancev was used (Модель для панорамной камеры высокого разрешения). As simulation of physics for camera is a complicated and time-requiring task, with the admission of Matvey Gancev, the simulation code was used. However modeling the camera movement and traversal algorithm are completed without use of intellectual property from other researchers. It is a complicated task to set the angle velocity, thus it can be estimated with a velocity on a euclidean plane.

Field of View: The calculatePanoramicSystemFOV method calculates the angle of view (FOV) for each camera in the panoramic system and returns a list of camera angles of view for the panoramic system.

The method defines the plane vector and the coordinates and angles of the panoramic system. Then for each camera in the panoramic system, the following occurs:

1) The camera coordinates and its angles of view are calculated.

2) Vectors defining the camera's angles of view are created.

3) Rotations are applied to the vectors of the camera and the panoramic system.

4) The intersection point of the line (defined by the camera) with the plane (panoramic system) is calculated.

5) The camera's angle of view and the main axis of the camera's view are calculated.

1. Calculation of vector a:

$$\text{vector\_a} = \begin{bmatrix} 1.0 \\ \tan\left(\frac{\text{camera\_width\_angle\_of\_view}}{2}\right) \\ -\tan\left(\frac{\text{camera\_height\_angle\_of\_view}}{2}\right) \end{bmatrix}$$

2. Rotation of vector **a**:

$$\mathbf{a} = (\mathbf{R}_{\text{ps}} \cdot \mathbf{R}_{\text{c}}) \cdot \mathbf{a}$$

Here $\mathbf{R}_{\text{ps}}$ and $\mathbf{R}_{\text{c}}$ are the rotation matrices for the panoramic system and the camera, respectively.

3. Re-rotation of vectors **b**, **c**, **d**, **p**:

$$\mathbf{b} = (\mathbf{R}_{\text{ps}} \cdot \mathbf{R}_{\text{c}}) \cdot \mathbf{b}$$
$$\mathbf{c} = (\mathbf{R}_{\text{ps}} \cdot \mathbf{R}_{\text{c}}) \cdot \mathbf{c}$$
$$\mathbf{d} = (\mathbf{R}_{\text{ps}} \cdot \mathbf{R}_{\text{c}}) \cdot \mathbf{d}$$
$$\mathbf{p} = (\mathbf{R}_{\text{ps}} \cdot \mathbf{R}_{\text{c}}) \cdot \mathbf{p}$$

Here $\mathbf{R}_{\text{ps}}$ and $\mathbf{R}_{\text{c}}$ are also the rotation matrices for the panoramic system and the camera.

4. Calculation of point $\mathbf{t}_0$:

$$\mathbf{t}_0 = \mathbf{r}_{\text{ps}} + \mathbf{R}_{\text{ps}} \cdot \mathbf{r}_{\text{c}}$$

Here $\mathbf{r}_{\text{ps}}$ and $\mathbf{r}_{\text{c}}$ are the coordinates of the panoramic system and the camera, respectively.

5. Definition of parameters a_0, b_0, c_0, d_0, p_0:

$$a_0 = -\frac{(\mathbf{v}_{\text{plane}} \cdot \mathbf{v}_{\text{t}_0}) + D}{(\mathbf{v}_{\text{plane}} \cdot \mathbf{v}_a)}$$
$$b_0 = -\frac{(\mathbf{v}_{\text{plane}} \cdot \mathbf{v}_{\text{t}_0}) + D}{(\mathbf{v}_{\text{plane}} \cdot \mathbf{v}_b)}$$
$$c_0 = -\frac{(\mathbf{v}_{\text{plane}} \cdot \mathbf{v}_{\text{t}_0}) + D}{(\mathbf{v}_{\text{plane}} \cdot \mathbf{v}_c)}$$
$$d_0 = -\frac{(\mathbf{v}_{\text{plane}} \cdot \mathbf{v}_{\text{t}_0}) + D}{(\mathbf{v}_{\text{plane}} \cdot \mathbf{v}_d)}$$
$$p_0 = -\frac{(\mathbf{v}_{\text{plane}} \cdot \mathbf{v}_{\text{t}_0}) + D}{(\mathbf{v}_{\text{plane}} \cdot \mathbf{v}_p)}$$

