HUMAN ROBOT COOPERATION IN VIRTUAL ENVIRONMENT

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Fulfillment of the Requirements for the Degree of

Master of Science

by

Suming Qiu

Aug 2023

ABSTRACT

The rapid advancements in robotics, driven by technologies such as artificial intelligence, computer vision, and machine learning, have led to numerous applications in daily life. Real-world experiments for evaluating robotic performance can be expensive and time-consuming, prompting a shift towards virtual simulations. This essay explores the role of virtual simulations in robotics, focusing on critical tasks such as path planning, obstacle avoidance, and target detection. We discuss the use of A* algorithm for path planning, segmentation maps and depth maps for obstacle avoidance, and You Only Look Once (YOLO) for target detection in drones. Additionally, we highlight the potential of digital twin technology, which connects a physical drone with its virtual counterpart, allowing researchers to optimize performance and evaluate behavior in various environments. The increasing popularity of virtual simulations in robotics research offers a cost-effective and efficient method for optimizing robot performance and paves the way for new applications that can enhance our lives.

BIOGRAPHICAL SKETCH

Suming Qiu earned his Bachelor of Science degree in Theoretical and Applied Mechanics from Lanzhou University and is currently pursuing a Master of Science degree in Mechanical and Aerospace Engineering, with a minor in Computer Science, at Cornell University. During his undergraduate studies, Suming participated in seven research programs and published three papers. His research interests span friction, finite element analysis (FEA), mechanical designs, computer vision, and Unreal Engine applications.

Suming is deeply passionate about interdisciplinary pursuits that integrate physics, computer science, and mathematics. His projects often demonstrate his ability to merge diverse knowledge domains effectively. In the future, Suming aspires to delve further into interdisciplinary research, exploring novel and intriguing intersections of these fields.

This document is dedicated to all Cornell graduate students.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In recent years, the field of robotics has experienced remarkable advancements, driven by the integration of cutting-edge technologies such as artificial intelligence (AI), computer vision, and machine learning[8, 17]. These technologies have revolutionized various aspects of daily life, as demonstrated by the widespread use of Roomba vacuum cleaners [12] and the development of autonomous vehicles by companies like Waymo. As the technology continues to progress, there is a growing demand for real-world experiments to assess performance [18]. However, such experiments can be time-consuming and expensive, prompting researchers to explore virtual simulations as a cost-effective alternative [28].

Virtual simulations have become increasingly popular, with prevalent tools such as Unreal Engine and the Robot Operating System (ROS) [39]. These tools allow researchers to set parameters and simulate environments, enabling them to evaluate and optimize robot performance [18]. The use of virtual simulations has been shown to be particularly effective in addressing critical challenges in robotics, such as ensuring optimal performance in the presence of sensor noise and environmental noise [10].

Path planning is another essential task in robotics, involving determining an optimal path for a robot to move from one point to another [31]. The A* algorithm, a popular heuristic search algorithm, has been widely employed for path planning in robotics, efficiently finding the shortest path while avoiding obstacles [21]. Recent studies have also explored other methods for path planning, such as Rapidly-Exploring Random Trees (RRT) [30] and Probabilis-

tic Roadmaps (PRM) [26], demonstrating the importance of this research area.

Obstacle avoidance is a critical challenge in robotics, with researchers using segmentation maps and depth maps to generate obstacle maps that can be used for real-time obstacle avoidance [38]. Techniques such as Mono-depth [15] and Detectron2 [47] have been commonly used to generate these maps, providing a comprehensive representation of the environment.

Target detection is another critical task in robotics, particularly for drones [40]. Researchers have utilized You Only Look Once (YOLO) to detect targets and track them using drones [5, 40], enabling applications such as surveillance, agriculture, and search and rescue operations.

Digital twin technology has emerged as a promising research avenue in robotics, connecting a physical drone to its virtual counterpart using optical track equipment for virtual testing and simulation [37]. This technology allows researchers to optimize performance and evaluate robot behavior in various environments, paving the way for the development of more efficient and effective robots.

In conclusion, the literature highlights the increasing popularity of virtual simulations in robotics as a cost-effective and efficient means of evaluating robot performance. Technologies such as the A* algorithm, segmentation maps, depth maps, YOLO, and digital twin technology exemplify the significant advancements in robotics research. By optimizing robot performance, enhancing capabilities, and developing new applications, these technologies have the potential to greatly impact our lives.

CHAPTER 2

PROBLEM FORMULATION AND ASSUMPTIONS

The problem of human-robot interaction is quite complex. In this article, we divide the problem into three parts: environment reconstruction, implementation of machine learning algorithms, and model-based target detection.

Environment reconstruction: Many simulation environments do not have adequate support from a physical engine. We choose to use Unreal Engine (UE) to implement our simulation environment. We first test the algorithms in environments like MATLAB and Python, then port them to UE for simulation testing, and finally validate our work in a lab test. In UE, we construct both a city environment and a test environment that matches our laboratory setting to verify the accuracy of our work. If the virtual reality laboratory and the real laboratory scenarios are similar, then the results obtained in the virtual city will be more credible.

Implementation of machine learning algorithms: We essentially divide the algorithm into two parts: path planning and AI-based detection. With these two algorithms, the robot can plan the path when given the start and end points. It can also detect surrounding areas and objects using the AI-based algorithm. For path planning, we mainly test A* and Rapidly-Exploring Random Trees (RRT), which are two traditional path planning methods. RRT is faster, while A* yields better results.

Model-based target detection: The focus of this article is human-robot interaction. we address the problem of tracking multiple moving targets through human-robot collaboration. This problem is highly relevant to various security

and surveillance applications where Unmanned Ground Vehicles (UGVs) and Unmanned Aerial Vehicles (UAVs) team up with human operators to actively observe a set of targets within a large region of interest.

Consider a team of mobile robots, denoted by $\mathcal{R} = \mathcal{R}1, \ldots, \mathcal{R}N$, operating within a closed and bounded two-dimensional (2D) workspace $\mathcal{W} = [0, L_x] \times [0, L_y]$. Here, $L_x, L_y \in \mathbb{R}^+$ represent the length and width of the workspace, and $\mathcal{N} = 1, \ldots, N$ is the index set of the mobile robots. We define an inertial frame, $\mathcal{FW}$, with origin $\mathcal{OW}$, embedded in $\mathcal{W}$, such that the xy-plane aligns with the ground plane.

Furthermore, a moving Cartesian frame, $\mathcal{F}_{\mathcal{A}n}$, is embedded in each robot, with the origin $\mathcal{OA}_n$ located at the principal point of the robot's camera. This frame aligns with the robot camera frame, as described in [20]. Without loss of generality, the state vector of the n^{th} robot, $\mathcal{R}_n$, is represented by $\mathbf{s}_n = [\mathbf{p}_n^T \quad \theta_n]^T$, where $\mathbf{p}n = [x_n \quad y_n]^T$ and θ_n denote the 2D coordinates and orientation with respect to $\mathcal{FW}$, as illustrated in Fig. 2.1 [36].

The state vector $\mathbf{s}_n$ complies with the robot dynamics equation, which in this study is given by the unicycle model, as detailed in [36, 11]. This model effectively captures the essential dynamics of the mobile robots and provides a suitable framework for the analysis and implementation of the proposed human-robot collaborative tracking approach.

$$\dot{\mathbf{s}}_n = \begin{bmatrix} \dot{x}_n \\ \dot{y}_n \\ \dot{\theta}_n \end{bmatrix} = \begin{bmatrix} v_n \cos\theta_n \\ v_n \sin\theta_n \\ \omega_n \end{bmatrix} = \mathbf{f}(\mathbf{s}_n, \mathbf{u}_n), \quad \forall n \in \mathcal{N} \tag{2.1}$$

where $\mathbf{u}_n = [v_n \quad w_n]^T \in \mathbb{R}^2$ is the control vector for the robot. It consists of linear velocity v_n and angular velocity w_n. Assuming Δt is the sampling interval, we can write the robot state and control vector at any discrete time k as $\mathbf{s}_n(k)$ and $\mathbf{u}_n(k)$ respectively.

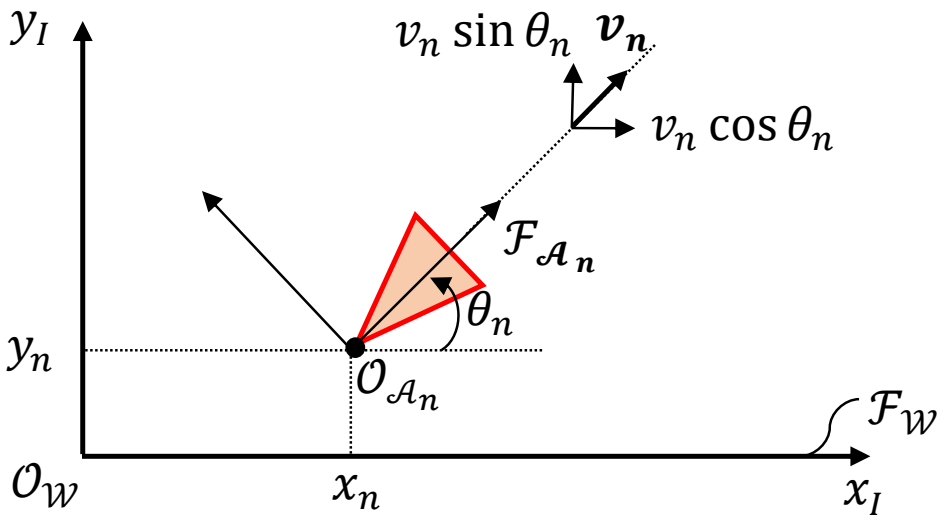

Figure 2.1: Definition of the state of a robot.

Assume a Multi-Human Robot Team (MHRT) tasked with tracking a set of human targets denoted by $\mathcal{T} = \mathcal{T}_1, \ldots, \mathcal{T}_M$, where $\mathcal{M} = 1, \ldots, M$ represents the target index set. We also assume that the number of targets is no less than the size of the robot team, i.e., $M \geq N$. The i^{th} target's state is represented by $\mathbf{x}i = [xi \quad yi \quad vx,i \quad vy,i]^T \in \mathbb{R}^{4\times1}, i \in \mathcal{M}$. We can also write all targets' state as $\mathbf{x} = [\mathbf{x}1^T, \ldots, \mathbf{x}_M^T]^T \in \mathbb{R}^{4M\times1}$. We assume that the target velocity remains constant and is subject to process noise $\mathbf{w}(k)$, which is additive, Gaussian, and white. Combined all these together, we can then have the target prediction model:

$$\mathbf{x}_i(k) = \mathbf{F}\mathbf{x}_i(k-1) + \mathbf{w}(k), \quad \mathbf{w}(k) \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \tag{2.2}$$

In this equation, $\mathbf{F} \in \mathbb{R}^{4\times4}$ represents the state transition matrix for the constant-velocity model [3], and $\mathbf{Q}$ denotes the prediction covariance matrix. This model

effectively captures the essential dynamics of the moving targets and provides a suitable framework for the analysis and implementation of the proposed human-robot collaborative tracking approach.

