北京郵電大學

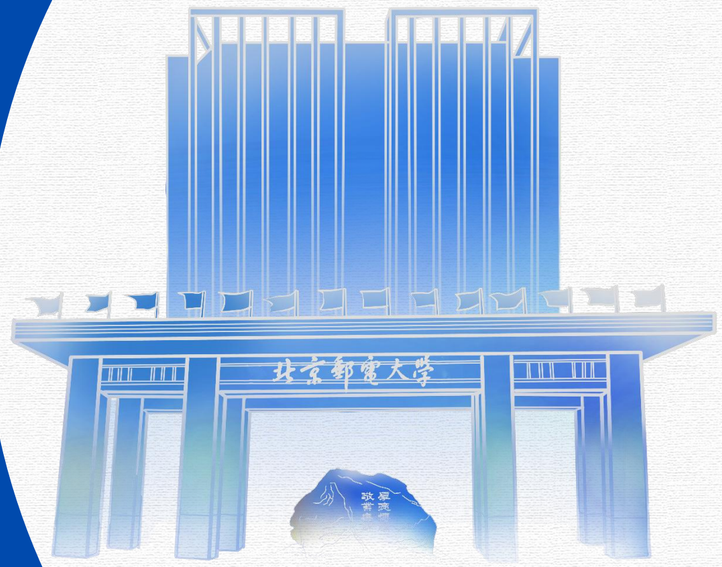
Beijing University of Posts and Telecommunications