6. Calculation of the coordinates of the angles of view and the main axis of the camera:

$$\begin{aligned}
\text{fov_a} &= a_0 \cdot \mathbf{v}_a + \mathbf{v}_{t_0} \\
\text{fov_b} &= b_0 \cdot \mathbf{v}_b + \mathbf{v}_{t_0} \\
\text{fov_c} &= c_0 \cdot \mathbf{v}_c + \mathbf{v}_{t_0} \\
\text{fov_d} &= d_0 \cdot \mathbf{v}_d + \mathbf{v}_{t_0} \\
\text{main_axis} &= p_0 \cdot \mathbf{v}_p + \mathbf{v}_{t_0}
\end{aligned}$$

These transformations are necessary to calculate the parameters of the angles of view and the main axis of the camera in the panoramic system. They perform transformations of coordinates and vectors, taking into account their initial positions and rotations relative to the coordinate system associated with the panoramic system. Then, using the found parameters, the points indicating the angles of view of each camera are determined, as well as the point representing the main axis of the camera. These steps allow us to determine the position and direction of view of each camera in the context of the panoramic system.

Angle of View: This method calculates the vertical angle of view of the camera. Let's consider the mathematics of this process.

Let f be the focal length of the camera lens, and H be the height of the camera's focal plane. We want to find the vertical angle of view θ_{vertical}, which determines how many degrees vertically the camera covers.

Using the theorem of similar triangles, we understand that the vertical angle of view can be expressed as double the arctangent of the ratio of the height of the focal plane to twice the focal length:

$$\theta_{\text{vertical}} = 2 \times \arctan\left(\frac{H}{2f}\right)$$

Thus, we find the angle at which the image on the focal plane is visible relative to the central axis of the camera. This gives us an idea of what portion of the vertical space is covered by the image.

4.2.3 Determining Δ Yaw and Δ Pitch during Movement

To simulate the camera and the algorithm as a whole, it is important to have the ability to control the camera. To control the camera in our study, we can send $\Delta\theta$ and $\Delta\phi$ (yaw and pitch changes) to our camera (simulated or real).

1. Central point of the field of view angles:

Find the central point between two field of view (FOV) angles:

Let P_1 and P_2 be the two field of view angles of the camera, then the central point C will be:

$$C = \left(\frac{P_{1x} + P_{2x}}{2}, \frac{P_{1y} + P_{2y}}{2} \right)$$

2. Calculation of the angle:

Calculate the tilt angle of the camera relative to the horizon and the vertical angle of view of the camera:

Let (x_c, y_c) be the coordinates of the camera, (x_i, y_i) the initial position, and (x_t, y_t) the target position. Also, h is the height of the camera, and θ_{pitch} the vertical angle of view, θ_0 the current vertical tilt angle, and $\theta_{\text{top\_aov}}$ the angle between the highest incoming light ray on the camera and the main axis of the camera ($AOV_{\text{vert}}/2$).

First, determine the camera's yaw angle, using direction vectors from the camera to the initial and target positions:

$$\Delta\theta_{\text{yaw}} = \text{atan2}(y_t - y_c, x_t - x_c) - \text{atan2}(y_i - y_c, x_i - x_c)$$

Then calculate the pitch angle, which is the tilt angle of the camera relative to the horizon for:

$$\theta_{\text{pitch}} = 90^\circ - \text{toDegrees}(\arctan(\frac{|(x_t - x_c, y_t - y_c)|}{h}))$$

$$\Delta\theta_{\text{pitch}} = \theta_{\text{pitch}} - \theta_{\text{top\_aov}} - \theta_0$$

Here, atan2 is the arc-tangent function of two arguments, $\| \cdot \|$ is the vector norm, and $\mod$ is the modulo operation.

