Volley

Xiao Feng, Jason Ha, Zherui Luo, Arthur Wang
December 12, 2024
EK505, Fall 2024

Abstract

When playing racquet sports such as table tennis, tennis, pickle-ball, etc., a significant amount of time is taken chasing balls after points. This is especially true for solo practice, where dozens of balls might be used and need collecting. As such, our company is focusing on building robots that will help to automate this task to augment and enrich the player's experience. The Loop autonomous robot is built on the LIMO Pro developed by AgileX and is controlled by our Volley Autonomy Stack (VAS) incorporating SLAM, CNN Object Detection, Deep Reinforcement Learning, and more to efficiently detect and gather balls.

1 Introduction and Background

In this report, Volley presents the motivation, detailed technical documentation and testing results of our Loop autonomous robot. The motivation for such a robot comes from lived and heard experience of table tennis players. One of the most tedious things about table tennis is chasing after the balls as they scatter across the play area. This is especially vexing when the balls carry great velocity and spin, causing them to end up far away from the table as in professional play.

Counterintuitively, ball collection is more of a problem for beginner players due to the lower skill level. One might expect that weaker hits and spin would result in less tedious ball collection, but this is far from the truth. In reality, low skill players cannot keep up a long rally. This means that while the balls might not stray as far, the amount of balls needing collection is greatly increased.

The problem becomes even more exacerbated for inexperienced solo practice. Table tennis players often train with the aid of a table tennis ball launching machine that can adjust the speed, direction and spin. While expensive models come with their own net and feed tray, cheaper models do not and require manual refilling. Furthermore, amateur players can easily miss the collection net leading to manual collection. As such, it would be immensely beneficial to have an autonomous robot that could automate the collection task.

Section 2 is our analysis of the competition and wider market for ball collecting robots. It additionally reviews relevant literature and research that inspires modules of the Volley Autonomy Stack (VAS) in Subsection 2.1. Subsection 2.2 highlights key contributions and features that set the Loop apart from the competition.

Section 3 provides an overview of the system including hardware such as the LIMO Pro and its relevant sensors in Subsection 3.1. It additionally details assumptions about the Loop's operating environment as well as deployment limitations in Subsection 3.2. Subsection 3.3 wraps up with an overview of VAS and with brief descriptions of the software modules contributed by our founding members.

Finally, Section 4 is a deep dive into the technical contributions and analysis performed by our members including the SLAM system, Perception, Path Planning and Delivery modules. Each module is accompanied by explanations of relevant theory, pseudo-code algorithms, analysis results, and a short discussion on the risks and limitations.

2 Related Work and Market Research

2.1 Related Work

2.1.1 Scan Matching SLAM

In this project, Scan Matching Simultaneous Localization and Map Building (SLAM) is used to realize autonomous mapping and accurate positioning of the robot in a dynamic environment [13]. The method uses the LiDAR sensor to obtain the two-dimensional point cloud data of the surrounding environment in real-time as can be seen in Figure 1. It calculates the pose change of the robot relative to the map by matching the features between the continuous scanning frames and constantly updates the environment map.

2.1.2 CNN Object Detection

In the realm of robotics, the integration of object detection systems with autonomous platforms, such as the LIMO car, has garnered significant attention. Previous studies have leveraged advanced object detection models, including YOLO and SSD, to enable robust navigation and environment interaction [7]. However, adapting these models to detect small, fast-moving objects like table tennis balls introduces unique challenges, particularly in dynamic and resource-constrained settings.

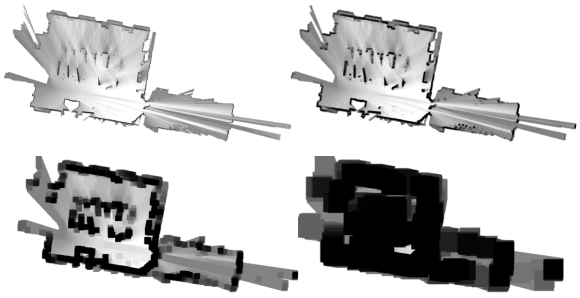

Figure 1: Adapted from Figure 3 of Hess et al. [3]. Lidar scan precomputed grids of size 1, 4, 16 and 64.

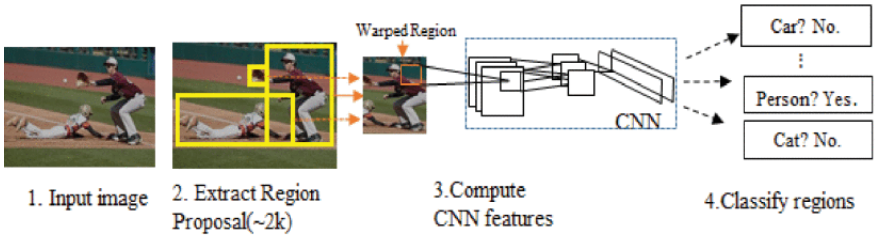

Figure 2: Adapted from Figure 4 of Kim et al. [7]. Object Recognition Sequence of CNN.

Our work builds upon this body of research by exploring the use of MobileNetV3 as a backbone to enhance the efficiency and accuracy of table tennis ball detection. Specifically, we replace YOLOv8's default backbone, DarkNet, with MobileNetV3 and integrate coordinate attention mechanisms to improve spatial and channel feature encoding [4]. This approach enables the model to capture fine-grained features critical for detecting small objects in complex environments faster. The resulting model is deployed on the LIMO car platform, demonstrating its effectiveness in real-time detection scenarios.

2.1.3 External Skeletonization of Binary Images

Morphological Skeletonization, also known as Medial Axis Representation, is a method to extract a skeleton from shapes in an image [18]. This is a classical computer vision method which can be used to find the centerline between contours. Traditional skeletonization of an occupancy map would result in median axes running between each obstacles and the map boundary and offshoots connected to boundaries. For our applications, we require no offshoots and we must also avoid the play area centered on the table tennis table. Niblack et al. proposes an algorithm to only retrieve the centerline of the skeleton which we modify to retrieve the external skeleton centerline [12]. The difference between a centerline and a full skeleton can be seen in Figure 3.

2.1.4 Deep Reinforcement Learning Path Planning

Deep Q-Learning was first proposed by Mnih et al. to play Atari [11]. For the Path Planning module, the Loop utilizes the Double Dueling Deep Q-Network (D3QN) reinforcement learning method to learn to avoid obstacles while moving towards a goal in a Grid World (GW). D3QN is an improvement over the basic Deep Q-Network (DQN) by Mnih et al. and Double Deep Q-Networks (D2QN) proposed by van Hasselt et al. which uses a second network to prevent overestimation [21]. The D3QN method introduces the separation of Value and Advantage in the neural network to better generalize learning over different actions without changing the learning process [22]. This separation is achieved by splitting the network into two streams. One learns to estimate the Value and the other to estimate the Advantage of each action. This architecture can be seen in Figure 4.

Figure 3: Adapted from Figure 5 of Niblack et al. [12]. Difference between centerline skeleton and full skeleton.

The paper by Yin et al. uses a rewards based e-greedy with a D3QN to better learn grid-based path planning around obstacles [23]. Although successful, this method proved difficult to extrapolate to the larger grid world used by Loop. However, the learning stability and ability of D3QN over other Deep Q-Network (DQN) methods led us to adopt it for our Path Planning module.

Like Lei et al., we implement a PyGame based simulation to model LiDAR behavior as an agent explores its environment [8]. We adopt some of their training techniques as they work in a grid environment with randomly placed goal points and obstacles as well.