BUPT

Design&Build

Project Report

2025.10.23



contents

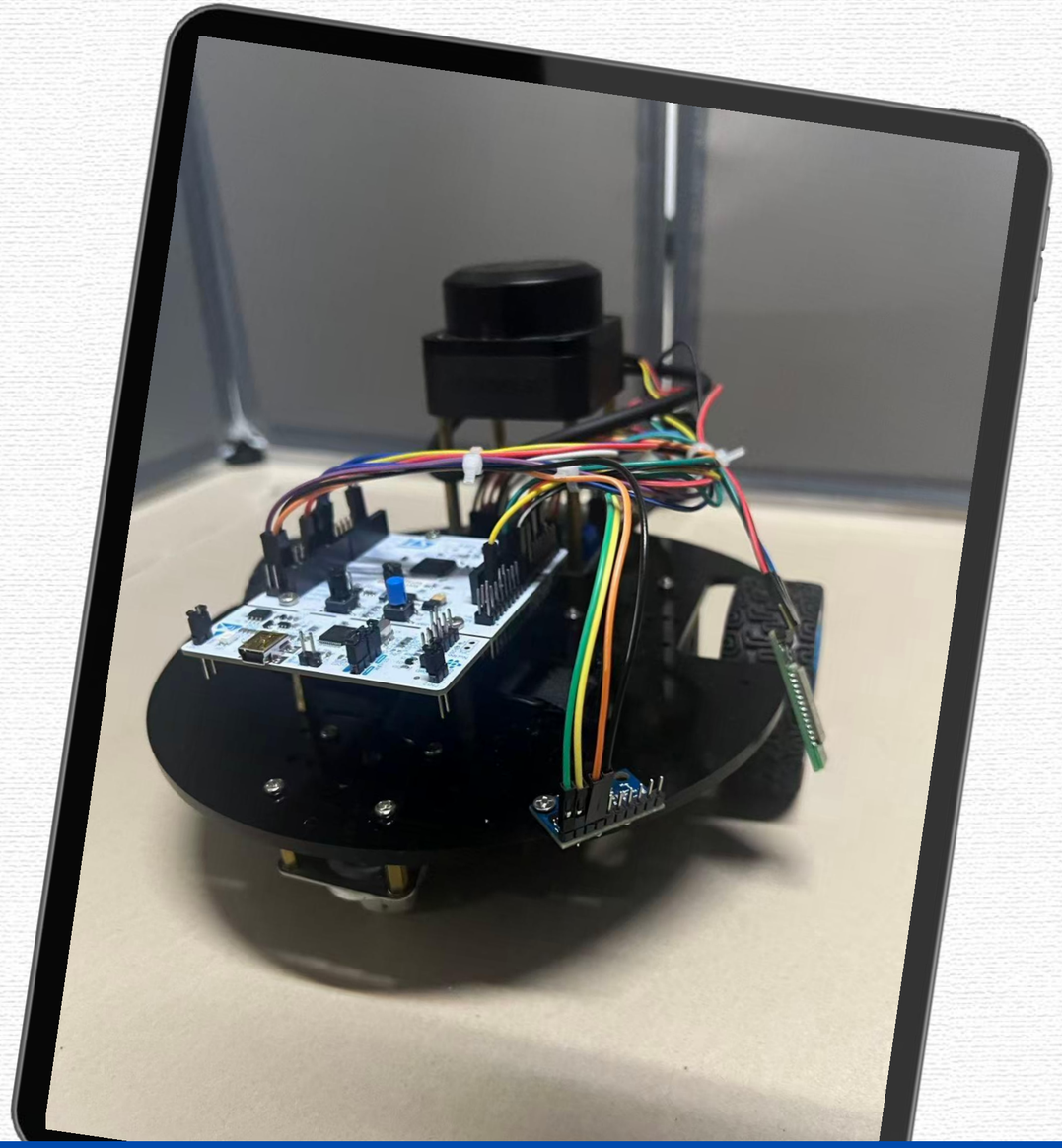
01 Project background and objectives

02 Introduction of the team

03 technical detail

04 function realization

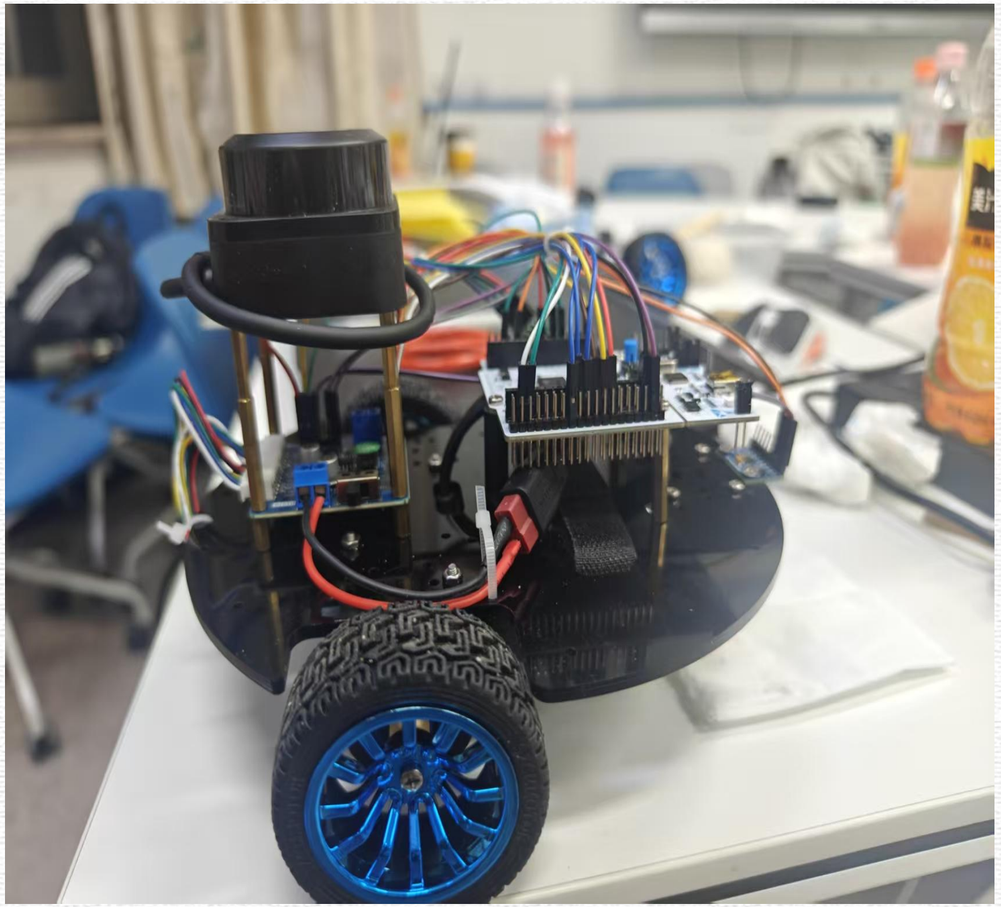
1. Project background and objectives



Project Objectives

This project focuses on the autonomous maze exploration robot as the core research and implementation object. The aim is to achieve the robot's autonomous mapping, path planning and return control in an unknown environment through the collaboration of software and hardware.