4.2.4 Player Detection inside of a Field of View

It is already mentioned, that the algorithm is able to infer how close the center of FOV to the target position, but the algorithm also have a player detection module for future algorithm enhancements (for example: awaiting for the moment when player view is not obstructed by other players).

```
Algorithm: is_agent_inside_fov
Input: Point (x, y), Tetragon [point1, point2, point3, point4]
Output: True if the point is inside the fov, otherwise False

1: begin
2:     Initialize windingNumber <- 0
3:     Initialize tetragon <- [point1, point2, point3, point4]

4:     for i from 0 to tetragon.size - 1 do
5:         x1, y1 <- tetragon[i]
6:         x2, y2 <- tetragon[(i + 1) mod tetragon.size]

7:         if y1 <= y then
8:             if y2 > y then
9:                 if is_left_of_line((x1, y1), (x2, y2), (x, y)) then
10:                    windingNumber <- windingNumber + 1
11:                end if
12:            end if
13:        else
14:            if y2 <= y then
15:                if is_left_of_line((x1, y1), (x2, y2), (x, y)) then
16:                    windingNumber <- windingNumber - 1
17:                end if
18:            end if
19:        end if
20:    end for

21:    return (windingNumber != 0)
22: end
```

4.3 Algorithm Development for the Case of Moving Players

4.3.1 Master-Route Baseline

The basic algorithm is to pass the camera over the field in a "snake"pattern. The camera sequentially surveys each strip of the observable field, moving from the left edge of the field to the right, and then from

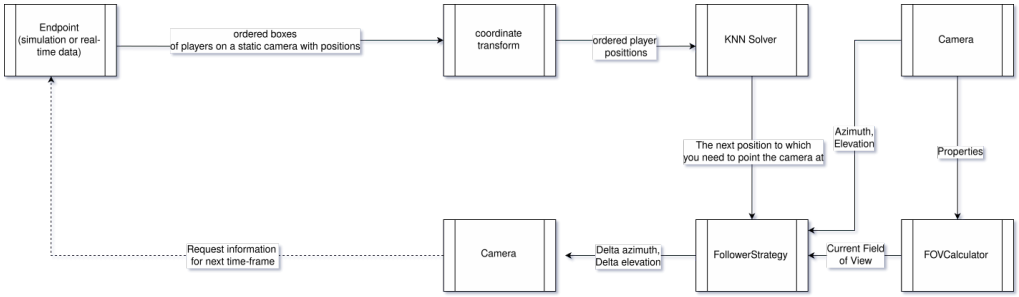

Рисунок 4.3 — General Pipeline

the right edge of the field to the left. This algorithm is easily implemented once the framework from section 4.4.2 has been implemented. To cover all players, the camera can be approximated along the path when the algorithm determines that the player is close enough to its initial territory.

4.3.2 KNN-Greedy Algorithm

```
Algorithm: Camera Movement to Track Players
Input: List of players with coordinates
Output: Camera movement plan to visit each player

1: Initialize camera
2: Set camera aim to the result of calc_principal_axis_intersection()
3: Initialize unvisited_players as a copy of players

4: while there are unvisited players do
5:     player <- get_closest_player(camera.aim, unvisited_players)
6:     Remove player from unvisited_players
7:     camera_target_aim <- (player.x, player.y)
8:     movement_plan_array <- generate_linear_trajectory(camera.aim,
   camera_target_aim)

9:     i <- 0
10:    while player is not visited by camera for 3 seconds do
11:        camera.phi <- camera.phi + calc_delta_phi(camera, camera.aim,
   movement_plan_array[i])
12:        camera.theta <- camera.theta + calc_delta_theta(camera,
   camera.aim, movement_plan_array[i])
13:        i <- i + 1
14:    end while
15: end while

Function calc_delta_phi(camera, cur_aim, target_aim)
    # Calculate the yaw adjustment
    1: init_vec <- [cur_aim.x - camera.x, cur_aim.y - camera.y]
    2: target_vec <- [target_aim.x - camera.x, target_aim.y - camera.y]
    3: init_vec_angle <- vec_to_angle(init_vec)
    4: target_vec_angle <- vec_to_angle(target_vec)
    5: return target_vec_angle - init_vec_angle
End Function

Function calc_delta_theta(camera, cur_aim, target_aim)
    # Calculate the pitch adjustment
    1: top_aov <- camera.pitch - camera.vertical_aov
    2: target_vec <- [target_aim.x - camera.x, target_aim.y - camera.y]
    3: ratio <- norm(target_vec) / norm(camera.height)
    4: raw_pitch <- to_degrees(arctan(ratio)) % 360.0
    5: pitch <- 90 - raw_pitch
    6: return pitch - top_aov
End Function
```

In the above fragment, the main algorithm is described.