Let $\mathcal{P} = \mathcal{P}_1, \ldots, \mathcal{P}N$ denote the target assignment to the robot team, such that $\mathcal{P}n \cap \mathcal{P}n' = \emptyset,\ n \neq n'$ and $\bigcup n = 1^N \mathcal{P}_n = \mathcal{P}$. To simplify, we assume that $\mathcal{P}$ is known a *priori* and remains constant throughout the mission. However, we will incorporate robot and target dynamics and adaptive assignment control strategies in the future. $\mathbf{z}_{n,i}(k)$ represents the measurements of the i^{th} target by the n^{th} robot. This can be achieved once the robot "sees" the target, denoted by $\mathcal{S}n$. In contrast to conventional sensors that measure the range and bearing to targets [13, 32], we derive a vision-based measurement model in Section 3.4.1, which relies on images streamed by the robot camera and directly measures target positions in the inertial frame $\mathcal{FW}$ as follows:

$$\mathbf{z}_{n,i}(k) = \begin{cases} H\mathbf{x}_i(k) + \mathbf{v}(k) & \text{if} \quad \mathbf{x}_i(k) \in \mathcal{S}_v(k) \\ \emptyset & \text{if} \quad \mathbf{x}_i(k) \notin \mathcal{S}_n(k) \end{cases} \tag{2.3}$$

where $\mathbf{H} = [\mathbf{I}_2 \quad \mathbf{0}_2]$ and $\mathbf{v}(k) \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$ is Gaussian noise.

Imagine a scenario in which robots collaborate with human operators in a tracking mission, aiming to maximize the cumulative tracking time for all targets. Cooperation within the mixed team is facilitated through a two-way message exchange mechanism to share mission-critical data. Both the robot message $\mathbf{q}_n, n \in \mathcal{N}$ and the human message $\mathbf{m}$, will be introduced in Section 3.4.2.

The tracking time of $\mathcal{T}_i, i \in \mathcal{P}_n$ up to time k is defined as:

$$t_i(k) = \sum_{\tau=1}^{k} \mathbf{1}(\mathbf{x}_i(\tau) \in \mathcal{S}_n(\tau)) \tag{2.4}$$

We can use a global utility function U_g to set the objective of the MHRT:

$$U_g = \sum_{i=1}^{M} t_i(T_f) \tag{2.5}$$

We aim to maximize the robot control ability to achieve optimal tracking performance by the task end time T_f.

Optimizing a global utility in a multi-robot team necessitates accumulating and processing information from all agents in the team [4, 2], which often results in high communication and computational loads. This paper introduces a distributed approach where the global objective is achieved by individual robots concurrently and efficiently optimizing a local utility function. The local utility function $U_{n,i}(\cdot)$ represents local measure of tracking performance which aligns with U_g [2] and is also defined as a function of the robot's state $\mathbf{s}n(k)$, the target measurement $\mathbf{z}n, i(k)$, and the human message $\mathbf{m}$ in Section 3.4.3. Furthermore, let $\mathcal{B}$ represent an obstacle map of the workspace, and let $C(\mathcal{B})$ be the penalty function for obstacle collision. The distributed cooperative tracking problem will be discussed in details in 3.4.

CHAPTER 3

AI ENABLED PATH PLANNING FOR AUTONOMOUS VEHICLE

3.1 Obstacles Reconstructions

The reconstruction of obstacle maps is an essential process for creating optimal paths for autonomous vehicles, robots, and drones. With the help of modern software and tools, it has become easier to create accurate 3D models of complex structures like big cities. In this essay, we will discuss how to reconstruct the obstacle map using the example of the big city model from Unreal Engine 5.

Firstly, we need to select a street or area in the 3D city model and extract its geometry. For this purpose, we can use software like Solidworks, which can import 3ds files and export them to a more compatible format such as STEP. This step is crucial as it allows us to work with the geometry of the selected area more efficiently.

Next, we need to convert the STEP file into STL format. STL (STereoLithography) is a widely used format for 3D printing and is also compatible with most of the modern CAD software. To convert the STEP file into STL, we can use a MATLAB function called 2STL, which converts the 3D model into an STL file. This file can then be used to create an accurate 3D representation of the area we selected.

After generating the 3D model, we need to simplify it to create a more manageable layout for the obstacle map. This step is essential as the complex 3D model can cause computational problems, making it challenging to create an accurate obstacle map. By simplifying the 3D model, we can generate a 2D ob-

stacle map, which represents a bird's eye view of the area. This 2D obstacle map can be used to create optimal paths from one point to another within the area.

Finally, we can use the 2D obstacle map to create optimal paths. By using algorithms such as A* and Dijkstra's algorithm, we can find the shortest path between two points in the area. These algorithms use the 2D obstacle map to determine the optimal path while avoiding any obstacles in the area.

Reconstructing an obstacle map for a complex structure like a big city requires multiple steps, including selecting the area, converting it to a more compatible format, simplifying it, and generating a 2D obstacle map. The process described above can be used as a general guideline for reconstructing obstacle maps for any complex structure.

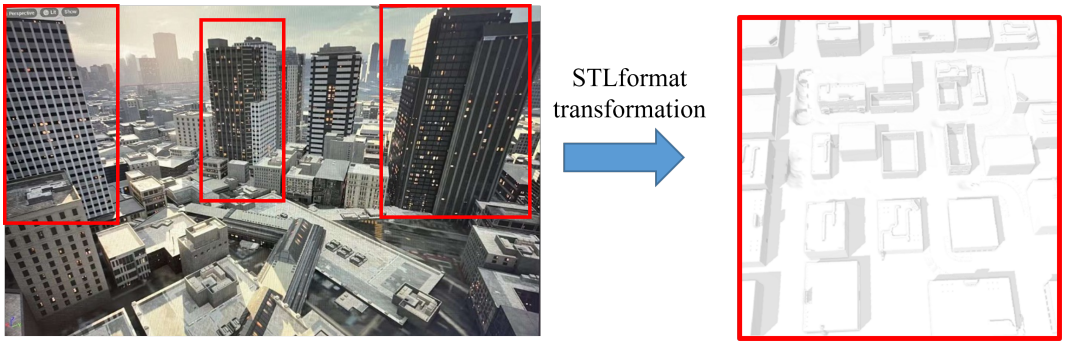

Figure 3.1: Reconstruction of the virtual environment.

3.2 Path Planning for Drones

A* and RRT are used for path planning in our work. We choose A* as it produces a more optimal path than RRT.

3.2.1 RRT Algorithm

The RRT algorithm works by randomly generating points in the search space and connecting them with a tree structure to build a path towards the goal state.

The basic idea behind RRT is to start with a single root node and randomly generate new nodes in the search space. The algorithm then connects the newly generated node to its nearest neighbor in the tree and repeats this process until a node is generated that is close enough to the goal state. The tree structure is incrementally built over time as the algorithm explores the search space, and once the goal node is reached, the algorithm can be used to backtrack from the goal node to find the path from the start node to the goal node [31].

One of the key advantages of the RRT algorithm is its ability to handle high-dimensional and complex environments. Unlike other pathfinding algorithms, such as A* or Dijkstra's algorithm, the RRT algorithm can be used to find paths in environments with many obstacles and with a large number of dimensions. Additionally, the RRT algorithm is probabilistically complete, meaning that it is guaranteed to find a solution if one exists, provided that enough time is given [25]. Another advantage of RRT is its simplicity and ease of implementation. Unlike other pathfinding algorithms, the RRT algorithm does not require a grid-based representation of the search space, which can be computationally expensive for large and complex environments. Instead, the algorithm can be implemented using a simple tree structure, making it easy to implement and modify for different applications. However, the RRT algorithm also has some limitations. For example, the quality of the path found by the RRT algorithm may not be optimal, as the algorithm is based on a random sampling process and does not take into account the distance between nodes or the cost of moving from

one node to another [31]. Additionally, the algorithm may not find the shortest path between the start and goal states if there are many obstacles in the search space.

3.2.2 A* Algorithm

A* is a popular pathfinding algorithm used in computer science and robotics to find the shortest path between two points on a graph or grid. The algorithm combines elements of both Dijkstra's algorithm and greedy best-first search to efficiently search through the space of possible routes and find the optimal solution [21].

The A* algorithm works by maintaining two lists of nodes: the open list and the closed list. The open list contains nodes that have been discovered but have not yet been explored, while the closed list contains nodes that have already been explored. The algorithm then evaluates each node on the open list and selects the node with the lowest total cost, which is a combination of the cost to reach the node from the starting point and an estimate of the cost to reach the end point [7].

The estimate of the cost to reach the end point is calculated using a heuristic function, which provides an optimistic estimate of the remaining distance based on the characteristics of the graph or grid. For example, in a grid-based maze, the heuristic function might be the Manhattan distance between the current node and the end point[42], while in a map-based navigation problem, the heuristic function might use the distance between two points as the crow flies[43] .

The A* algorithm is both complete and optimal, meaning that it is guaranteed to find a solution if one exists and to find the shortest path between the starting point and the end point. It is also relatively efficient[29], with a time complexity of O(b^d), where b is the branching factor of the graph and d is the depth of the solution[45].

One of the key advantages of the A* algorithm is its versatility. It can be used to solve a wide range of pathfinding problems, including maze navigation, robotic motion planning, and game AI. In addition, the algorithm can be easily customized by adjusting the heuristic function to fit the specific problem domain.

However, the A* algorithm does have some limitations. It can be sensitive to the quality of the heuristic function, which can affect the efficiency and accuracy of the search. In addition, the algorithm can struggle with problems that have a high degree of branching or that involve dynamic obstacles or changing environments.

A* algorithm is a powerful and flexible tool for solving pathfinding problems in a variety of domains. Its ability to find optimal solutions efficiently makes it a popular choice for robotics, game development, and other applications where efficient pathfinding is critical. While it does have some limitations, the A* algorithm remains one of the most widely used and effective pathfinding algorithms available today.