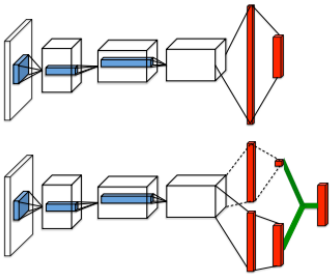

Figure 4: Adapted from Figure 1 of Wang et al. [22]. Difference between single stream Q-network (top) and a dueling Q-network (bottom).

2.1.5 AprilTags

The study by Betke and Gurvits introduces a method for localizing mobile robots using static landmarks, focusing on environments where a robot can identify fixed landmarks and measure their relative bearings [1]. Their approach is efficient, leveraging complex number representations for landmarks to streamline the process, making it computationally feasible with a linear time complexity. This is particularly advantageous for real-time navigation, allowing robots to determine their position and orientation even with noisy inputs.

Kallwies et al. explores how AprilTags can improve a robot's ability to pinpoint its location, especially when these tags are placed in fixed positions within the environment [6]. Their work demonstrates that when a robot can detect these tags, it significantly boosts its navigational accuracy, making it a reliable reference point in dynamic settings.

Figure 5: From top left in clockwise order: Echo Robotics Golf Picker. [15], TenniBot [17], Relox Range Picker [16].

2.2 Market Research

TenniBot is an automated ball-picking robot designed for tennis courts. It is capable of intelligently picking up tennis balls and storing them in containers. Although its design is extremely intelligent, TenniBot's application scenarios are strictly limited to tennis courts and cannot be easily adapted to different sports environments. Additionally, it is quite large making it unsuitable for the smaller size of table tennis play areas. On the other hand, Echo Robotics Golf Picker and Relox Range Picker are mainly targeted at golf driving ranges. They have large ball pick-up and storage capabilities as well as autonomous path planning functions. But, they are still limited to picking up large balls in specific sports venues and rely on GPS for positioning. They also cannot be adapted easily for indoor or small court applications.

Unlike these products, our Loop focuses on picking table tennis balls, a market that is currently completely untapped. As a widely popular sport, table tennis has a huge demand for picking up balls generated by training and competition. Currently, there are no automated solutions designed for this demand in the market which we plan to exploit. We use advanced SLAM technology to enable accurate indoor positioning and path planning while combining LiDAR and RGB-D cameras to achieve high-precision table tennis ball detection and picking. In addition, we use deep reinforcement learning to optimize our ball picking strategy so that our products can operate efficiently in dynamic environments.

Compared with TenniBot and golf ball pickup robots, Volley fills the market gap in table tennis ball collection, and surpasses existing robotic ball collector products in accuracy and adaptability through technical advantages. Thus, Loop possesses a unique competitiveness in the market. In addition, TenniBot has made $1.4 million in revenue in 2024 alone, which is quite an achievement given that there are similar competing products on the market. However, our products do not have similar competitive products to contend with. We are the first product in the field of table tennis collection and our products have a higher potential and revenue than TenniBot.

3 System Overview

3.1 Robotic and Hardware System

Our project uses a robotic system that integrates advanced technology and is designed to achieve efficient table tennis ball collection. The entire system is built on the AgileX LIMO Pro platform, a mobile robot platform known for its flexibility and adaptability. The LIMO Pro is equipped with a mecanum wheel design that enables flexible omnidirectional movement in limited indoor spaces. This is ideal for confined environments such as table tennis courts.

In terms of hardware configuration, the robot is equipped with a variety of advanced sensors providing the system with strong environmental perception and positioning ability as seen in Figure 6. The EAI XL2 LiDAR generates

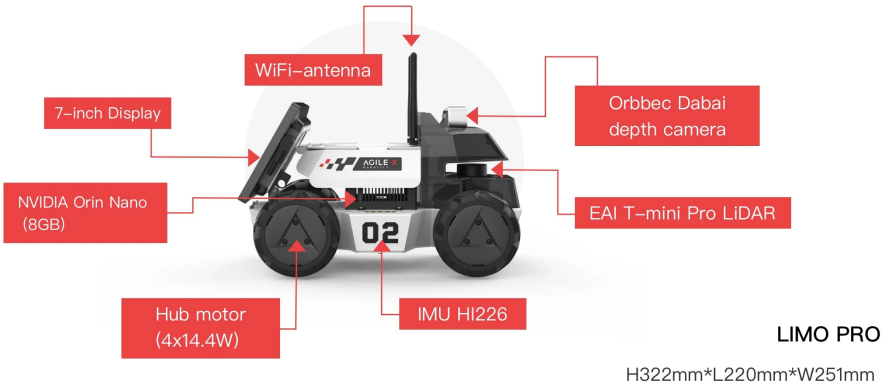

Figure 6: AgileX LIMO Pro schematic including relevant sensors and hardware components[16].

accurate 2-D or 3-D point cloud data for real-time site mapping and path planning. This is supplemented with an ORBBEC® Dabai RGB-D camera for other perception tasks. With these sensors, Loop is able to achieve accurate table tennis ball detection, recognition and collection.

The LIMO Pro uses a NVIDIA Jetson Orin Nano, a high-performance embedded computing platform designed for compute-intensive AI tasks. It supports running object detection algorithms and deep reinforcement learning models in real time. This high computing power ensures rapid response and intelligent decision-making in dynamic environments.

The core algorithm of the whole system is based on 2-D Scan SLAM technology, using LiDAR and camera data to build and locate the environment. This technology is particularly important for indoor venues such as table tennis courts, as it can generate accurate real-time maps and enable adaptive path planning for robots in complex environments. In addition, the system incorporates a deep reinforcement learning path planning algorithm , which makes it perform particularly well in dynamic obstacle and high-density target scenarios.

3.2 Assumptions and Limitations

The LIMO car operates under the assumption that it will function in indoor environments with controlled lighting conditions, ensuring predictable detection scenarios. It is also designed for flat and structured surfaces, allowing smooth movement and simplified navigation. These assumptions eliminate the need for advanced terrain-handling capabilities or adaptations to harsh or unpredictable environments, focusing its operation on well-structured, controlled spaces.

One of the operational limits of the LIMO car is its limited sensor ranges, making it difficult to detect objects in obscure areas or unreachable areas. In congested settings, where overlapping items add noise and lower detection confidence, it performs poorly. Furthermore, the car's capacity to perform retrieval tasks in obstructed environments is limited since it is unable to engage with table tennis balls that are inaccessible, such as those that are under furniture or on raised surfaces.

3.3 Autonomy Stack

Our Volley Autonomy Stack (VAS) integrates Perception, Path Planning and Decision Making into a compact software framework. As can be seen in Figure 7, in normal operation off of the charging dock, a LiDAR scan is performed. This 2-D scan is used to perform scan-matching with previous LiDAR observations to localize Loop with respect to its surroundings. This is vital as all processes have noise and to accurately navigate to table tennis

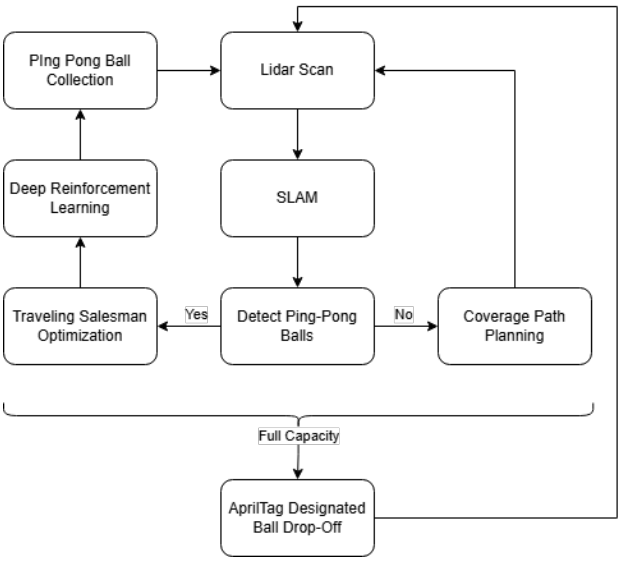

Figure 7: System flow diagram of the VAS including perception, path planning and overall decision logic.

detections, a precise location of the robot in its environment is needed. The SLAM implementation is described in more detail in Subsection 4.1.