4.3.3 Probability-Density-Graphs TSP

As a future enhancement such algorithm could be developed:

Assume we know the player graph at time t. The features of the vertices in this situation will be the coordinates of the players as well as identifiers. We train an algorithm that can predict the probabilistic distribution of each player's position over the sequence [0,t] (the simplest case being a 2-dimensional Gaussian, i.e., standard normal distribution), then the goal for the camera can be set to focus on the 95% confidence interval at time t+k, which can be calculated using the camera's angular velocity. The camera then focuses on the vertex so that the entire area is visible with sufficient confidence and zooms in on a specific player (linear interpolation of their movement can be included here). The algorithm then repeats for all players.

5 Results

5.1 Evaluation of Camera Traversal Algorithm

5.1.1 Metric and Experiment formulation

To evaluate the algorithm, the metric chosen is time required to traverse through all players with stopping time on a player equal to $t_{stop} = 5$. Number of players $n_{players} = 22$ and $n_{iter} = 100$. An experiment will be ran n_{iter} of times, and the T will be the metric for the according algorithm:

$$T = \frac{1}{n_{iter}} \sum_{i=1}^{n_{iter}} t_i$$

Note that t_i is measured in simulation ticks, with conversion formula $25 \cdot t_i = 1$ second, giving FPS (frames per second) to be 25 ($FPS = 25$). Also statistics like standard deviation, inter-quantile range and histogram will be displayed for further analysis. Given the fact that PTZ cameras are often able to gather angular velocity up to 30 ° per second (Source), given that in our case approximately 90 ° of horizontal panning is enough to traverse the field and also we can not expect camera to be always on max speed, we can approximate an average speed as 18 ° per second. Given such speed, it is possible to traverse the field in 5 seconds, that way yielding that on average linear speed of a camera may be roughly approximated to 20 meters per second or 0.8 meters per one frame given a FPS of 25 frames.)

Given such framework of evaluation, it is possible to compare quality of a baseline to the quality of the developed KNN-Greedy approach. (Possibly delete a part about comparing)

5.1.2 Dataset

To assure that experiments are representative of the real-world scenario, a soccer match simulation from Timur Khaibrakhmanov was used. In the simulation, player movement is modeled with similar to real-world laws. Despite that simulation quite accurately represents football matches, it has a randomization component, that allows to create $n_{iter} = 100$ diverse simulations, that represent the soccer dynamics. On those generated

simulations, the algorithm is tested. Algorithm only possesses the information prior to during timestamp, thus it simulates real-world unpredictability of players' behavior.

5.1.3 Result of the Experiment

Таблица 5.1 — Statistics of the KNN Greedy Performance for $N = 100$ in frames

Statistic	Frames for Complete Traversal
Count	100.00000
Mean	**408.56000**
Standard Deviation (std)	46.73919
Minimum (min)	305.00000
25th Percentile (25%)	375.75000
Median (50%)	412.50000
75th Percentile (75%)	442.25000
Maximum (max)	549.00000

Таблица 5.2 — Statistics of the KNN Greedy Performance for $N = 100$ in seconds

Statistic	Seconds for Complete Traversal
Count	100.00000
Mean	**16.3424**
Standard Deviation (std)	1.8695676
Minimum (min)	12.2
25th Percentile (25%)	15.03
Median (50%)	16.5
75th Percentile (75%)	17.69
Maximum (max)	21.96

Using the KNN Greedy algorithm yields mean of 16.3424 seconds for traversal of moving players in the case of 22 players on a 100 by 100