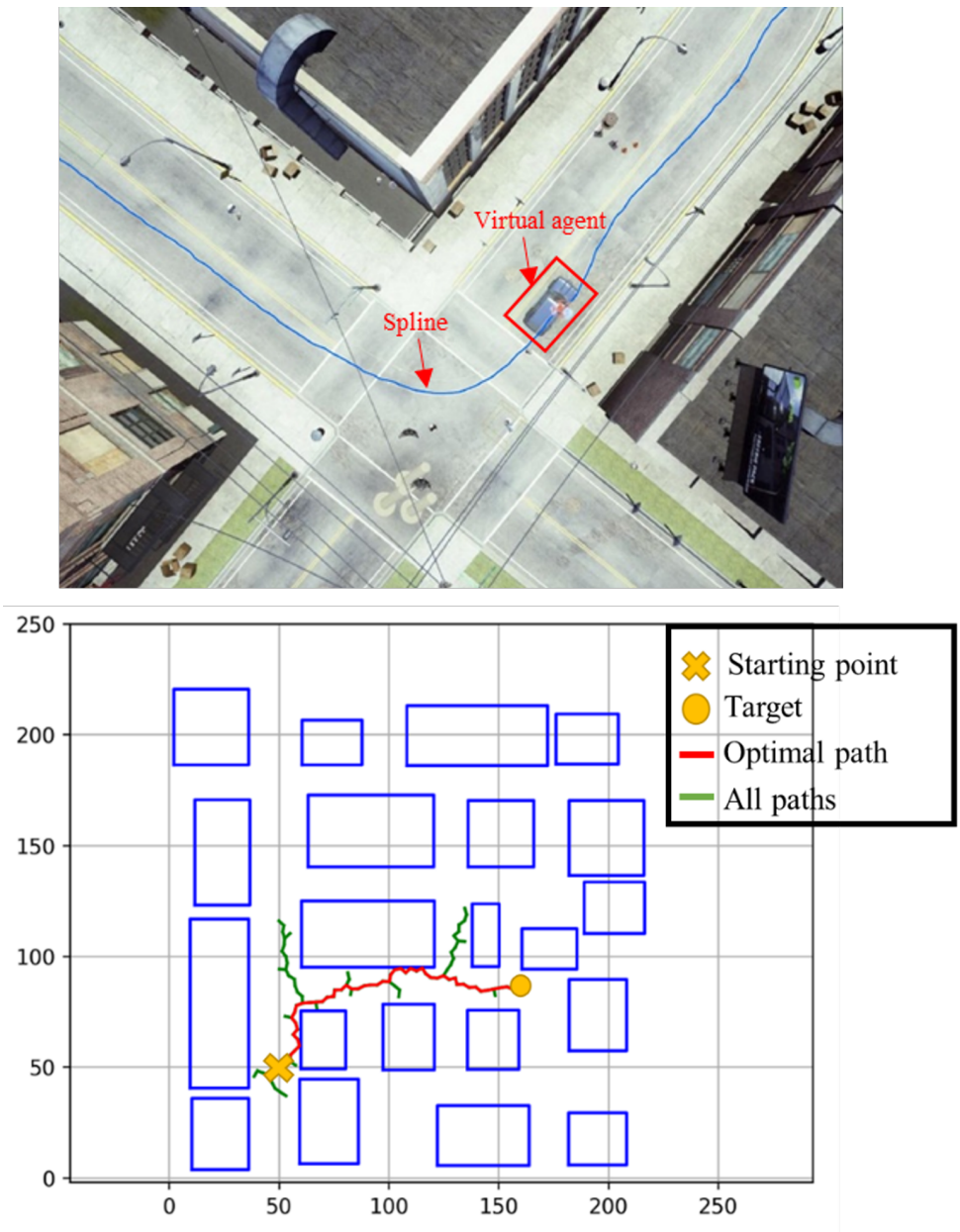

Figure 3.2: Illustration of the tracking

3.2.3 Implementation on drones

Due to the complexity of RRT and A* algorithms in three-dimensional space, we assume that the unmanned aerial vehicle (UAV) does not encounter any obstacles in the z-axis direction. Thus, we can simplify the obstacle map into a

two-dimensional form. This simplification makes it easier to apply pathfinding algorithms and generates feasible routes for UAVs.

To implement this algorithm, we place the two-dimensional obstacle map in Matlab and establish communication between Matlab and the UAV. Matlab generates the route using the pathfinding algorithm and sends it to the UAV control function. The UAV then follows the specified path until it reaches the destination.

However, as the environment is dynamic, there may be moving obstacles such as pedestrians and vehicles that cannot be anticipated beforehand. To overcome this problem, we implement YOLO and Detectron2 algorithms on the UAV to avoid real-time obstacles. The depth map algorithm provides information on the distance to the nearest obstacle, while the segmentation algorithm distinguishes between different objects in the environment. By using these algorithms, the UAV can detect and avoid obstacles in real-time.

In conclusion, by simplifying the obstacle map into two dimensions, we can apply pathfinding algorithms to generate feasible routes for UAVs. The integration of Matlab and UAV control functions enables the UAV to follow the specified route accurately. The addition of depth map and segmentation algorithms enables the UAV to detect and avoid real-time obstacles, making the algorithm applicable in dynamic environments.

3.3 Deep-learning based algorithms' implementations

A depth map is a 2D representation of the distance between an object and a camera or sensor. This information can be used to detect the presence of obstacles and estimate their distance from the robot.

A segmentation map is a type of image processing technique used to separate an image into multiple segments or regions, each representing a distinct object or part of the image. The goal of segmentation is to divide the image into meaningful parts, based on certain features, such as color, texture, shape, or depth, and to identify the objects or regions in the image.

By combining the depth map with a segmentation map, which segments the image into different objects or regions, the robot can identify and classify different types of obstacles. To perform obstacle avoidance, the robot would use its depth map and segmentation map to identify obstacles in its path and determine the best course of action to avoid them. For example, if the robot encounters a large, stationary obstacle, it can use its depth map to determine the distance to the obstacle and adjust its path to avoid it. On the other hand, if the robot encounters a moving obstacle, such as a person or another vehicle, it can use its segmentation map to identify the object as a moving obstacle and adjust its path accordingly.

To improve the accuracy and reliability of the obstacle avoidance system, the depth map and segmentation map can be combined with additional sensors, such as lidar or radar, to provide a more comprehensive view of the environment. The robot can also use machine learning algorithms, such as neural networks or support vector machines, to analyze the data from the depth map

and segmentation map and make more informed decisions about how to avoid obstacles.

3.3.1 Detectron2 and Mask R-CNN

Detectron2 is an open-source software library that provides a wide range of computer vision algorithms for object detection, instance segmentation, and other related tasks. It is built on top of the PyTorch deep learning framework and provides a modular and flexible architecture that allows users to easily customize and extend the existing models.

One of the key features of Detectron2 is its support for state-of-the-art object detection models, such as Faster R-CNN and RetinaNet. These models use a combination of region proposals and deep convolutional neural networks to detect and classify objects in an image[41, 34]. The models are trained on large datasets such as COCO, Pascal VOC, and ImageNet, which provide a rich variety of annotated images for training and testing[35, 9, 8].

The detection model in Detectron2 is based on the following formula:

$$\text{Detection Score} = \text{Classification Score} \times \text{Regression Score} \tag{3.1}$$

where the classification score is the probability of the object belonging to a certain class, and the regression score is the predicted bounding box coordinates of the object. The detection score is used to rank the object proposals and select the top-scoring proposals as the final detections.

Instance segmentation is another important task in computer vision, and

Detectron2 provides a wide range of models and algorithms for this task as well. One of the popular models is Mask R-CNN[23], which extends Faster R-CNN by adding a segmentation branch to the network. The segmentation branch generates a binary mask for each object proposal, which is then used to refine the object boundaries and improve the accuracy of the detection.

The instance segmentation model in Detectron2 is based on the following formula:

$$\text{Mask Score} = \text{Classification Score} \times \text{Mask Probability} \tag{3.2}$$

where the mask probability is the probability of each pixel belonging to the foreground or background. The mask score is used to refine the object boundaries and generate more accurate segmentation results.

3.3.2 Mono-depth

Monodepth is a popular algorithm used for depth estimation from single images. It is based on a deep neural network that is trained to predict the depth map of a given image[16].

The core idea of Monodepth is to use a convolutional neural network (CNN) to estimate the depth of each pixel in the input image. This is achieved by training the network on a large dataset of stereo images, where the depth information is available for both the left and right camera views[48].

The Monodepth algorithm is based on the following formula:

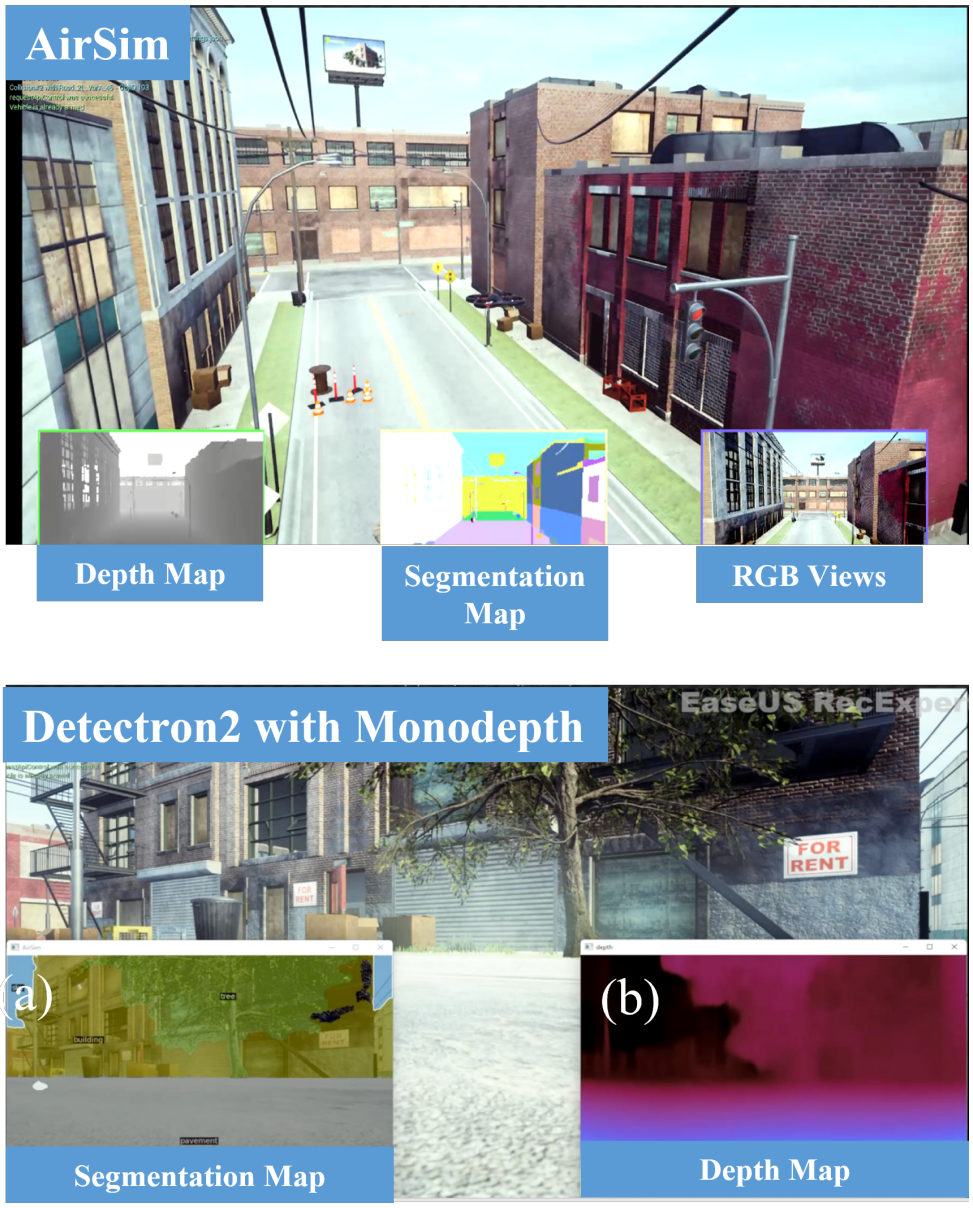

Figure 3.3: Implement the deep learning algorithm on drones.

$$D = f \cdot \frac{B}{z} \tag{3.3}$$

where D is the depth of the pixel, f is the focal length of the camera, B is the baseline distance between the left and right cameras, and z is the distance of the pixel from the camera.