The RGB-D camera is used to detect table tennis balls using a Convolutional Neural Network (CNN) using the RGB information. By modifying YOLOv8, a popular and effective CNN, with MobileNetV3, a lightweight CNN backbone, Loop is better able to perform real-time detection tasks. The exact process is detailed in Subsection 4.2.

If one or more table tennis balls are detected, their distances to Loop are estimated by indexing the corresponding depth image using detection bounding box coordinates. Multiple table tennis balls must be collected in the order that minimizes distance traveled, a type of Traveling Salesman Problem (TSP). This visitation order is kept in a queue which the path planning module reads from to move to a goal position.

The path planning module utilizes two methods to direct Loop. The first is using a D3QN reinforcement learning method to navigate around obstacles towards a table tennis ball position in real-time. The second, when there are no table tennis balls detected by the CNN, utilizes an skeletonization method to generate a path for room sweeping. More details can be found in Subsection 4.4 and 4.3 respectively.

When arriving within a certain distance of a table tennis ball, Loop will collect and store the ball and continue to plan towards the next table tennis ball if applicable. Once the Loop's capacity is reached it will home onto a detected AprilTag's position using the reinforcement learning policy. The AprilTag is detected during coverage or goal seeking and can be placed anywhere accessible to the Loop.

4 Technical Contributions and Analysis

4.1 Scan Matching SLAM

4.1.1 Scan Matching SLAM Theory

The core of Scan Matching SLAM is to establish a reliable point cloud matching relationship through the scanning data of LiDAR [13]. Specifically, the point cloud data obtained from each scan is compared with the point cloud of the previous frame, and the current pose change of the robot is calculated using an optimization algorithm.

During construction, the robot integrates the point cloud data from each scan into a global map to generate a

two-dimensional raster representation of the environment. The map not only reflects the distribution of obstacles around the robot, but also provides high-precision environmental information support for the path planning module. In addition, Scan Matching SLAM synchronizes positioning and mapping by continuously updating the pose relationship between the robot and the map, thus ensuring the robot's navigation ability in unknown environments [3].

4.1.2 Implementation

We simulate the functions of 2-D LiDAR with a custom LiDAR module. The LiDAR acquires point cloud data of the surrounding environment at a fixed resolution (in the code we set it to have 360 beams per scan). The end coordinates of these beams are calculated from the LiDAR's current position and scanning range. In order to simulate the uncertainty of the real sensor, Gaussian noise is added to the scanning distance and angle after acquisition to reflect the measurement error of the sensor in the real environment.

The point cloud data generated after the LiDAR scan is further converted into a polar coordinate format in order to calculate the relative distance and angle difference between points during subsequent point cloud matching. Through such pre-processing, the SLAM module is able to extract critical information from complex sensor data [3][9].

The system integrates point cloud data into the global map to update the environment model. The map takes the form of a two-dimensional grid, reflecting the distribution of obstacles around the robot. The realization of closed-loop detection is completed by matching the current frame point cloud with the saved submap in the global map [3]. When the closed loop is detected, the system uses the image optimization method to correct the pose error and recalibrate the global map.

The core of map update is how to effectively integrate old and new data. In the code, the new scan point cloud is merged with the occupied state of the current map through a weighted update mechanism [10]. The value of each grid is weighted according to the confidence of its history with the new data, ensuring that the map reflects the current environment. In practice, each beam scanned by the LiDAR is projected onto a grid map, calculating all the grids it covers, and updating the occupancy probability of those grids. The code uses sparse data storage structures to optimize storage and computational efficiency, especially for map construction in a wide range of environments.

4.1.3 Results & Analysis

In the SLAM simulation results of this project, as shown in the Figure 8, the system realizes real-time positioning and map construction through LiDAR. The results show that the blue trajectory generated by SLAM represents the actual position estimated by the robot based on LiDAR scanning and point cloud matching. The green trajectory represents the actual position of the robot measured by external sensors. The red track is the ideal path for comparison and evaluation of SLAM performance. White dots and rectangles in the scene represent obstacles and environmental boundaries.

The localization accuracy of SLAM is evaluated by comparing the blue trajectory (the SLAM estimated location) with the green trajectory (the real location). It can be observed that the blue trajectory can roughly follow the green trajectory accurately, but there is a certain degree of lag and offset near fast turns and dynamic obstacles. This error is mainly due to the uncertainty of LiDAR point cloud matching and the influence of sensor noise.

While SLAM performs well overall, there is still room for improvement in complex scenarios. For example, when a robot passes over a dynamic obstacle, dynamic interference in the point cloud data can lead to transient positioning errors.

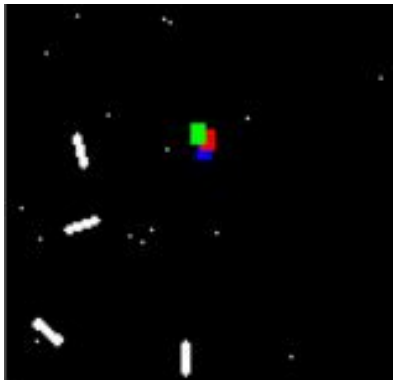

Figure 8: SLAM simulation results.

4.2 CNN Based Table Tennis Ball Detection

4.2.1 YOLOv8 & MobileNetV3 & Coordinate attention Theory

1.YOLOv8

YOLOv8 in Figure 9 (You Only Look Once, Version 8) is the latest version in the YOLO series of real-time object detection frameworks, known for its balance of accuracy, speed, and usability [20]. It introduces several improvements over its predecessors, making it a highly robust and flexible solution for diverse object detection tasks. Below are its key characteristics and features:

Dynamic Anchor Boxes: During training, YOLOv8 uses dynamic anchor boxes that adjust to the dataset on their own. This adaptability increases the accuracy of detection, particularly for items with large size variations, like table tennis balls in our application.

Advanced Detection Heads: The detection heads in YOLOv8 are designed to predict bounding boxes, class probabilities, and object confidence scores efficiently, utilizing a unified end-to-end training pipeline.

State-of-the-Art Performance: YOLOv8 achieves excellent performance on standard benchmarks, with improvements in mean Average Precision (mAP), precision, and recall compared to earlier YOLO versions.

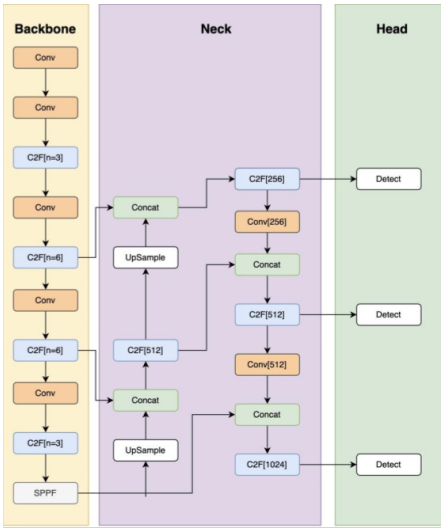

Figure 9: YOLOv8 architecture.