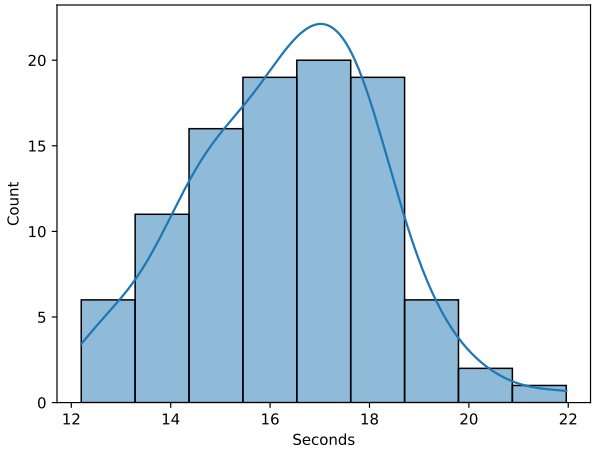

Рисунок 5.1 — Histogram of experiment in seconds

meters region. There are some outliers, but they have relatively low deviation from the sample mean, minimum (fastest time) is considered 12.2 seconds, and maximum is considered 21.96 seconds. Standard deviation is almost 2 second, that equates to approximately 15% of mean, which shows robustness of the algorithm. KDE plot is similar to the bell curve of normal distribution, however left tail is much heavier, that can be interpreted as that there are a lot of surprisingly faster experiments and not quite a lot of surprisingly slow experiments (in terms of time taken to traverse all agents). One could also add that at value of 19 seconds it seems like there is some barrier that makes it hard for the simulation to take longer than this time, although it is just a hypothesis and may be due to simulation specifics.

Yaw pitch dynamic graph 5.2 represents how angle may differ through a simulation. What is worth of noting, is how pitch (tilt) changes only between 6 and 18 degrees, while yaw (panning) is spanning between 160 to 220 degrees. It gives us a hint, of how sensitive the camera FOV to the tilt compared to panning. Those 2 graphs completely describe the dynamics of a camera movement in a single simulation. Times when camera stops at a single point are almost non-visible, as it happens only for a fraction of 5 frames. There also seen the fast spikes on pitch graph, it is for now unknown what is the reasoning of such behavior, but one is certain, it may not be caused by knn-greedy algorithm.