1. Project background and objectives



2. Introduction of the team

Firmware:

Hu Weize

Focus:LIDAR; IMU layout and shock absorption; I²C pull-up resistor and line length control.Complete the communication protocol.

Ang Boyu

Focus:Encoder level and filtering;bluetooth connection;Construction of PID algorithm;hardware debug; Interface alignment.

Shen Yuqing

Focus:Debugging hardware,video clip,Refine the code,interface connection.

Sun Yicheng

Focus: Complete the report and build the intelligent vehicle.

Sun Tiancheng

Focus:Construction and operation debugging of the car model.

Li Yuchun

Focus:Construction of the car's exterior design.

Software:

He Mengyang

Focus: Data Pre-processing, System Integration and Final Report.

Liu Xuanchen

Focus: SLAM Pipeline Design and Global Path Planning.

Qin Ruimin

Focus: Frontier Detection and Goal-Oriented Selection.

Lyu Hanxin

Focus: Bluetooth Communication and User Interface Development.

3. technical detail （Firmware）

LiDAR：

As the main environmental perception unit of the system, communicates with the main control unit via the USART6 serial port. The data acquisition adopts a "interrupt-triggered + DMA two-stage structure": first, a 7-byte frame header is received, and then 2600 bytes of scanning data (corresponding to 520 point cloud samples) are efficiently transmitted in DMA mode. The radar data is uniformly parsed and uploaded by the TIM1 interrupt scheduler to ensure the balance between real-time performance and bus occupancy.

code:

```
    else if (huart->Instance == USART6) {
        if (g_lidar_data_ready_flag == 0)
        {
            if (g_lidar_rx_header_idx < LIDAR_HEADER_SIZE)
            {
                g_lidar_header_buffer[g_lidar_rx_header_idx++] = g_lidar_rx_byte;
                if(g_lidar_rx_header_idx == LIDAR_HEADER_SIZE)
                {
                    HAL_UART_Receive_DMA(&huart6, g_lidar_raw_buffer, LIDAR_PAYLOAD_SIZE);
                }
                {
                    HAL_UART_Receive_IT(&huart6, &g_lidar_rx_byte, 1);
                }
            }.
            else
            {
                g_lidar_data_ready_flag = 1;
                g_lidar_rx_header_idx = 0;
                HAL_UART_Receive_IT(&huart6, &g_lidar_rx_byte, 1);
            }
        }
    }
}
```

test output：

```
========================================
--- [雷达帧 LidarFrame] ---
  收到了 520 个雷达点。
  [build_scan360] 已生成 360 个点的完整扫描。
  预览前 5 个点:
    #00  Q= 78  Angle= 189.34°  Dist= 1460.75 mm
    #01  Q=102  Angle= 190.06°  Dist= 1462.50 mm
    #02  Q= 82  Angle= 190.77°  Dist= 1468.50 mm
    #03  Q= 78  Angle= 191.48°  Dist= 1480.50 mm
    #04  Q= 78  Angle= 192.31°  Dist= 1489.50 mm
```

3. technical detail （Firmware）

IMU (Inertial Measurement Unit)：

Accessed through the I2C1 bus, it is used to measure the three-axis acceleration and angular velocity information. After power-on, the system automatically performs 500 zero bias calibrations to eliminate the gyroscope drift. The main controller calculates the angular velocity integral within a 20 ms sampling period and updates the heading angle yaw to achieve continuous estimation of vehicle attitude and steering control.