The depth map is estimated by training the CNN to predict the inverse depth of each pixel, which is proportional to the distance of the pixel from the camera. The focal length and baseline distance are known values that can be set based on the camera calibration, and are used to convert the predicted inverse depth back into the actual depth of each pixel[16].

One of the key advantages of Monodepth is its ability to estimate depth from a single image, without requiring a stereo camera setup. This makes it suitable for a wide range of applications, such as robotics, autonomous vehicles, and virtual reality[6].

In addition to depth estimation, Monodepth can also be used for 3D reconstruction, by combining the depth maps from multiple images to generate a 3D point cloud. This can be done using techniques such as Structure from Motion (SfM) and Multi-View Stereo (MVS)[44].

3.3.3 YOLO

YOLO, short for You Only Look Once, is a popular object detection algorithm that is widely used in computer vision applications. YOLO is known for its high accuracy and speed, making it a popular choice for real-time object detection tasks[40].

The core idea of YOLO is to divide the input image into a grid of cells and predict the object class and location for each cell. This is achieved by using a deep convolutional neural network (CNN) that is trained on a large dataset of annotated images.

The YOLO model consists of two main parts: a feature extractor and a detection head. The feature extractor is typically a pre-trained CNN such as ResNet [24] or Darknet [40], which is used to extract a set of high-level features from the input image. The detection head is a set of convolutional layers that predict the object class and location for each cell in the grid.

The YOLO algorithm is based on the following formula:

$$\text{Detection Score} = \text{Objectness Score} \times \text{Classification Score} \times \text{Localization Score} \tag{3.4}$$

where the objectness score is a measure of how likely the cell contains an object, the classification score is the probability of the object belonging to a certain class, and the localization score is the predicted bounding box coordinates of the object.

The objectness score is computed based on the intersection over union (IoU) between the predicted bounding box and the ground truth bounding box. The classification score is computed using a softmax function over the predicted class scores for each cell. The localization score is computed using a regression function that predicts the offset between the center of the cell and the center of the object.

One of the key advantages of YOLO is its speed. Since the detection is performed on the entire image at once, YOLO can process images in real-time, making it suitable for applications such as autonomous driving[14], robotics[19], and surveillance. Additionally, YOLO is able to detect objects at different scales and aspect ratios, making it robust to variations in object size and orientation.

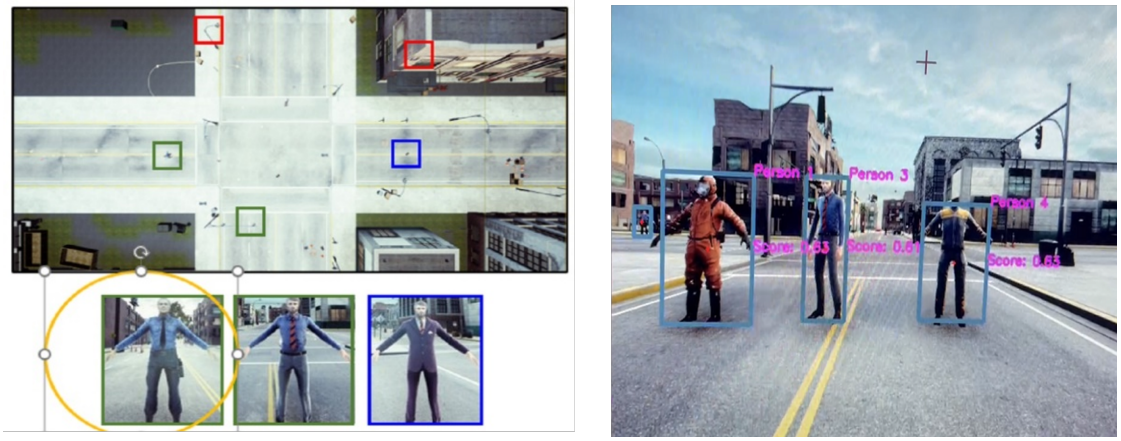

Figure 3.4: YOLO's implementation on drones.

3.3.4 Path Re-planing with Imperfect Map Knowledge

Utilizing the information provided, we aim to design a virtual robot that is capable of autonomously planning a path to its destination, while simultaneously performing real-time target detection and obstacle avoidance. To evaluate the drone's ability to tackle invisible obstacles, we conduct two separate experiments within the city environment designed earlier, which includes two 360-degree cameras. The drone is tasked with planning a path that avoids detection by any of the cameras within the city.

As depicted in the accompanying figure, the red dot represents the initial position, while the blue dot signifies the destination. The red region illustrates the detection range of the cameras, and the green region demonstrates the drone's sensing area. Initially, the drone plans the blue path; however, upon detecting an unanticipated obstacle in the scene that was not included in the obstacle map, the drone updates its path. This new path takes advantage of the obstacle's position within the camera's view, providing the drone with a more discreet route.

In the second experiment, we introduce imperfections to the obstacle map by

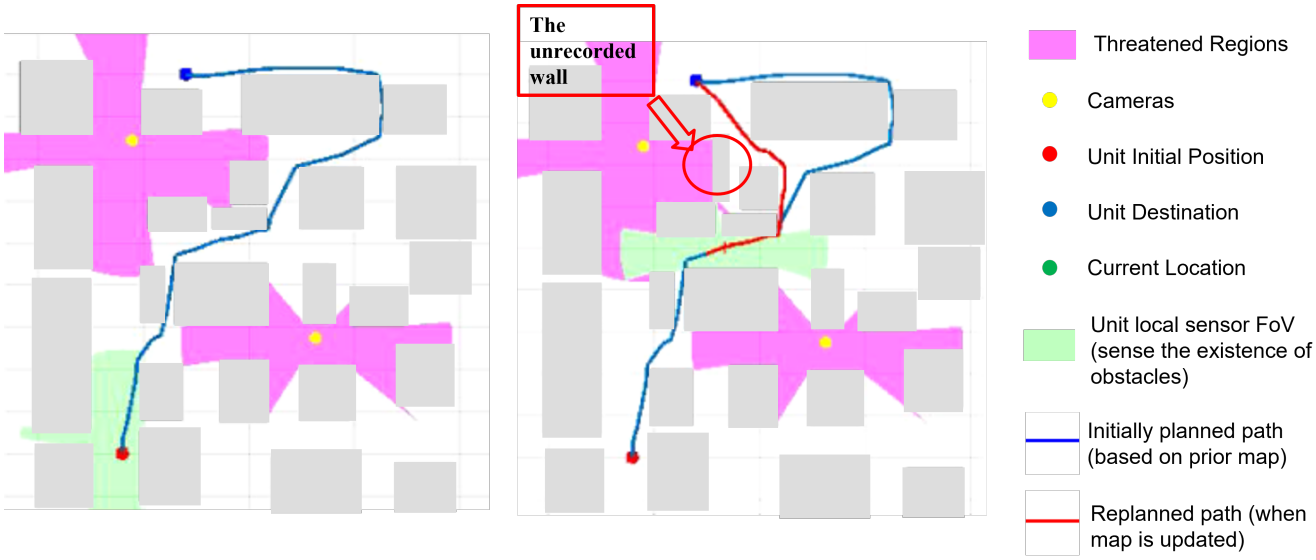

Figure 3.5: Find imperfect walls in the scene.

removing a wall without recording the change in the strategy. Consequently, the drone must autonomously locate the missing wall and replan its path to avoid being detected by the cameras. As shown in the figure, the drone successfully identifies the discrepancy in the map and adjusts its path accordingly to achieve its goal. The second figure reveals that the drone manages to find an improved path without significantly increasing the time needed to reach its destination, effectively avoiding detection. These experiments demonstrate the virtual robot's ability to adapt and navigate within an environment containing unexpected obstacles and imperfections in the obstacle map. By detecting and replanning its path in real-time, the drone can evade detection by the cameras while efficiently reaching its destination.

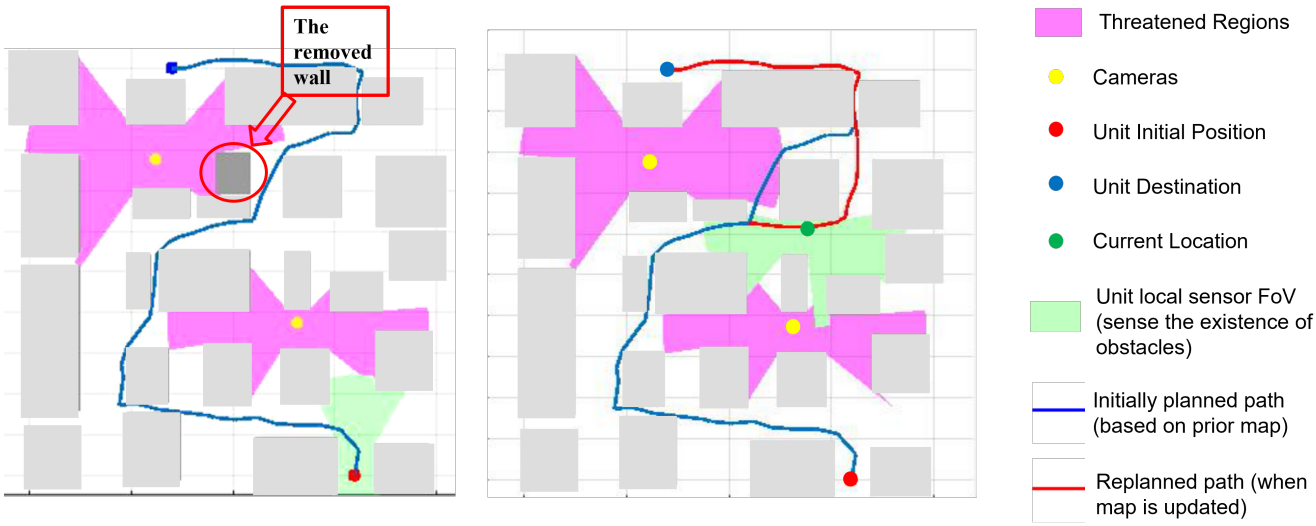

Figure 3.6: Find imperfect walls in the scene.

3.4 Model Based target Detection and Tracking

This section proposes a novel approach for multi-target tracking that achieves human-robot cooperation, online sensing, and distributed control optimization in a single framework, as shown in Fig.3.7. Because the robot states $\mathbf{s}_n$ can be easily estimated using onboard or external motion sensors, this section focuses on introducing the vision-based target state estimation, the cooperation mechanism in the MHRT, and the tracking utility function for distributed optimization.