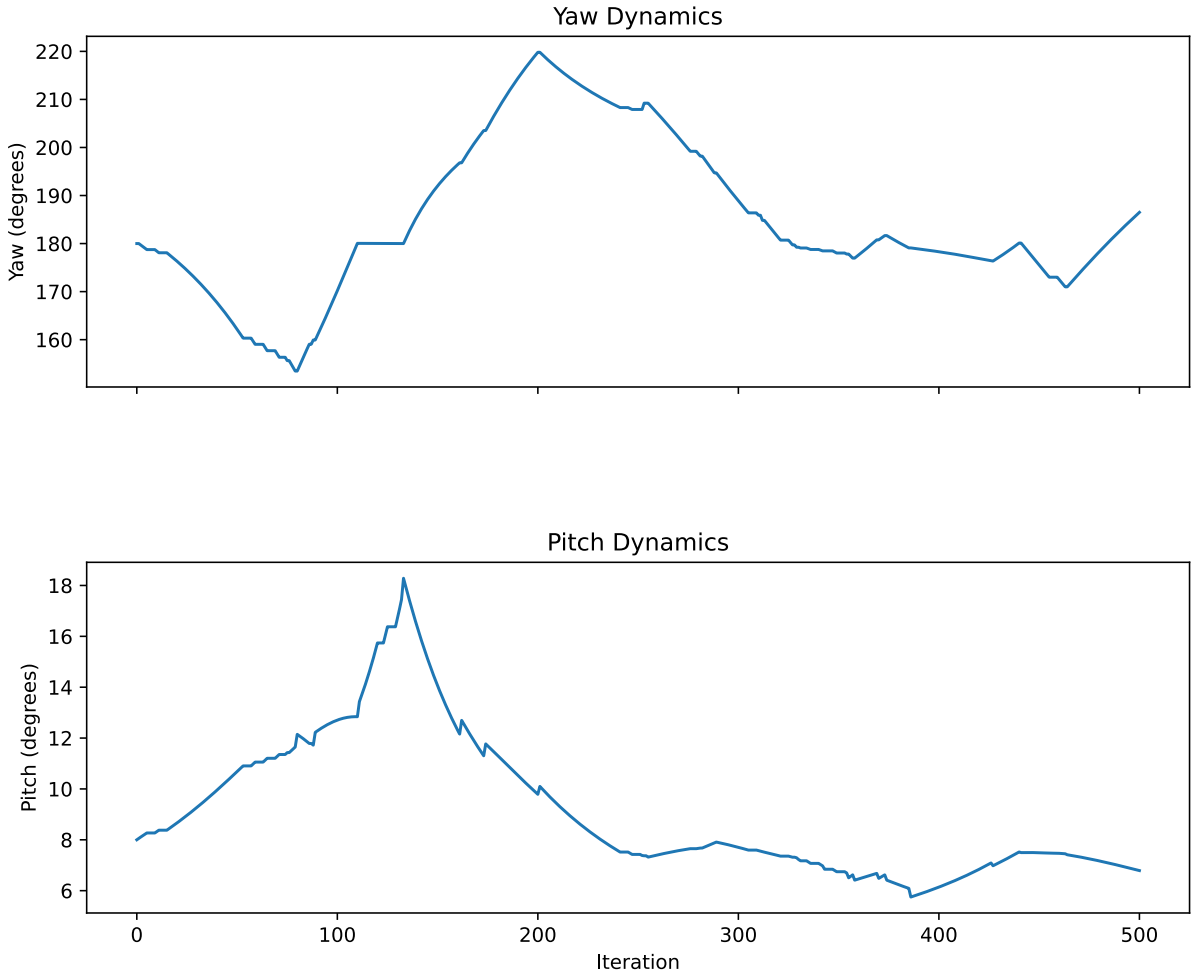

Рисунок 5.2 — Yaw-Pitch Dynamics during the first simulation

5.2 Conclusion

In this study, we endeavored to develop and assess a camera traversal algorithm tailored to the task of tracking moving players in a simulated soccer match setting. Through a systematic investigation encompassing algorithmic design, simulation, and empirical evaluation, we aimed to address the multifaceted challenges inherent in efficient field traversal and player tracking.

The foundational phase of our inquiry involved establishing rigorous coordinate systems to facilitate seamless integration between the soccer field and camera perspectives. By elucidating the transformation matrix and elucidating the interrelation between coordinate planes, we ensured a robust foundation upon which subsequent algorithmic developments could be built.

Subsequently, we delved into the intricacies of camera simulation, including the precise determination of line-plane intersections, field of view calculations, and dynamic adjustments of yaw and pitch angles during traversal. These components were pivotal in emulating realistic camera behaviors, thereby enabling the faithful reproduction of real-world scenarios within our computational framework.

The crux of our investigation lay in the algorithmic development phase, where we explored two distinct methodologies: the master-route baseline and the KNN-Greedy algorithm. The former provided a structured approach to field traversal, while the latter introduced a dynamic heuristic based on nearest-neighbor principles. Through meticulous experimentation and evaluation, we discerned significant insights into the performance and efficacy of each approach.

Our empirical findings revealed that the KNN-Greedy algorithm exhibited commendable efficiency, with an average traversal time of approximately 16.34 seconds and a standard deviation of 1.87 seconds across 100 simulation iterations. These results underscore the algorithm's robustness and efficacy in dynamically tracking moving players while navigating the soccer field.

Furthermore, our analysis extended beyond quantitative performance metrics to encompass qualitative aspects such as yaw-pitch dynamics. By visualizing the camera's orientation throughout the simulation, we gained valuable insights into its adaptability and responsiveness to evolving game scenarios, thereby elucidating avenues for further refinement and optimization.