code:

```c
void MPU6500_Init(I2C_HandleTypeDef* hi2c) {
    uint8_t check, data;
    HAL_I2C_Mem_Read(hi2c, MPU6500_ADDR, WHO_AM_I, 1, &check, 1, 100);
    if (check != 0x70) { Error_Handler(); }
    data = 0x00; HAL_I2C_Mem_Write(hi2c, MPU6500_ADDR, PWR_MGMT_1, 1, &data, 1, 100); HAL_Delay(100);
    data = GYRO_FS_SEL_500DPS; HAL_I2C_Mem_Write(hi2c, MPU6500_ADDR, GYRO_CONFIG, 1, &data, 1, 100);
    data = 0x08; HAL_I2C_Mem_Write(hi2c, MPU6500_ADDR, ACCEL_CONFIG, 1, &data, 1, 100);
}
```

test output:

```
========================================
--- [状态帧 StatusFrame] ---
  Timestamp: 42740000 us
  Status: 0 (停止(完成))
  Encoders: L=-608, R=898
  Yaw: 136.176°
  [里程计] dXY=0.00 mm | dθ=0.000° | dt=0.0200 s

========================================
--- [状态帧 StatusFrame] ---
  Timestamp: 42760000 us
  Status: 0 (停止(完成))
  Encoders: L=-608, R=898
  Yaw: 136.173°
  [里程计] dXY=0.00 mm | dθ=0.000° | dt=0.0200 s

========================================
```

3. technical detail （Firmware）

Encoder feedback module：

The left and right wheel encoders are respectively connected to the encoder interfaces of TIM2 and TIM4, and continuously read the rotational pulse counts. Combined with the wheel diameter (32.5 mm) and the resolution of 770 pulses per revolution, the system can accurately calculate the mileage and angle deviation, which is used for straight-line correction and closed-loop control.

code:

```
    if (g_target_distance_m != 0.0f) {
        g_car_status = STATUS_MOVING_STRAIGHT;
        float target_revolutions = fabsf(g_target_distance_m) / WHEEL_CIRCUMFERENCE_M;
        int32_t target_ticks = (int32_t)(target_revolutions * TICKS_PER_REV);
        int32_t current_ticks_l = abs((int32_t)__HAL_TIM_GET_COUNTER(&htim2) - g_start_encoder_l);
        int32_t current_ticks_r = abs((int32_t)__HAL_TIM_GET_COUNTER(&htim4) - g_start_encoder_r);
        int32_t avg_ticks = (current_ticks_l + current_ticks_r) / 2;

        if (avg_ticks < target_ticks) {
            if (g_target_distance_m > 0) {
                ControlMotor(DIR_FORWARD, MAX_SPEED, MAX_SPEED);
            } else {
                ControlMotor(DIR_BACKWARD, MAX_SPEED, MAX_SPEED);
            }
        } else {
            g_target_distance_m = 0.0f;
            ControlMotor(DIR_STOP, 0, 0);
            g_car_status = STATUS_STOPPED;
        }
    }
}
```

test output：

```
==========================================
--- [状态帧 StatusFrame] ---
  Timestamp: 26040000 us
  Status: 0 (停止(完成))
  Encoders: L=15, R=65528
  Yaw: 309.707°
  [里程计] dXY=0.00 mm | dθ=0.000° | dt=0.0600 s
```

3. technical detail （Firmware）

PID：

In this project, PID control is applied to the speed and steering control of the trolley: the proportional term (P) enables the trolley to respond quickly to errors, the integral term (I) eliminates long-term deviations, ensuring the trolley accurately stops at the target position, and the differential term (D) predicts the trend of error changes, suppressing overshoot and oscillation. Through the PID algorithm, the car can maintain smooth driving and precise steering under different ground and load conditions, achieving higher motion stability and control accuracy