Despite YOLOv8's strong feature extraction and excellent detection accuracy, its computational demands make it difficult to implement on systems with constrained memory and computing capability. We intend to preserve YOLOv8's detection capabilities while drastically lowering its computing load by substituting MobileNetV3 for the CSP-Darknet backbone, making it feasible for real-time use on a LIMO car.

2.MobileNetV3

MobileNetV3 was selected to replace YOLOv8's CSP-Darknet backbone due to its lightweight design and high efficiency [5]. Key aspects of MobileNetV3 include:

Advanced Activation Functions: Compared to CSP-Darknet in YOLOv8, MobileNetV3 uses lightweight blocks like Conv BN HSwish to maintain efficiency, making it suitable for real-time applications on resource-constrained platforms like the LIMO car. This algorithm can be found in Algorithm 1.

Efficient Building Blocks: Separable convolutions in depth reduce the computation, while the squeeze and excitation modules enhance the extraction of characteristics. This algorithm can be found in Algorithm 2.

3.Coordinate Attention

To enhance the detection capability of the MobileNetV3 backbone, we incorporated the Coordinate Attention mechanism [4]. This module is specifically designed to improve the model's ability to capture cross-channel relationships and spatial information, which is essential for accurately detecting small and spatially distributed objects like table tennis ball. This algorithm can be found in Algorithm 3.

Algorithm 1 Conv_BN_HSwish: Convolution + BatchNorm + Hardswish.

Input: c_1 (input channels), c_2 (output channels), *stride*
Output: Processed tensor with Convolution, BatchNorm, and Hardswish
procedure INIT(c_1, c_2, *stride*)
    Define a 3×3 convolution with stride $=$ *stride* and padding.
    Add BatchNorm layer for c_2 channels.
    Add Hardswish activation function.
procedure FORWARD(x)
    Apply the convolution operation on x.
    Apply BatchNorm on the result.
    Apply Hardswish activation on the result.
    Return the processed tensor.

Algorithm 2 MobileNetV3_InvertedResidual: Main Block.

Input: inp, oup, $hidden\_dim$, *kernel_size*, *stride*, *use_se*, *use_hs*
Output: Processed tensor with optional SE block and skip connections
procedure INIT
    Set *identity* $\leftarrow$ (*stride* $= 1 \wedge inp = oup$)
    if $inp = hidden\_dim$ **then**
        Define depthwise convolution, BatchNorm, activation
        if *use_se* **then** Add SE block
        Add pointwise convolution, BatchNorm
    else
        Define pointwise convolution, BatchNorm, activation
        Add depthwise convolution, BatchNorm
        if *use_se* **then** Add SE block
        Add final activation, pointwise convolution, BatchNorm

Algorithm 3 CoordAtt: Coordinate Attention Module.

Input: x (tensor of shape (n, c, h, w))
Output: Tensor with coordinate attention applied
procedure INIT($inp, oup, reduction$)
Define pooling: $pool\_h$, $pool\_w$
Compute intermediate channels: $mip \leftarrow \max(8, inp/reduction)$
Define conv layers: $conv1$, $conv\_h$, $conv\_w$
Add BatchNorm and activation: $bn1$, h_swish
procedure FORWARD(x)
Compute $x\_h \leftarrow pool\_h(x)$, $x\_w \leftarrow$ permute$(pool_w(x))$
$y \leftarrow$ concat$(x_h, x_w,$ dim $= 2) \rightarrow conv1 \rightarrow bn1 \rightarrow h\_swish$
Split y: $x\_h, x\_w \leftarrow$ split$(y, [h, w])$, adjust x_w
Compute attention: $a\_h, a\_w \leftarrow$ sigmoid$(conv_h(x_h))$, sigmoid$(conv_w(x_w))$
Return $identity \times a\_h \times a\_w$

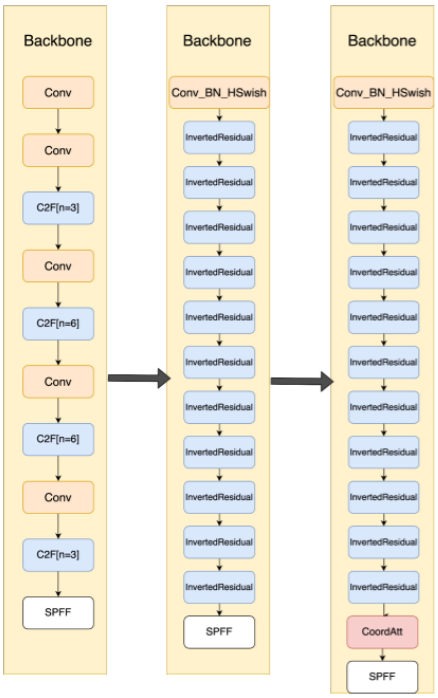

Figure 10: The evolution of Backbone's structures.

4.2.2 Implementation

1.Model Training Pipeline

The training pipeline is designed to optimize the table tennis ball detection model for real-time deployment on a LIMO car. It leverages the strengths of YOLOv8's architecture while replacing the backbone with MobileNetV3 for lightweight and efficient feature extraction.

The backbone of the network (shows in Figure 10) uses MobileNetV3 blocks with integrated Coordinate Attention to extract feature maps at multiple scales. This lightweight backbone replaces YOLOv8's original CSP-Darknet, reducing computational cost while retaining detection accuracy.

The head remains consistent with YOLOv8's original design, ensuring efficient multi-scale feature aggregation

and object detection. It processes the feature maps from the backbone and outputs predictions at three scales: P3 (small), P4 (medium), and P5 (large).

Finally, this training pipeline combines the detection head of YOLOv8 with the lightweight backbone and coordinate-focused benefits of MobileNetV3 to achieve a balance between accuracy and real-time efficiency.

2.Dataset Preparation

We chose the already collected training set "Ping Pong Detection Computer Vision Project" [14]. This dataset contains 9,607 open-source training images and their annotations, which are mainly used for training computer vision models to detect table tennis balls. Using this dataset satisfies the training capacity and requirements, so we decided to use this dataset for the subsequent training.

4.2.3 Results & Analysis

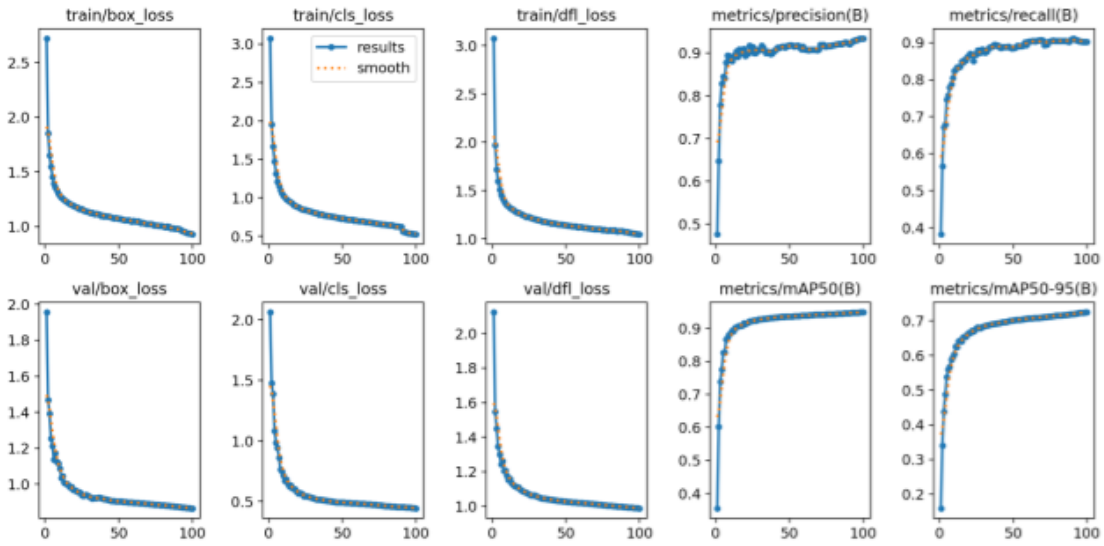

Figure 11: The evolution of Backbone's structures.