3.4.1 Online Target State Estimation

Previous studies on vision-based target estimation often rely on a monocular camera to gather measurements in the image frame or virtual image plane [46, 33], resulting in intricate non-linear sensor measurement models. Alternatively,

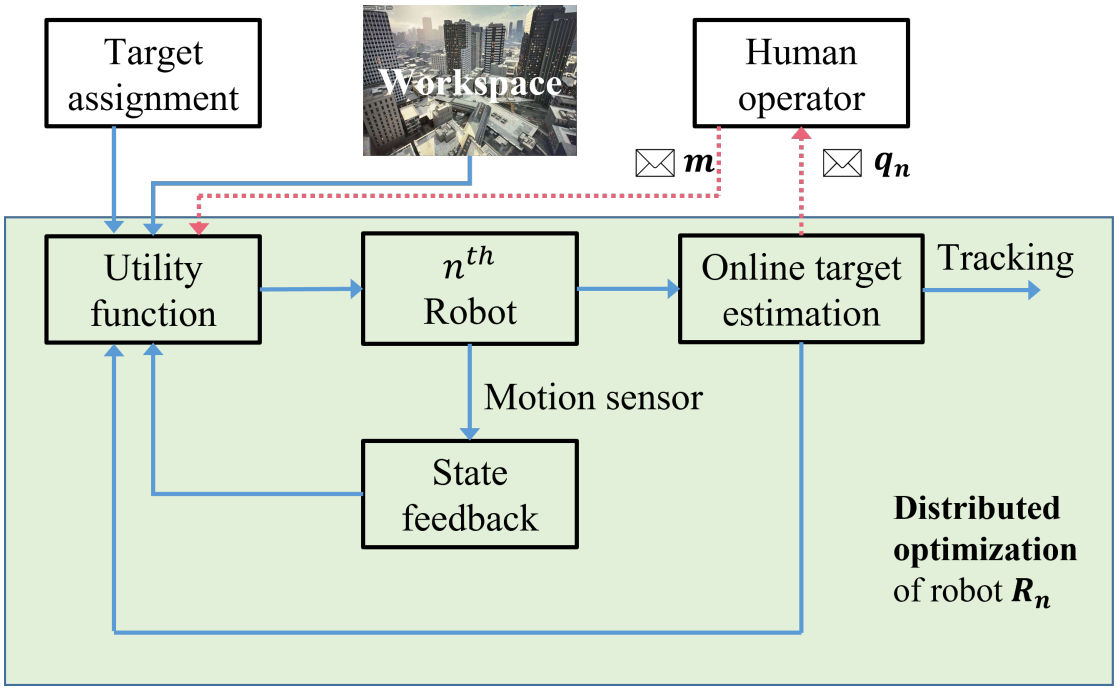

Figure 3.7: Cooperative tracking framework.

this paper fuses Convolutional Neural Network (CNN)-based target detection with the ray-tracing technique (Fig.3.8) to estimate the target position in the inertial frame $\mathcal{F}_W$ directly, utilizing RGB-D images captured by the onboard robot camera, as illustrated in Fig.3.9.

The discrete-time index k is omitted in this subsection for the sake of simplicity. Let F_I denote the image reference frame, and let $x_{i|\text{image}}$ represent the 2D position of target $\mathcal{T}_i$ relative to $\mathcal{F}_I$. This 2D position can be approximated using the image coordinate located at the center of the target's bounding box, derived from detection algorithms like MASK-RCNN [22]. With $\mathbf{x}_{i|\text{image}}$ at hand, the target depth, d_i, can be determined by the corresponding pixel value in the depth image. Following this, the target position with respect to the camera frame F_{A_n}

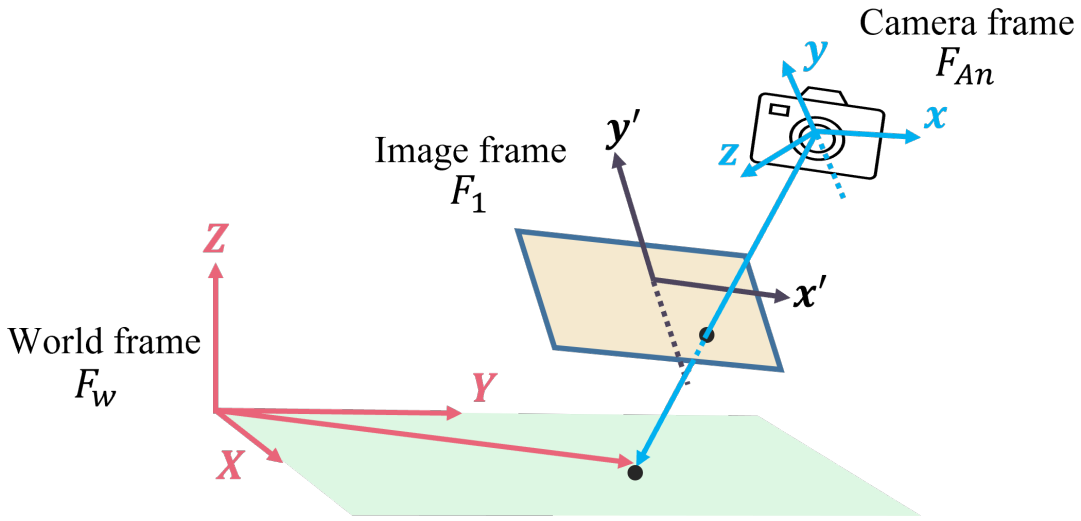

Figure 3.8: Illustration of the ray tracing method.

is expressed as:

$$\mathbf{x}_i|_{\text{camera}} = d_i \mathbf{K}^{-1} [\mathbf{x}_i|_{\text{image}} \quad 1]^T \tag{3.5}$$

where $\mathbf{K} \in \mathbb{R}^{3\times3}$ represents the camera intrinsic matrix. The target measurement $\mathbf{z}_{n,i}$ in the inertial frame $\mathcal{F}_\mathcal{W}$ is acquired by transforming $\mathbf{x}_i|_{\text{camera}}$ from $\mathcal{F}_{\mathcal{A}_n}$ to $\mathcal{F}_\mathcal{W}$:

$$\mathbf{z}_{n,i} = \mathbf{R}_n \mathbf{x}_i|_{\text{camera}}^T + \mathbf{r}_n^T \tag{3.6}$$

where $\mathbf{R}_n$ and $\mathbf{r}_n$ are camera extrinsic parameters that can be estimated from the robot state vector as follows:

$$\mathbf{R}_n = \begin{bmatrix} cos\theta_n & -sin\theta_n & 0 \\ sin\theta_n & cos\theta_n & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{r}_n = [x_n \quad y_n \quad 0]^T \tag{3.7}$$

In comparison to the true target state $\mathbf{x}_i$, the measurement $\mathbf{z}_{n,i}$ does not in-

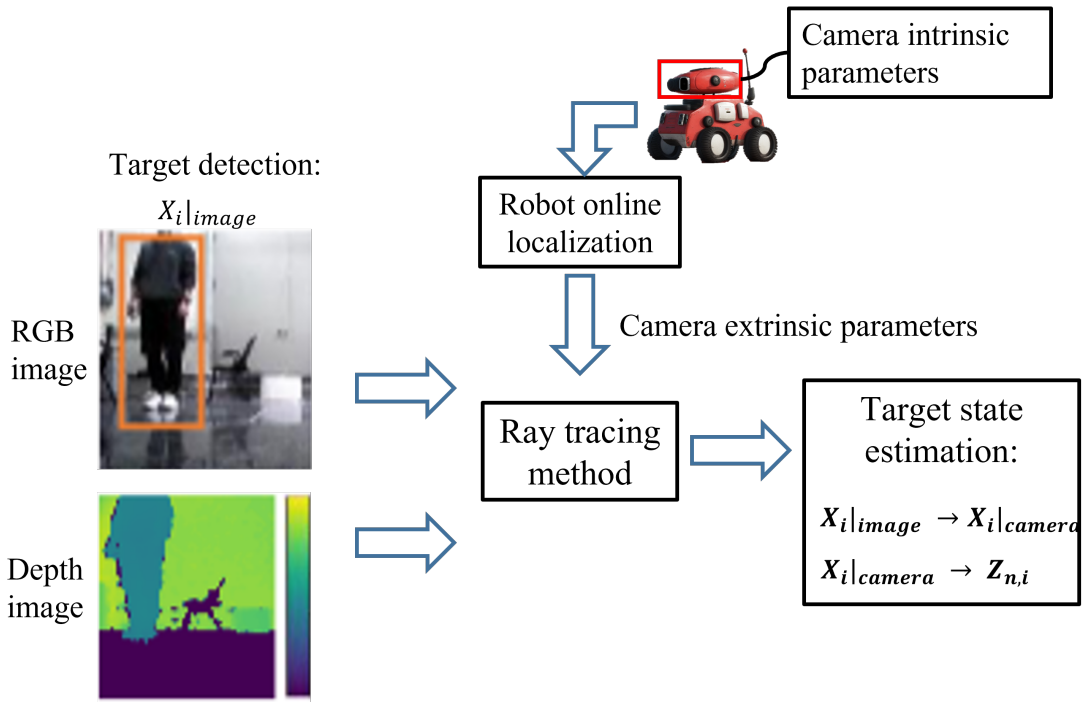

Figure 3.9: Vision-based online target estimation.

clude velocity terms and is assumed to be subject to white, additive Gaussian noise $\mathbf{v}$, which yields:

$$\mathbf{z}_{n,i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \mathbf{x}_i + \mathbf{v} \tag{3.8}$$

3.4.2 Human Robot Cooperation Strategy

Integrating operators into the game setting requires several steps, including implementing fixed cameras in the environment, defining the world coordinate system, designing the operator interface, and updating the world coordinates to the algorithm. These steps work together to enable players to interact with the game environment in a meaningful way.

The first step in integrating operators into the game setting is to implement several fixed cameras in the environment. These cameras capture images from different perspectives, providing players with different views of the game world. By using multiple cameras, players can gain a better understanding of the game environment and make more informed decisions about how to interact with it.

The next step is to define the world coordinate system. This involves using the pixel coordinates from the camera images and the corresponding real-world coordinates in the virtual environment to create a transfer function from 2D pictures to 3D world coordinates. This allows the game to map player actions in the 2D camera images to the 3D game world, enabling players to interact with the game environment more intuitively.

Once the world coordinate system is defined, the next step is to design an interface for human operators. This interface allows players to click on objects in the camera images, specifying their position in the game world. The interface then analyzes these clicks and translates them into world coordinates, which are passed to the game algorithm.

Finally, the world coordinates are updated to the algorithm, and the drone will update its path and go to the target position specified by the player. Even in foggy conditions where the drone may lose track of targets, the game algorithm can use the world coordinates to keep track of the target and ensure that the drone reaches its intended destination.