code:

```
if (1) { // Bypassing CRC check for now
    memcpy(&g_current_cmd_id, &g_downlink_buffer[4], 2);
    g_target_turn_rad = 0.0f;
    g_target_distance_m = 0.0f;
    ControlMotor(DIR_STOP, 0, 0);
    float turn_rad, distance_m;
    memcpy(&turn_rad, &g_downlink_buffer[6], 4);
    memcpy(&distance_m, &g_downlink_buffer[10], 4);
    g_start_encoder_l = (int32_t)__HAL_TIM_GET_COUNTER(&htim2);
    g_start_encoder_r = (int32_t)__HAL_TIM_GET_COUNTER(&htim4);
    g_start_yaw = yaw;
    if (fabsf(distance_m) > 0.001f) {
        g_target_distance_m = distance_m;
        g_target_turn_rad = 0.0f;
    } else if (fabsf(turn_rad) > 0.001f) {
        g_target_turn_rad = turn_rad;
        g_target_distance_m = 0.0f;
        float target_angle_deg = g_target_turn_rad * 180.0f / 3.14159265f;

        // 1. 初始化PID参数 (Kp, Ki, Kd 是关键，需要您后期调试)
        //    输出限幅为电机的最大PWM值，可以防止输出过大
        PID_Init(&g_turn_pid, 40.0f, 2.0f, 25.0f, -TURN_SPEED, TURN_SPEED);

        // 2. 设置PID的目标值 (Setpoint)
        //    注意：我们使用相对角度，所以目标是转过的角度，而不是绝对的yaw值
        PID_Set_Setpoint(&g_turn_pid, target_angle_deg);
    } else {
        g_car_status = STATUS_STOPPED;
    }
    memmove(&g_downlink_buffer[0], &g_downlink_buffer[16], g_downlink_buffer_idx - 16);
    g_downlink_buffer_idx -= 16;
    return;
} else {
    memmove(&g_downlink_buffer[0], &g_downlink_buffer[1], g_downlink_buffer_idx - 1);
    g_downlink_buffer_idx--;
}
```

3. technical detail （Software）

Communication frame & decoding：

Implements a robust, framed serial protocol over Bluetooth. The PC transmits motion commands (turn radius + straight distance) with a Modbus CRC16; the MCU streams back timestamped telemetry: encoders, IMU yaw, and raw LiDAR points (quality, angle_q6, distance_q2). The PC side reassembles frames, validates lengths, converts units, and exposes simple APIs to “get latest frame” and “send command”.

Downstream transmission (PC → STM32)

```python
def send_cmd(self, cmd_id: int, turn_rad: float, distance_m: float):
    """
    发送命令帧：
      AA 55 | len=10 | cmd_id(u16) | turn_rad(f32) | distance_m(f32) | CRC16(Modbus)
    """
    payload = struct.pack('<Hff', int(cmd_id), float(turn_rad), float(distance_m))
    body_wo_crc = HEADER_DOWN + struct.pack('<H', 10) + payload
    crc = struct.pack('<H', crc16_modbus(body_wo_crc))
    frame = body_wo_crc + crc
    self._write(frame)
    self.log.debug(f"send_cmd: id={cmd_id} turn={turn_rad:.4f}rad dist={distance_m:.4f}m")
```

Upstream frame extraction (STM32 → PC)

```python
def get_frame(self, timeout: Optional[float] = None) -> Optional[UpFrame]:
    """FIFO 取一帧（最旧帧）。"""
    try:
        return self._queue.get(timeout=timeout)
    except Empty:
        return None

def get_latest_frame(self) -> Optional[UpFrame]:
    """只取最新一帧（丢弃队列里较旧的）。"""
    try:
        frm = self._queue.get(timeout=0.0)
    except Empty:
        return None
    while True:
        try:
            frm = self._queue.get_nowait()
        except Empty:
            break
    return frm
```

3. technical detail （Software）

SLAM input assembly:

Tool: BreezySLAM Library

Data Pre-processing：

Noise Reduction: Gaussian Filtering to eliminate isolated noise points.

Outlier Rejection: Filtering based on distance and intensity thresholds.

Data Packaging: Converting processed data into the required SLAM format。

Core SLAM Update：

Directly call BreezySLAM functions to perform pose estimation and update the Occupancy Grid Map.