Through many training runs, we selected one of the best performing models for testing.

Firstly, Figure 11 provides a visualization of the training and validation performance metrics. As analyzed in the figure, the loss converges and stabilizes quickly, and the training loss and validation loss are very close to each other. The mAP metric highlights the model's ability to accurately detect objects at different scales and IoU thresholds, with mAP50 reaching $\tilde{0}$.93 and mAP50-95 stabilizing at $\tilde{0}$.80. These findings demonstrate the efficacy of integrating Coordinate Attention with the MobileNetV3 backbone, which lowers computing complexity while preserving excellent detection accuracy.

Based on the images tested in the dataset (Figure 12), the model performs well in detecting table tennis balls in different scenes. The model's ability to generalize across different environments, its ability to handle multi-target scenes, and its ability to provide high-confidence detection highlights demonstrates its robustness.

After these tests, we decided to use this model applied to LIMO car. Before that, we first uploaded a photo taken from our cell phone for testing, and the model was able to recognize the table tennis ball in the picture very well. We then uploaded this model to the LIMO car as shown in Figure 13, and it was able to recognize it successfully and with a high level of confidence.

While the proposed model effectively integrates MobileNetV3 and Coordinate Attention to enhance table tennis

balls detection, certain limitations arise when deployed on the LIMO car due to real-world challenges like lighting variations, dynamic backgrounds, and recognition speed limitations, which hinder real-time performance and accuracy.

Figure 12: The test detections on the training dataset.

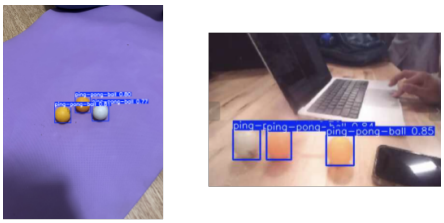

Figure 13: Real world inspection, phone photo detection on the left, LIMO car on the right.

4.3 External Midline Room Coverage

4.3.1 Concentric Polygon Midline

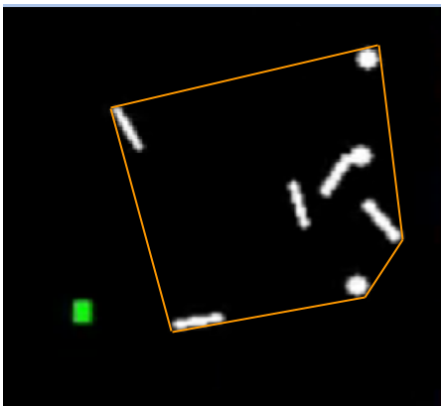

Figure 14: Example convex hull in orange of occupancy map obstacles.

An interesting challenge about real-time table tennis ball collection is the need to avoid the direct play area during live play unless there is a ball to collect. As such, it is advantageous for Loop to enter the play area as lit-

tle as possible. What this means for Loop is that its coverage method should plan a path between the grid map boundaries and the convex hull formed by detected obstacles as seen in Figure 14.

Figure 15: Adapted from Figure 4 of Niblack et al. [12]. Example centerline skeletonization with (right) and without (left) "bones".

As mentioned in Subsection 2.1.3 on Skeletonization, it is possible to compute the centerline skeleton of a shape beginning by applying the Euclidean Distance Transform (EDT) using OpenCV and finding the peaks points [12]. However, this method would also produce the "bones" of the internal skeleton inside of the convex hull formed by the object detections as can be seen in Figure 15.

We modify this algorithm to compute only the external skeleton by computing the EDT of two images using OpenCV. The first is composed of only the boundary and the second consists of only the obstacles.

4.3.2 Implementation

Our implementation of this utilizes OpenCV's built in functions such as distanceTransform() and findContours(). As the EDT calculates the distance between a white pixel and the nearest black pixel in a binary image, the boundaries and detections in their respective images need to be black. This yields two images that encode the distance to the nearest black pixel in each pixel. The external skeleton is then simply the points in each image where their distance to a black pixel is within some absolute difference.

This produces an inconsistently thick centerline which must be processed to be a single pixel wide to create the list of waypoints for Loop to follow. This can be done by simply gathering the contours present in an image via Suzuki et al.'s border following algorithm implemented in OpenCV [19]. We can then sort the contours by perimeter and discard the two largest contours which correspond to the image boundary and the outer contour of our centerline path leaving our single pixel path as the largest. The outline of this algorithm can be found in Algorithm 4.

Algorithm 4 External Skeletonization of Shapes in Image.

$path\_thickness \leftarrow 1.5$
$boundary \leftarrow boundary\_image$
$obstacles \leftarrow detection\_image$
$edt\_bdy \leftarrow calculate\_EDT(boundary)$
$edt\_obs \leftarrow calculate\_EDT(obstacles)$
$path\_points \leftarrow arg\_where(abs(edt\_bdy - edt\_obs) \leq path\_thickness)$
$path\_image = zeros\_copy\_size(boundary)$
$path\_image = draw\_points(path\_image, path\_points)$
$single\_line\_path \leftarrow extract\_single\_pixel\_wide\_path(path\_image)$

4.3.3 Results & Analysis

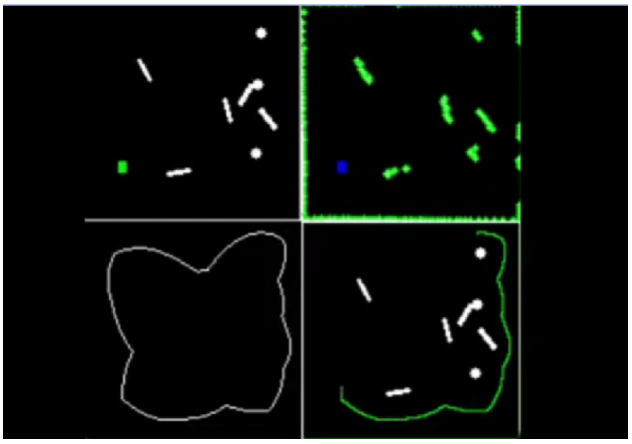

Figure 16: Different view of our simulator environment. In order from top left in clockwise order: Global Map with Loop Agent, LiDAR Occupancy Map with Loop Agent, Global Map with Centerline Path Progress, Centerline Path.

Our algorithm is able to accurately find the centerline between the image boundary and the outside of the obstacles as seen in the bottom left of Figure 16. This method also guarantees margins between obstacles and the simulated Loop Agent as it is theoretically the perfect midline path.