In conclusion, integrating operators into the game setting requires several steps that work together to create a more immersive and interactive gaming

experience. By implementing fixed cameras, defining the world coordinate system, designing the operator interface, and updating the world coordinates to the algorithm, players can interact with the game environment more intuitively and make more informed decisions about how to proceed.

Coordinate transform

The coordinate transform of here is same as illustrated in 2.4.1. We use Ray Tracing methods to change the image frame coordinate to world frame coordinate.

Ray tracing is a rendering technique that is widely used in computer graphics to create realistic images by simulating the behavior of light as it interacts with objects in a scene. The method involves tracing the path of light rays as they reflect and refract off surfaces in a three-dimensional space, in order to determine which surfaces should be visible in a given image.

To perform ray tracing, a virtual camera is placed in the 3D space (Figure 2.2), and rays of light are traced from the camera through each pixel in the image plane. These rays are then traced through the virtual environment, and at each point where the ray intersects with an object or surface, various properties of the material such as reflectivity, transparency, and color are calculated using complex mathematical algorithms.

The method is computationally intensive, as it requires tracing a large number of rays for each pixel in the image. However, the resulting images can be highly realistic, as ray tracing accurately models the way light behaves in the real world, including the effects of reflections, refractions, shadows, and other lighting phenomena.

In the context of autonomous driving, ray tracing can be used to detect and track objects in the environment, as well as to plan and optimize the vehicle's path based on the properties of the objects in the scene.

We use the online figures captured by the fixed camera inside the virtual environment for supervision window. Then we can click on the target which will catch the position of our click point. Using the ray tracing algorithm, we get the world coordinate.

Operator's assistance in extreme environment

In order to enable effective cooperation between a human operator and robots in the Multiple Human-Robot Teams (MHRT) system, it is necessary to establish a two-way communication channel that enables message exchanging through a shared network [1]. The communication network is assumed to have negligible delays to ensure smooth and real-time interaction between the human operator and the robots.

In the MHRT system, the tracking of targets is subject to uncertainties in the target dynamics. As the time gap between the tracking intervals increases, the accuracy of the robot's prediction of the target's position decreases, which lowers the probability of successful recovery of the target. To mitigate this effect, robot $\mathcal{R}_n$ proactively initiates the broadcasting of *query* messages to the human operator, requesting information on the least tracked target. The index of this target is expressed as

$$\iota_n = \arg\max_{i \in \mathcal{P}_n} (k - t_i) \tag{3.9}$$

By doing so, the robot seeks to improve the accuracy of its target tracking

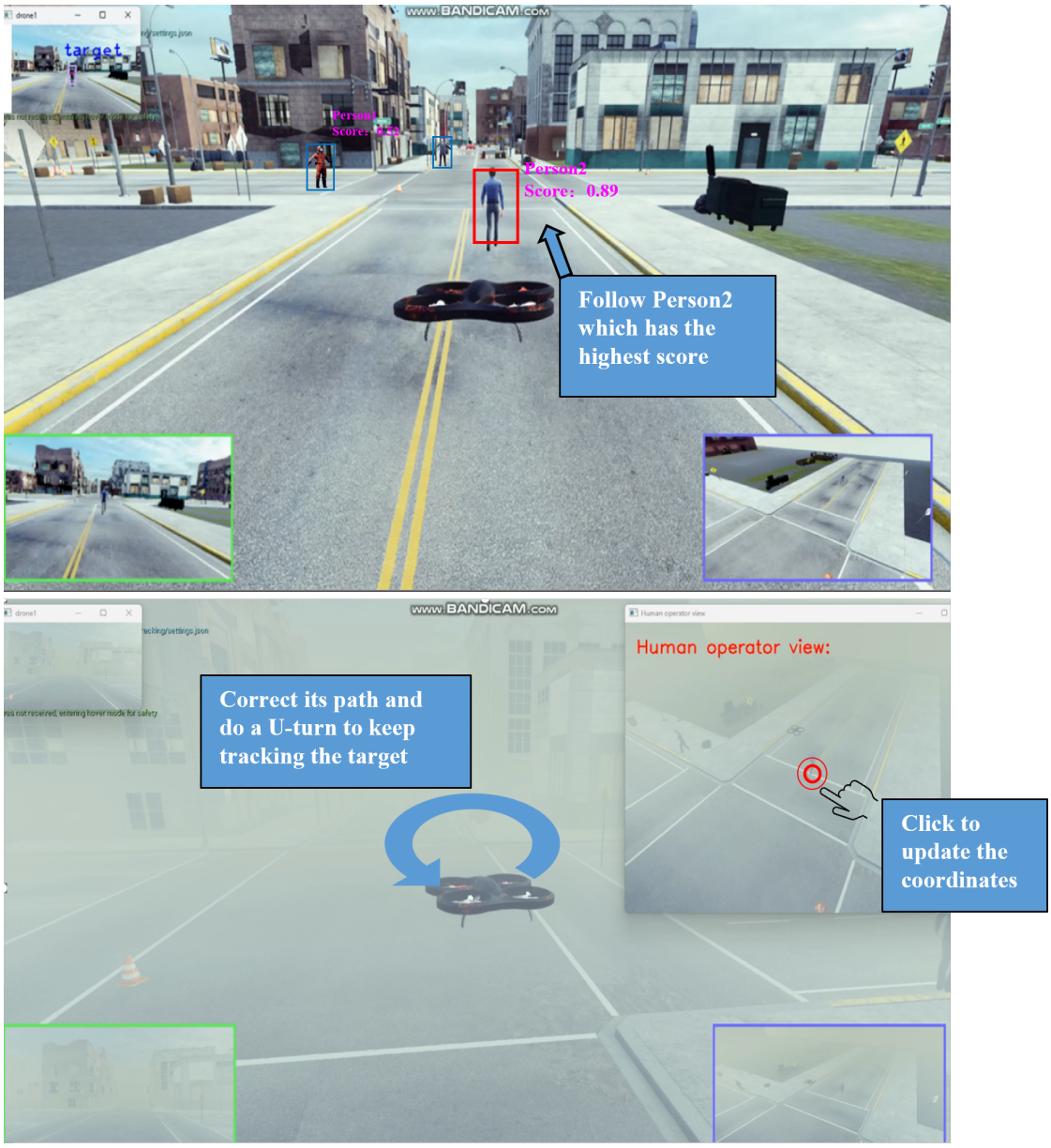

Figure 3.10: Human operator's assistance in tracking.

performance and ensure timely recovery of targets even in the face of dynamic uncertainties.

The missed-tracking time of the least tracked target is $\tau_n = \max_{i \in \mathcal{P}_n} (k - t_i)$, which refers to the duration of time that has passed since the robot $\mathcal{R}_n$ lost track of the

target with the lowest tracking count. This metric is an important indicator of the target's importance and determines the priority of the robot's request for expert information from the human operator. The longer the missed-tracking time, the less accurate the robot's prediction of the target's trajectory, and the less likely it is for the robot to recover the target.

To facilitate the communication between the robots and the human operator, a *query* message is sent by robot $\mathcal{R}_n$ to request information on the least tracked target. This message is defined as $\mathbf{q}_n \triangleq [\iota_n \quad \tau_n \quad n]^T$, where the robot index n serves as a unique identifier for the message, and the priority indicator τ_n informs the operator of the importance of the requested information. The priority indicator is determined by the missed-tracking time of the least tracked target, with larger values indicating higher priority for the operator to provide expert information about the target. By providing this information, the human operator can help the robot team improve their tracking time and increase the likelihood of successfully recovering the target.

The coordination and communication between human operators and robots are crucial for successful target tracking. Human operators can selectively update the states of the least tracked target among all queried targets in response to robot queries, and they can do so using *update* messages. These messages are sent asynchronously, allowing the operator to decide when to update the robot team without affecting the tracking task negatively. In addition, humans can identify unexpected changes in the environment, such as the appearance of new targets, or intruders, and designate robots to track them through *intruder* messages.

To distinguish between these two types of messages, a binary variable

$l \in \{0, 1\}$ is used, where $l = 0$ represents an update message and $l = 1$ represents an *intruder* message. The human message is then encoded as $\mathbf{m} \triangleq [l \quad \hat{n} \quad \jmath \quad \boldsymbol{\chi}_\jmath^T]^T$. Here, $\hat{n}$ is the index of the robot that will receive the message, $\jmath$ is the index of the human selected target, and $\boldsymbol{\chi}_\jmath \in \mathbb{R}^2$ is the human observed target position. By transmitting these messages, the human operator can effectively guide the robots to track the most important targets and adapt to changes in the environment.

The two-way message-exchange scheme is illustrated in Fig. 3.11, which shows a user interface that has been developed specifically to facilitate communication between operators and robots. In order to update its estimation of the target states, the robot relies on the information contained in *update* messages that are received from the operator. These messages typically contain information about the current state of the target, such as its position or velocity, as well as any other relevant contextual information.

Once the update message is received, the robot updates its online estimation of the target states, $\hat{\mathbf{x}}_{n,i}(k)$, as follows:

$$\hat{\mathbf{x}}_{n,i}(k) = \begin{cases} \boldsymbol{\chi}_\jmath & \text{if } \hat{n} = n,\ l = 0,\ \jmath = i \\ \mathbf{z}_{n,i}(k) & \text{if } \hat{n} \neq n,\ \mathbf{x}_{n,i} \in \mathcal{S}_n(k) \\ \mathbf{F}\mathbf{x}_{n,i}(k-1) & \text{otherwise} \end{cases} \tag{3.10}$$

It can also updates its online estimation of the target states using a variety of techniques, such as Kalman filtering or particle filtering. These methods enable the robot to more accurately estimate the current state of the target, which in turn can be used to improve the robot's ability to track and interact with the target.

When a robot receives an *intruder* message, its response is typically different than in other types of human-robot communication. In particular, the robot will append the new target index to its existing assignment matrix, which is denoted as $\mathcal{P}_n = \mathcal{P}_n \cup \{J\}$, with the following initialization:

$$\hat{\mathbf{x}}_{n,|\mathcal{P}_n|}(k) = \boldsymbol{\chi}_J, \quad \text{if} \quad \hat{n} = n,\ l = 1 \tag{3.11}$$

The intruder will then be actively tracked by robot $\mathcal{R}_n$.

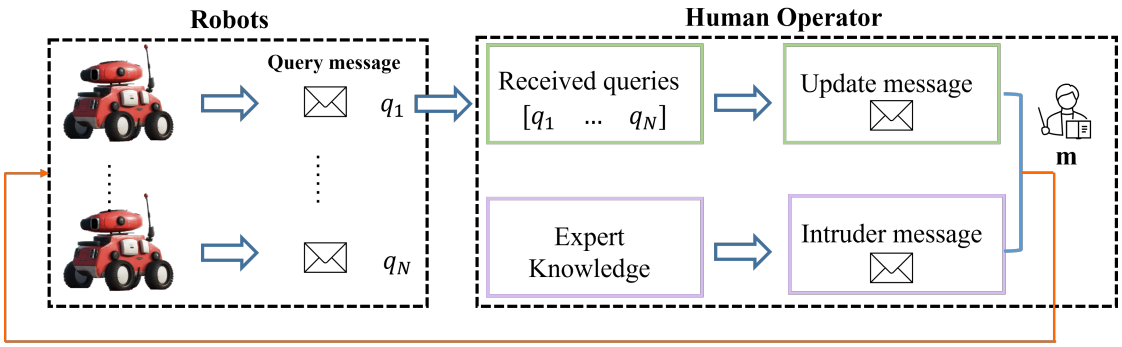

Figure 3.11: Message exchange within the human-robot team.