In summation, this study represents a rigorous exploration of camera traversal algorithms in the context of tracking moving players in simulated soccer matches. By leveraging mathematical principles, computational techniques, and empirical evaluation, we have developed a sophisticated framework that bridges theoretical concepts with practical applications. Moving forward, continued research and refinement of these algorithms hold the promise of advancing the efficiency, accuracy, and versatility of camera systems in diverse sporting and surveillance domains.

Литература

1. *Cordeau, Jean-François.* Analysis and Branch-and-Cut Algorithm for the Time-Dependent Travelling Salesman Problem / Jean-François Cordeau, Gianpaolo Ghiani, Emanuela Guerriero // *Transportation Science.* — 2014. — Vol. 48, no. 1. — Pp. 46–58. `http://www.jstor.org/stable/43666995`.

2. *Grey, Elizabeth.* What is focal length in photography. — `https://photographylife.com/what-is-focal-length-in-photography#focal-length-definition`. — Accessed: 2024-01-14.

3. *Huang, Shenyang(Andy).* Temporal Graph Learning in 2024 / Shenyang(Andy) Huang // *Towards Data Science.* — 2024. — Accessed: 2024-06-01. `https://towardsdatascience.com/temporal-graph-learning-in-2024-feaa9371b8e2`.

4. The Traveling Salesman Problem and Its Variations / Ed. by Gregory Gutin, Abraham P. Punnen. Combinatorial Optimization. — 1 edition. — Springer New York, NY, 2007. — Pp. XVIII, 830. — Published: 31 May 2002.

5. *Tinós, Renato.* Analysis of the dynamic traveling salesman problem with weight changes / Renato Tinós // 2015 Latin America Congress on Computational Intelligence (LA-CCI). — 2015. — Pp. 1–6. — doi:10.1109/LA-CCI.2015.7435936.

6. *Li, Changhe.* A New Approach to Solving Dynamic Traveling Salesman Problems / Changhe Li, Ming Yang, Lishan Kang // Simulated Evolution and Learning / Ed. by Tzai-Der Wang, Xiaodong Li, Shu-Heng Chen et al. — Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. — Pp. 236–243. — https://doi.org/10.1007/11903697_31.

7. Solving Dynamic Traveling Salesman Problems With Deep Reinforcement Learning / Zizhen Zhang, Hong Liu, MengChu Zhou, Jiahai Wang // *IEEE transaction on neural networks and learning systems.* — 2023. — Vol. 34, no. 4. — Pp. 2119–2132.

8. *Bigras, Louis-Philippe.* The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times / Louis-Philippe Bigras, Michel Gamache,

Gilles Savard // *Discrete Optimization*. — 2008. — Vol. 5, no. 4. — Pp. 685–699. https://www.sciencedirect.com/science/article/pii/S1572528608000339.

9. *Kool, Wouter*. Attention, Learn to Solve Routing Problems! — 2019.

10. *Isik, Senem*. Tackling the Traveling Salesman Problem with Graph Neural Networks / Senem Isik, Michael Atkin // *Stanford CS224W GraphML Tutorials*. — 2023. https://medium.com/stanford-cs224w/tackling-the-traveling-salesman-problem-with-graph-neural-networks-

11. A Gentle Introduction to Graph Neural Networks / Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, Alexander B. Wiltschko // *Distill*. — 2021. — https://distill.pub/2021/gnn-intro.

12. *Montero, Agustín*. An integer programming approach for the time-dependent traveling salesman problem with time windows / Agustín Montero, Isabel Méndez-Díaz, Juan José Miranda-Bront // *Computers Operations Research*. — 2017. — Vol. 88. — Pp. 280–289. https://www.sciencedirect.com/science/article/pii/S0305054817301612.

13. *SoleimanianGharehchopogh, Farhad*. New Approach for Solving Dynamic Traveling Salesman Problem with Hybrid Genetic Algorithms and Ant Colony Optimization / Farhad SoleimanianGharehchopogh, Isa Maleki, Masoum Farahmandian // *International Journal of Computer Applications*. — 2012. — 09. — Vol. 53. — Pp. 39–44.