This algorithm also runs quite quickly with an average process time of 31.25 ms on an 128 * 128 pixel image using an Intel i9-13900H CPU. Since this only needs to be calculated if the map changes substantially, this level of computational load is acceptable.

This method, while convenient, unfortunately has a major downside. First, it does not guarantee full environment coverage by the Loop's sensors. This is especially true of spaces within the play area. As such, there may exist parts of the environment that will never be scanned by the Loop's camera creating voids. If a table tennis ball enters these voids, the Loop will never be able to detect and retrieve them. This can be seen in the bottom right of Figure 16 where certain pockets of space surrounded by obstacles are never scanned with the FOV of the RGB-D camera. The only chance for detection and retrieval is if during the retrieval process for a different ball, the void is exposed to the RGB-D camera.

4.4 Deep Q-Learning Path Planning

4.4.1 DQN & D3QN Theory

Deep Q-Learning (DQN) is a natural extension of Q-learning that replaces the Q-table with a neural network (NN) [11]. This is because a NN can be used as an universal function approximator. Unlike the original Q-table approach, the DQN method does not guarantee to converge but promises greater generalization and flexibility. It also uses the Bellman function to propagate rewards and update the Q-values. This equation can be seen in Equation 1 where the policy predicts the target Q-values of the next state. The γ denotes the discount factor which is a mathematical trick to converge infinite horizon problems. As each iteration of the Bellman equation is computed, a γ less than one will diminish the rewards over time. By modifying this hyperparameter, it is possible to change how much the learning process values future rewards. The closer to 0, the more myopic and the closer to 1, the more far-seeing.

To help with convergence, Hasselt et al. proposed using a second target network to form the Double DQN (D2QN) [21]. This necessitates a modification to the Bellman equation which can be seen in Equation 2. A second network called the target network is used to calculate the Q-values of the next state using the actions selected by the policy network. The rationale is that the original Bellman update can overestimate the Q-values by cyclically updating them and then using them to perform actions. This can explode the importance of undesirable actions

and prevent learning the optimal behavior. The target network is setup to be fixed compared to the policy network, providing a reference point.

Wang et al. improves upon the D2QN by separating the learning inside the NN into Value (V) and Advantage (A) streams. The outputs of this these streams are used to calculate the Q-value. The V stream learns the value of the input state without regard to the actions while the A stream learns the advantages of doing actions to the input state. This is useful in situations where multiple actions would have similar Q-values such as our grid world application. If the goal is northeast of the agent, using Manhattan distance would dictate that moving either up or right will yield the same reduction in distance. As such, it can be said in the absence of relevant obstacles, that both of those actions would have the same Q-value. The model's Q-value is thus transformed into Equation 3. By mean-zeroing the advantage scores of each action, actions with similar advantage scores will be closest to zero, lessening their impact on the Q-value.

$$Q(s_t, a_t; \theta) = R_t + \gamma * maxQ(s_{t+1}, a; \theta) \tag{1}$$

$$Q(s_t, a_t; \theta) = R_t + \gamma * Q(s_{t+1}, argmaxQ(s_{t+1}, a; \theta); \theta_{target}) \tag{2}$$

$$Q(s_t, a_t; \theta) = V(s_t) + A(s_t, a) - \frac{1}{n} \sum_{a} A(s_t, a) \tag{3}$$

4.4.2 Implementation

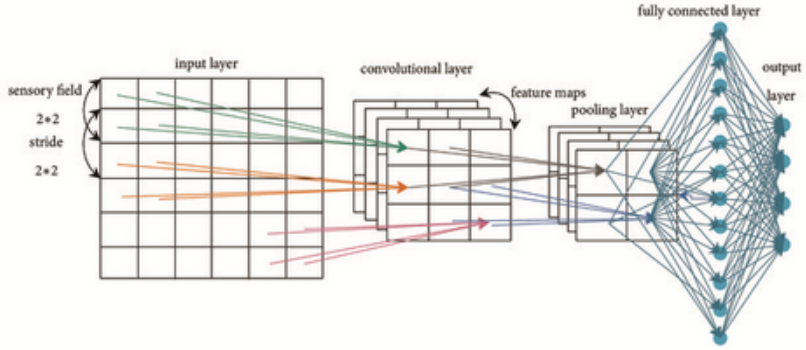

Figure 17: Overall network architecture for our D3QN. Adapted from Figure 2 in Lei et al. [8].

We use a simulated LiDAR environment to train the policy as physical training is too costly as seen in Figure 16 and detailed in Subsection 4.1.2. We adopt a similar method to Lei et al. for training by randomizing our environment states and incrementally increasing the training difficulty over time [8]. This coincides with decreasing the e value of the e-greedy exploration method. To reduce complexity and problem difficulty, the previous 128 * 128 occupancy grid is reduced to a 64 * 64 environment. This encourages a higher ratio of goal successes versus collisions in the replay buffer. To further improve the positive to negative reward ratio, the environment is frozen for 5000 timesteps to allow the D3QN to acclimate.

Another thing adopted from Lei et al. is the overall Convolutional Neural Network (CNN) model architecture which we implemented using PyTorch. They take the 360 LiDAR points and pack them into a 20 * 20 * 2 pseudo-image [8]. The channels carry the relative cartesian coordinates of detections. To pad the point cloud to the correct size, the rest of the pseudo-image is filled with the relative cartesian goal coordinates. This provides the CNN with the necessary information to seek the goal while avoiding obstacles. The CNN contains Convolutional Layers to extract features from the point cloud which are flattened for fully connected layers as can be seen in Figure 17. We then modify it into a Dueling CNN by including the V and A streams. The exact parameters used can be found in Figure 18.

```
----------------------------------------------------------------
       Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1          [-1, 16, 10, 10]             144
          Conv2d-2           [-1, 64, 5, 5]           4,160
          Conv2d-3           [-1, 256, 1, 1]        409,856
         Flatten-4                 [-1, 256]              0
          Linear-5               [-1, 32768]      8,421,376
          Linear-6                [-1, 8192]    268,443,648
          Linear-7                 [-1, 512]      4,194,816
          Linear-8                   [-1, 1]            513
          Linear-9                   [-1, 4]          2,052
================================================================
Total params: 281,476,565
Trainable params: 281,476,565
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.34
Params size (MB): 1073.75
Estimated Total Size (MB): 1074.10
----------------------------------------------------------------
```

Hyperparameter	Value
T-Steps	1,000,000
Buffer Size	8,000,000
Exp. Fraction	0.3
Exp. Final e	0.15
n-steps	3
Batch Size	512
Gamma	0.999
Hard Update Frequency	50,000

Figure 18: The left image shows model architecture and summary of our D3QN and the right shows the training hyperparamters used for both the DQN and D3QN.

4.4.3 Results & Analysis

$$R = \begin{cases} r = -1.0, & \text{if } collided. \\ r = max(1.0, init\_man\_dist), & \text{if } goal\_reached. \\ r = min(1.0, init\_man\_dist \div (cur\_man\_dist + 1)) * 0.02, & \text{if } cur\_man\_dist < prev\_man\_dist. \\ r = -0.02, & \text{if } cur\_man\_dist \geq prev\_man\_dist. \end{cases} \tag{4}$$

$$R = \begin{cases} r = min(-1.0, -init\_man\_dist), & \text{if } collided. \\ r = max(1.0, init\_man\_dist), & \text{if } goal\_reached. \\ r = min(1.0, init\_man\_dist \div (cur\_man\_dist + 1)) * 0.02, & \text{if } cur\_man\_dist < prev\_man\_dist. \\ r = -0.02, & \text{if } cur\_man\_dist \geq prev\_man\_dist. \end{cases} \tag{5}$$

The main point of comparison is the performance difference between the base DQN algorithm and D3QN. As can be seen in Figure 19, the base DQN is unable to learn an effective policy with an average episode reward of around 5.0. The reward function for the base DQN can be seen in Equation 4 and the reward function for the D3QN can be found in Equation 5. The difference in reward functions was a mistake during training and skews the episode rewards for the DQN higher than they should be. The actual average episode rewards should be negative due to the large amount of collisions and time-outs.