3.4.3 Tracking Utility Function

In cooperative multi-robot systems, distributed tracking of multiple targets is a challenging task that requires the robots to work together effectively. In this work, the authors propose a distributed cooperative tracking approach where each robot maximizes a local utility function. The local utility functions are designed to align with the global utility function which is aimed at achieving the overall MHRT goal [2] of maximizing the cumulative tracking time of all targets.

In the paper, they consider tracking scenarios where the number of targets exceeds the number of robots, making it impossible to simultaneously and consistently track all targets. To address this challenge, authors propose a novel local utility function that captures the trade-off between the instantaneous improvement in tracking time and the exploration of missed targets.

The local utility function is designed as a combination of two components: a tracking reward and a navigation reward. The tracking reward is a function of the planned robot FOV $\mathcal{S}_n(k+1)$ and the predicted target states $\hat{\mathbf{x}}_{n,i}(k+1)$ obtained by applying the prediction model on the target state estimates $\hat{\mathbf{x}}_{n,i}(k)$. The tracking reward is defined as

$$D_{n,i} = \gamma \cdot \mathbf{1}(\hat{\mathbf{x}}_{n,i}(k+1) \in \mathcal{S}_n(k+1)) \tag{3.12}$$

where $\gamma \in \mathbb{R}^+$ is a reward that has been determined through empirical analysis and is generated if the planned robot FOV is utilized to track a target in the upcoming time step. Therefore, the tracking reward component of the local utility function incentivizes the robots to make immediate improvements in the tracking time of targets by prioritizing their movements and actions towards those that can be detected within their FOV.

The navigation reward, on the other hand, encourages exploration of missed targets by rewarding robots that move towards targets that are not currently within their FOV. This reward is designed to incentivize robots to move towards missed targets in order to improve the overall tracking performance.

By combining the tracking reward and the navigation reward, the authors are able to design a local utility function that effectively captures the trade-off between tracking time and exploration. This allows the robots to work together to maximize the overall tracking time, while also exploring new areas of the en-

vironment in order to identify missed targets. By doing so, the authors demonstrate the effectiveness of their approach in achieving the MHRT goal in challenging multi-target tracking scenarios.

Next, assume the workspace $\mathcal{W}$ is tessellated into equally sized and disjoint 2D cells [27] represented by $C = \{C_1, \ldots, C_q\}$. Letting $\mathbf{y}_\ell \in \mathbb{R}^2$ denote the 2D centroid coordinate of the ℓ^{th} cell, an attractive potential for target $\mathcal{T}_i$ is defined in terms of the Euclidean distance between the centroid of each cell and the target's predicted position

$$M_i(\ell) = \eta(t_i) \cdot \|\mathbf{y}_\ell - \hat{\mathbf{x}}_{n,i}(k+1)\|, \quad \forall \ell \tag{3.13}$$

where $\eta(t_i)$ is a factor that is inversely proportional to the tracking time t_i, hence incentivizing exploration of the lesser tracked targets. Then, the navigation reward for robot $\mathcal{R}_n$ is defined as the cumulative negative attractive potential of the cells covered by the planned FOV $\mathcal{S}_n(k+1)$

$$M_{n,i} = -\sum_{\ell=1}^{q} M_i(\ell) \cdot \mathbf{1}(C_\ell \in \mathcal{S}_n(k+1)) \tag{3.14}$$

where the negative sign ensures consistency with the maximization framework such that larger rewards pull the robot closer to the targets. Then, the local utility is defined as

$$U_{n,i}(\mathbf{z}_{n,i}, \mathbf{s}_n, \mathbf{m}) = M_{n,i} + D_{n,i} \tag{3.15}$$

Notice that the target observation $\mathbf{z}_{n,i}$, robot state $\mathbf{s}_n$, and human message $\mathbf{m}$ are implicitly integrated to the utility function through $\hat{\mathbf{x}}_{n,i}(k)$ and $\mathcal{S}_n(k+1)$ in (3.12-3.14).

Finally, collision with static obstacles $\mathcal{B} \in \mathcal{W}$ is avoided by incurring a penalty $\zeta \in \mathbb{R}^+$ on the planned robot state $\mathbf{s}_n(k+1)$ that will collide with the obstacles

$$C(\mathcal{B}) = \zeta \cdot \mathbf{1}(\mathbf{s}_n(k+1) \in \mathcal{B}) \tag{3.16}$$

Combining (3.15-3.16) together gives the objective function for distributed optimization that is solved locally and concurrently by each robot.

3.4.4 Simulation Results

The cooperative tracking approach presented in this paper is thoroughly assessed via both simulation-based and real-world experimental scenarios. In order to make a comparison, a weak-cooperative tracking algorithm and a non-cooperative tracking algorithm are also implemented within the simulation framework.

In the simulation environment, MHRT is set up within a 100m x 50m workspace, where four mobile robots are randomly placed. A human operator observes the workspace through a surveillance camera, tasked with tracking six targets that move unrestrained throughout the same workspace. Fig.3.12-3.13 display a sample testing instance, where targets are designated by the color of the robot assigned to them, and the area observed by the human operator is represented by a pink polygon. Although the initial tracking scenario in Fig.3.12 presents a significant challenge, with no target in the robots' Field of View (FOV), the combined human-robot team effectively cooperates and successfully tracks all targets by the time step k = 17, as shown in Fig.3.13.

A separate investigation compares the proposed *Cooperative tracking* approach against two alternative strategies: 1) *Non-cooperative tracking*, in which robots individually optimize the local tracking utility function without any human operator involvement, and 2) *Weak cooperative tracking*, where robots transmit query messages, but omit the priority indicator (explained in Section 3.4.2)

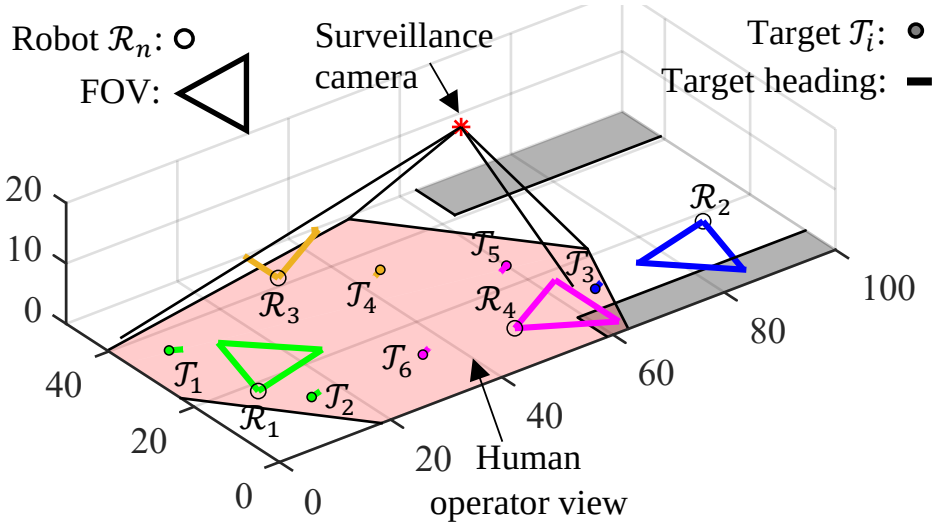

Figure 3.12: The initial configuration of a testing instance.

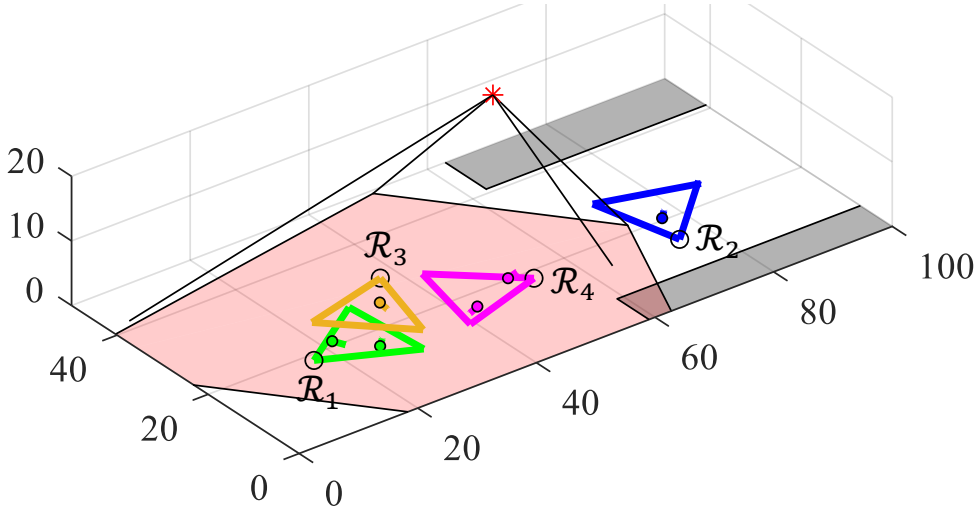

Figure 3.13: Illustration of the MHRT tracking all targets at time step $k = 17$.

necessary for informing human operators about the least tracked target. This comparison study comprises 20 testing instances for each of the three aforementioned strategies. The outcomes are quantitatively appraised using two time-based metrics: the Average Target Tracking Rate (ATTR), calculating the ratio of average target tracking time to the total simulation time, and the Minimum Target Tracking Rate (MTTR), determining the ratio of tracking time for

the least tracked target to the total simulation time. Superior performance is indicated by higher values for both metrics.

It's shown that the performance vary a lot across various tracking strategies as presented in Table 3.1. The low value of MTTR in non-cooperative tracking suggests that a homogeneous robot team is likely to lose track of at least one target for the majority of the time. This occurs because the robots, having limited and directional FOV, cannot guarantee simultaneous and consistent tracking of a larger number of freely moving targets. Additionally, the increasing uncertainty in target estimates as a target is lost makes it difficult for the robots to recover the lost target.

The weak-cooperative tracking strategy offers a modest improvement over non-cooperative tracking by facilitating communication within the mixed human-robot team. However, the robots fail to share information about the lesser tracked targets effectively, causing human operators to be short-sighted about target priorities when deciding which targets to update.

In contrast, the cooperative tracking strategy consistently delivers the best performance due to the implementation of a two-way cooperation mechanism. On one hand, robots actively send messages containing priority indicators about missed targets, which aids human decision-making. On the other hand, the human operator selectively updates the target with the highest priority based on the robot queries, guiding the robots to recover lost targets and ultimately improve the overall tracking time.

These findings underscore the importance of effective communication and collaboration between human operators and robots in achieving optimal track-

ing performance. By addressing the limitations of the non-cooperative and weak-cooperative strategies, the cooperative tracking approach proves to be a more reliable and efficient solution for multi-target tracking applications.

Table 3.1: Tracking Performance Comparison

Methods	**ATTR**	**MTTR**
Non-cooperative tracking	44.2%	5.6%
Weak-cooperative tracking	50.6%	11.8%
Cooperative tracking	57.0%	26.5%

CHAPTER 4

FUTURE WORK

In the preceding sections, we have described the development of an autonomous vehicle that can operate in a virtual environment. This virtual vehicle is designed to perform several functions, including target detection, target tracking, and path planning. The virtual vehicle is capable of operating in extreme environments, with the aid of human operators who can intervene when necessary to correct the vehicle's path.

To further test and refine the capabilities of the virtual vehicle, we intend to integrate it with a real robot car in our laboratory. This will enable us to evaluate the virtual vehicle's performance in a more realistic setting, where it must navigate a physical environment and interact with other objects and vehicles. Specifically, we will use the virtual car to perform obstacle avoidance and send control signals to the real car via a local area network as shown in figure 4.1. The virtual robot will move around the box in the simulation environment. And the real lab robot will do the same as the virtual robot.

By allowing the virtual car to control the real car autonomously, we can test the system's ability to handle unexpected situations and refine its algorithms for more efficient and effective operation. This approach will enable us to identify any limitations of the virtual car and to improve its functionality as necessary. Ultimately, the goal of this research is to develop autonomous vehicle technology that can enhance transportation safety, efficiency, and accessibility.

In this paper, we provide substantial evidence affirming the effectiveness of virtual simulation as a viable tool for real robot simulation. Our research re-

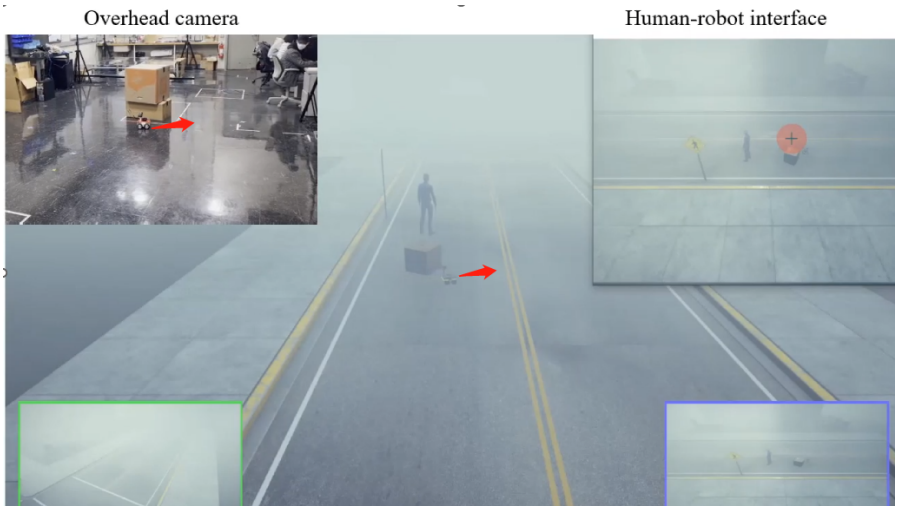

Figure 4.1: Illustration of the simultaneously obstacle avoidance.

volves around successfully applying various AI algorithms to a virtual drone and assessing their performance and reliability across several case studies. Through this practical approach, we were able to underscore the utility of virtual environments for developing, testing, and optimizing AI algorithms, thus contributing to their improvement before transitioning to real-world applications.

The experimental findings demonstrated an impressive correlation between the simulation predictions and the actual outcomes, reinforcing the validity of our approach. Moving forward, our research intends to advance the scope of this study by incorporating drones as agents in our simulations. This strategic addition will introduce the complexities of a three-dimensional environment, requiring considerations for the Z-axis.

Moreover, we aim to replace the virtual human avatar with a real human avatar to further enhance the authenticity of the simulation. By leveraging the capabilities of Virtual Reality (VR) headsets, we plan to simulate a human pres-

ence within the virtual environment. This immersive VR experience will help us better understand the intricate dynamics between humans and robots in a shared environment, enabling us to develop AI algorithms that are highly adaptive, responsive, and effective in real-world scenarios. These advancements are projected to significantly contribute to the evolving field of robotics and AI.

CHAPTER 5

ACKNOWLEDGEMENT

I would like to take this opportunity to extend my heartfelt gratitude to several individuals without whom this thesis would not have been possible.

Firstly, I would like to express my sincere appreciation to Silvia, whose exceptional projects have been a constant source of inspiration and learning for me. Her challenging tasks have shaped my understanding and competence in the field, for which I am truly grateful. I am pleased to report that I have completed the path planning implementations and transitioned them to the Unreal Engine under her guidance. I have managed to reconstruct the 3D environment and human characters in the virtual environment for all the projects. I also implement the Airsim package in the environment which give us the drone platform for all the tasks.

My deep appreciation goes to Dr. Yuchen Chen for his invaluable theoretical guidance. His expertise and commitment have provided a robust academic framework to my work, strengthening the overall quality of my thesis.

I am deeply thankful to Dr. Junyi Dong and Sushrut for their substantial contributions to the robots' collaboration framework, target state estimation, the decentralized robots' control, and online sensing. Their insights have greatly enhanced the technical depth of my work.

Special thanks to Cong Liu and Rak for their significant role in establishing the communication network between the lab robot and the virtual robot. Their expertise in network communication has been crucial to the successful integration of our systems.

Lastly, I would like to acknowledge my own contributions to this project. I am proud of the work I have put into the human-robot interface design, fog system design, and the implementation of YOLO, Detectron 2, and Monodepth on the drone.

Thank you all for your support, guidance, and invaluable contributions. This journey would not have been as rewarding and fulfilling without your help.

BIBLIOGRAPHY

[1] Marco Aggravi, Giuseppe Sirignano, Paolo Robuffo Giordano, and Claudio Pacchierotti. Decentralized control of a heterogeneous human-robot team for exploration and patrolling. *IEEE Transactions on Automation Science and Engineering*, 2021.

[2] Gürdal ARSLAN, Jason R MARDEN, and Jeff S SHAMMA. Autonomous vehicle-target assignment: A game-theoretical formulation. *Journal of dynamic systems, measurement, and control*, 129(5):584–596, 2007.

[3] M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on signal processing*, 50(2):174–188, 2002.

[4] Jacopo Banfi, Jérôme Guzzi, Francesco Amigoni, Eduardo Feo Flushing, Alessandro Giusti, Luca Gambardella, and Gianni A Di Caro. An integer linear programming model for fair multitarget tracking in cooperative multirobot systems. *Autonomous Robots*, 43(3):665–680, 2019.

[5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[6] Jieneng Chen, Yongyi Lu, Qihang Yu, Xiangde Luo, Ehsan Adeli, Yan Wang, Le Lu, Alan L. Yuille, and Yuyin Zhou. Transunet: Transformers make strong encoders for medical image segmentation, 2021.

[7] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a*. 32(3), 1985.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[9] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 2010.

[10] L. Feng, Bart Everett, and J. Borenstein. Where am i? : sensors and methods for autonomous mobile robot positioning. 02 2006.

[11] Silvia Ferrari and Thomas A Wettergren. *Information-Driven Planning and Control*. MIT Press, 2021.

[12] Jodi Forlizzi, Carl F. DiSalvo, and Francine Gemperle. Assistive robotics and an ecology of elders living independently in their homes. *Human–Computer Interaction*, 19:25 – 59, 2004.

[13] Eric W Frew and Jack Elston. Target assignment for integrated search and tracking by active robot networks. In *2008 IEEE International Conference on Robotics and Automation*, pages 2354–2359. IEEE, 2008.

[14] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32:1231 – 1237, 2013.

[15] Clement Godard, Oisin Aodha, and Gabriel Brostow. Unsupervised monocular depth estimation with left-right consistency. 07 2017.

[16] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency, 2017.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[18] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7272–7281, 2017.

[19] Saurabh Gupta, Ross B. Girshick, Pablo Arbeláez, and Jitendra Malik. Learning rich features from rgb-d images for object detection and segmentation. *ArXiv*, abs/1407.5736, 2014.

[20] Christian G Harris, Zachary I Bell, Runhan Sun, Emily A Doucette, J Willard Curtis, and Warren E Dixon. Target tracking in the presence of intermittent measurements by a network of mobile cameras. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 5962–5967. IEEE, 2020.

[21] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[22] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, Venice, 2017.

[23] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[25] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *CoRR*, abs/1105.1186, 2011.

[26] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[27] Asif Khan, Evsen Yanmaz, and Bernhard Rinner. Information exchange and decision making in micro aerial vehicle networks for cooperative search. *IEEE Transactions on Control of Network Systems*, 2(4):335–347, 2015.

[28] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[29] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[30] J.J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2, 2000.

[31] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[32] Su-Jin Lee, Soon-Seo Park, and Han-Lim Choi. Potential game-based non-myopic sensor network planning for multi-target tracking. *IEEE Access*, 6:79245–79257, 2018.

[33] Keith A LeGrand, Pingping Zhu, and Silvia Ferrari. A random finite

set sensor control approach for vision-based multi-object search-while-tracking. In *2021 IEEE 24th International Conference on Information Fusion (FUSION)*, pages 1–8. IEEE, 2021.

[34] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[35] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[36] Wenjie Lu, Guoxian Zhang, and Silvia Ferrari. An information potential approach to integrated sensor path planning and control. *IEEE Transactions on Robotics*, 30(4):919–934, 2014.

[37] Igiri Onaji, Divya Tiwari, Payam Soulatiantork, Boyang Song, and Ashutosh Tiwari. Digital twin in manufacturing: conceptual framework and case studies. *International Journal of Computer Integrated Manufacturing*, 35(8):831–858, 2022.

[38] Taihú Pire, Thomas Fischer, Gastón I. Castro, Pablo de Cristóforis, Javier Civera, and Julio Jacobo-Berlles. S-ptam: Stereo parallel tracking and mapping. *Robotics Auton. Syst.*, 93:27–42, 2017.

[39] Morgan Quigley. Ros: an open-source robot operating system. In *IEEE International Conference on Robotics and Automation*, 2009.

[40] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[41] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[42] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson series in artificial intelligence. Pearson, 2021.

[43] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Longman Publishing Co., Inc., USA, 1990.

[44] Johannes L. Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4113, 2016.

[45] Nathan R. Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.

[46] Hongchuan Wei, Pingping Zhu, Miao Liu, Jonathan P How, and Silvia Ferrari. Automatic pan–tilt camera control for learning dirichlet process gaussian process (dpgp) mixture models of multiple moving targets. *IEEE Transactions on Automatic Control*, 64(1):159–173, 2018.

[47] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[48] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video, 2017.