On the other hand, the D3QN implementation is able to learn an effective policy as evidenced by reaching an average episode reward of around 15.0 as compared to the max reward of 20.0 as seen in Figure 20. Additionally, the D3QN was able to converge its performance compared to the DQN at around 250,000 timesteps versus no convergence. When evaluated on completely new environments, the DQN received an average reward of -0.110. The D3QN achieved an average reward of 24.382, a marked improvement that was expected.

Using our Deep Q-Learning implementation for path planning contains some benefits. The first is that the policy is theoretically grid world size independent. This is due to the relative coordinate input to the CNN. Additionally, it should not matter if the goal point is dynamic as the CNN only knows the relative distance and does not require a sequence of states as an input. A-star must recalculate the path each time the environment changes such as when a previously occluded obstacle appears or if the goal position changes. This can be costly to recalculate every time the state changes while the CNN's computation time per frame is relatively static.

However, Deep Q-Learning has some limitations for this application. First is learning an optimal policy. Despite

the promising performance of the D3QN, it still learns to "jitter" in place in some states. This is especially apparent when nearing the goal position and requires early stopping to transition to a "last-foot" path planning module. Another limitation is the use of a CNN which takes substantial computing resources. While the Nvidia Jetson Orin Nano is capable of AI tasks, we are also using the MobileNetV3 + YOLO CNN for object detection. Scheduling the usage of the chip between the two networks requires extra integration and resource management. A final risk is that this method of path planning is a complete black box. It is difficult to interpret exactly what the policy learned during training and what might happen in novel situations. This makes it difficult to iterate and debug its behavior.

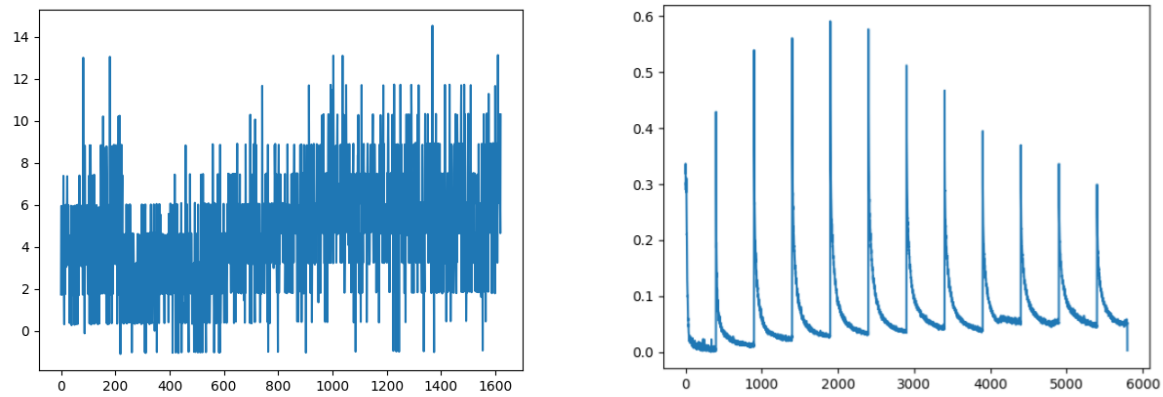

Figure 19: Training episode rewards and loss using basic DQN. Collision punishment was -1.0 and goal reward was scaled by original goal distance. Consequently, reward graph is skewed positive.

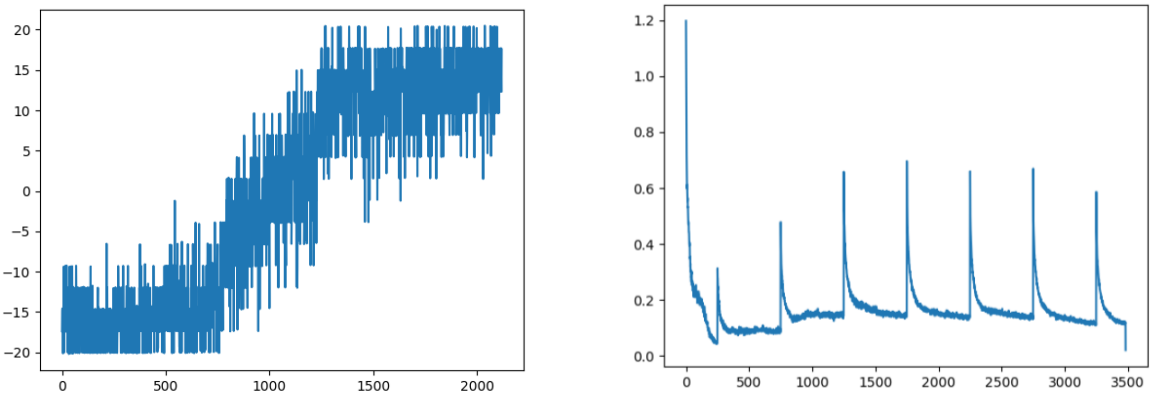

Figure 20: Training episode rewards and loss using D3QN. Collision punishment was the negative of the goal reward and the goal reward was scaled by original goal distance.

4.5 AprilTag Informed Delivery Marking

4.5.1 Why Use AprilTags?

AprilTags are widely used in many different applications such as robotics, computer vision, and camera calibration. In our project, we implement AprilTags as a delivery goal to deposit the collected loose table tennis balls. For robust detection, AprilTags work well under challenging conditions, like low, glare, steep, angles, and motion blur,

ensuring reliable performance. AprilTags' fast detection operates in real-time, which is crucial for autonomous systems and interactive applications. Additionally, AprilTags provide a precise 6-Degrees of Freedom pose estimation, making them ideal for use in robotics (such as the AgileX LIMO Pro), drones, and AR Applications. Another benefit is scalability and error correction by using the Strong Hamming Error Correction Algorithm and the pattern supports thousands of unique tags. This makes AprilTags scalable for large tasks like warehouse mapping or multi-robot coordination. AprilTags are open-source, free to use, easily customized, and can be easily integrated into popular computer vision frameworks like OpenCV and are compatible with platforms such as Python, C++, and ROS.

4.5.2 AprilTag Theory

AprilTag detection begins by preprocessing the image to detect edges and locate potential quadrilateral markers through outline detection. Once a quadrilateral is found, a perspective transformation, also known as the homography transform, normalizes the tag's view, correcting for the camera angle and distortion. This is done using the homography matrix where **x' = Hx**. The formula **x' = Hx** represents the relationship between the original points **x** and their transformed counterparts **x'** in a normalized reference. Our **H** in this case, is a 3x3 homography matrix that encodes transformations such as scaling, rotations, translation, and perspective distortion. **x** in the original point in homogeneous coordinates $[x, y, 1]^T$. For our **x**, it is a 3x1 column vector in homogeneous coordinates, representing a point in the image plane as $[x, y, 1]^T$. For our **x'**, it is a 3x1 column vector in homogeneous coordinates, which represents the transformed point after applying **H**.

The transformation aligns with the tag's perspective in the image with a flat, idealized view, enabling accurate decoding and pose estimation. When you multiply **H** and **x**, you're applying the transformation defined by **H** to **x**, producing the new point **x'**. This operation aligns the distorted view of the tag in the camera image to a flat, normalized reference frame for further processing. Here, **x** are the corner points in the image and **x'** are the transformed points in a reference frame. The tag's grid is then extracted with each cell decoded into a binary sequence. Error corrections, such as Hamming Code Correction, are applied to handle noise or occlusion, ensuring accurate decoding. The final part output is the 6-Degrees-of-Freedom pose estimation, giving precise position and orientation relative to the camera.

4.5.3 AprilTag Implementation

We utilized an open-source AprilTag library from GitHub in our code [2]. Using the RGB-D camera, we can determine the transformations of the AprilTag and calculate the relative position between the vehicle and the AprilTag. Based on this position, we employ our D3QN module to path plan from the current vehicle position to the AprilTag.

4.5.4 Results & Analysis

For our results, Figure 21 contains the data collected from two of the AprilTag examples. As can be seen, accurate 3-D transformation and rotation information can be captured from the AprilTags in different lighting and orientations. Additionally, the run-time of the open-source algorithm was around 20 ms on a AMD Ryzen 7 7730U which enables real-time detection and processing.

The open-source AprilTag system is a widely used tool for visual marker detection, but it has notable limitations. Its performance is sensitive to environmental factors such as lighting, motion blur, and background clutter. It also struggles with small, occluded, or highly angled tags, as well as those that are damaged or poorly printed. Pose estimation accuracy relies heavily on precise camera calibration, and the absence of adaptive mechanisms reduces robustness in varied conditions. While the system's open-source nature allows for customization, integrating or optimizing it for specialized pipelines and hardware often requires significant expertise and effort.

Tag Family	Tag ID	Hamming	Decision Margin	Centerpoint	Translation-3D	Rotation 3D
tag16h5	0	1	44.82143021	x=679.0770427167619, y=400.3197454466794	x=0.333470, y=0.150775, z=0.478404	x = 0.286693, y = 0.411858, z = -0.148987
tag36h11	1	0	162.4381256	x=290.2841159722061, y=217.07593833520406	x=-0.110673, y=-0.085384, z=1.862207	x=-0.021993, y=-0.120889, z=-0.021934

Figure 21: Example AprilTag detections with transform between camera and tag.

Figure 22: AprilTag Example 1 - Tag ID: 0.

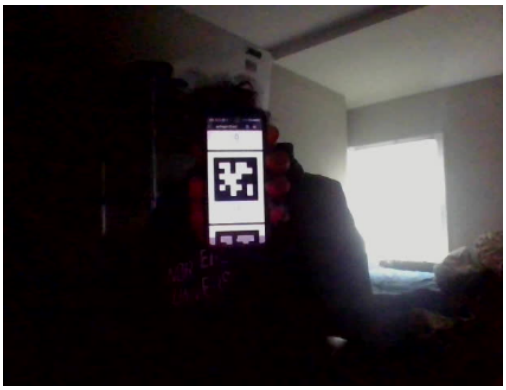

Figure 23: AprilTag Example 2 - Tag ID: 1.

5 Conclusion

In this project, we developed an intelligent robotic system integrating SLAM, object detection, deep reinforcement learning path planning, and AprilTag navigation for table tennis collection tasks in dynamic environments. Through the 2-D LiDAR SLAM module, the system can build high-precision environment maps in real-time and realize reliable robot positioning. The object detection module uses deep learning models to accurately identify table tennis balls, and the path planning module effectively realizes the dynamic navigation of target points through deep reinforcement learning. The AprilTag navigation function ensures that the robot can accurately identify the storage point and complete the collection and delivery task.

Nevertheless, the system still has some room for improvement when dealing with dynamic environments and high-density target tasks. Future work will further optimize the system performance and expand its application scenarios.

5.1 Future Work

Future research will focus on improving the adaptability and functional diversity of the system to meet the needs of more complex environments. Multi-sensor data fusion is an important research direction, combining LiDAR and RGB-D camera data, and will further improve Loop's perception ability in dynamic environments and the robustness of its SLAM module. Semantic segmentation of sensor data using a deep learning model can more accurately identify and filter the interference of dynamic objects to update the map.

In terms of path planning, more powerful deep reinforcement learning algorithms such Proximal Policy Optimization or Soft Actor Critic improve performance. Additionally, the introduction of multiple Loops can be researched using multi-agent-based collaborative planning methods. This enables robots to achieve efficient allocation and execution in multi-objective tasks. This will not only help improve the efficiency of a single robot tasks, but will also provide the basis for the collaboration of multi-robot systems.

Improvements at the hardware level will also be the focus of future work. An addition of a dedicated edge tensor processing unit could improve deep neural network inference speed and increase the number of models that can be run on the Loop in parallel. At the same time, through lightweight design and energy consumption optimization, the robot's endurance in long-term tasks can be improved. Finally, migrating the system from the simulation environment to the actual application scenario is the key step to test its practicability.

References

[1] M. Betke and L. Gurvits. Mobile robot localization using landmarks. *IEEE Transactions on Robotics and Automation*, 13(2):251–263, 1997.

[2] RobotPy Contributors. Robotpy apriltag tests, 2024. Accessed: 2024-12-13.

[3] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, 2016.

[4] Qibin Hou, Daquan Zhou, and Jiashi Feng. Coordinate attention for efficient mobile network design. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13713–13722, 2021.

[5] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[6] Jan Kallwies, Bianca Forkel, and Hans-Joachim Wuensche. Determining and improving the localization accuracy of apriltag detection. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8288–8294, 2020.

[7] Jeong-ah Kim, Ju-Yeong Sung, and Se-ho Park. Comparison of faster-rcnn, yolo, and ssd for real-time vehicle type recognition. In *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*, pages 1–4, 2020.

[8] Xiaoyun Lei, Zhian Zhang, and Peifang Dong. Dynamic path planning of unknown environment based on deep reinforcement learning. *Journal of Robotics*, 2018(1):5781591, 2018.

[9] Ruijun Liu, Xiangshang Wang, Zhang Chen, and Bohua Zhang. A survey on visual slam based on deep learning. *Journal of System Simulation*, 32(7):1244–1256, 2020.

[10] Ivan Maurović, Marija Seder, Kruno Lenac, and Ivan Petrović. Path planning for active slam based on the d* algorithm with negative edge weights. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(8):1321–1331, 2017.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[12] C.W. Niblack, D.W. Capson, and P.B. Gibbons. Generating skeletons and centerlines from the medial axis transform. In *[1990] Proceedings. 10th International Conference on Pattern Recognition*, volume i, pages 881–885 vol.1, 1990.

[13] Juan Nieto, Tim Bailey, and Eduardo Nebot. Recursive scan-matching slam. *Robotics and Autonomous systems*, 55(1):39–49, 2007.

[14] pingpong.

[15] Echo Robotics.

[16] Relox Robotics.

[17] Indonesia Sentinal.

[18] Jean Serra. *Image Analysis and Mathematical Morphology*. Academic Press, 1982.

[19] Satoshi Suzuki and KeiichiA be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[20] Ultralytics. Yolov8, Nov 2024.

[21] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[22] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2016.

[23] Yan Yin, Zhiyu Chen, Gang Liu, and Jianwei Guo. A mapless local path planning approach using deep reinforcement learning framework. *Sensors*, 23(4), 2023.