code:

```python
def create_360_scan_map(sample_list: List[ScanSample],
                        max_range_mm: int = 8000,
                        quality_min: int = 0,
                        do_interpolate: bool = False, # Interpolation logic not in original, keeping param
                        do_median3: bool = False) -> List[int]: # Median logic not in original
    """
    Maps {angle_deg, distance_mm} points to a 0..359 distance array (mm)
    - Takes the minimum distance for duplicate angles (rounded to nearest int)
    - Missing angles are handled (original used 10, here we use 0 or max_range)
    """
    # Step 1: Aggregate minimum distances, using None for missing
    distance_bins: List[Optional[int]] = [None] * 360
    for p in sample_list:
        if p.signal_strength < quality_min:
            continue

        angle = p.angle_degrees % 360.0
        index = int(round(angle)) % 360
        distance = int(max(0, min(65535, round(p.distance_millimeters))))

        if distance <= 0:
            continue

        if distance_bins[index] is None or distance < distance_bins[index]:
            distance_bins[index] = distance

    # If all are missing, return max range
    if all(v is None for v in distance_bins):
        print("All scan points missing")
        return [max_range_mm] * 360

    # Fill in Nones with a default (original used 10, let's use max_range_mm)
    scan_cw = [int(v) if v is not None else max_range_mm for v in distance_bins]

    # (Interpolation/Median filtering would go here if enabled)

    # Reorder to match SLAM coordinate system
    return remap_scan_to_centric(scan_cw)
```

3. technical detail （Software）

SLAM map robustness:

Strategy: Multi-Frame Voting Mechanism.

Concept: For each map cell, a "vote" is conducted over multiple consecutive frames to confirm its status.

State Locking: Once a cell's state is statistically confirmed, it is locked and skipped in future checks, ensuring stability against temporary sensor fluctuations.

Post-processing

Morphological Filtering: Applied to the final map to smooth wall contours and eliminate small, spurious noise regions.

code:

```python
def process_packet_for_slam(previous_state: Optional[Tuple[int, int, int]],
                            packet: InboundPacket,
                            geom: VehiclePhysics,
                            max_range_mm: int = 8000,
                            quality_min: int = 0
                           ) -> Tuple[List[int], Tuple[float, float, float], Tuple[int, int, int]]:
    """
    Converts one InboundPacket to:
      scan360_mm, (dxy_mm, dtheta_deg, dt_s), current_state
    For use with BreezySLAM.update(scan, velocities)
    """
    scan360 = create_360_scan_map(packet.scan_points,
                                  max_range_mm=max_range_mm,
                                  quality_min=quality_min)

    current_state = (packet.encoder_left_ticks, packet.encoder_right_ticks, packet.timestamp_micros)

    if previous_state is None:
        motion_deltas = (0.0, 0.0, 1e-6) # No motion, tiny dt
    else:
        motion_deltas = calculate_odometry(previous_state, current_state, geom)

    return scan360, motion_deltas, current_state
```

code:

```python
def ProcessSlamLayer(self, mapbytes, origin_lower_left=True):
    """
    接收 SLAM 算法的原始地图数据，并更新自己的“投票”和“锁定”状态。
    """
    slam_map = np.reshape(np.frombuffer(mapbytes, dtype=np.uint8),
                          (self.map_size_pixels, self.map_size_pixels))

    self._tally_sensor_data(slam_map)

    self._evaluate_committed_cells()

    final_map = self._build_display_matrix(slam_map)

    self.grid = final_map.tolist()
```

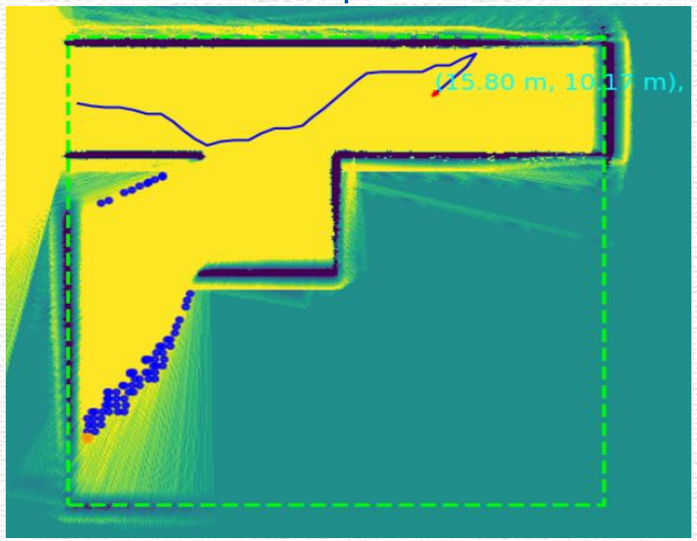
3. technical detail （Software）

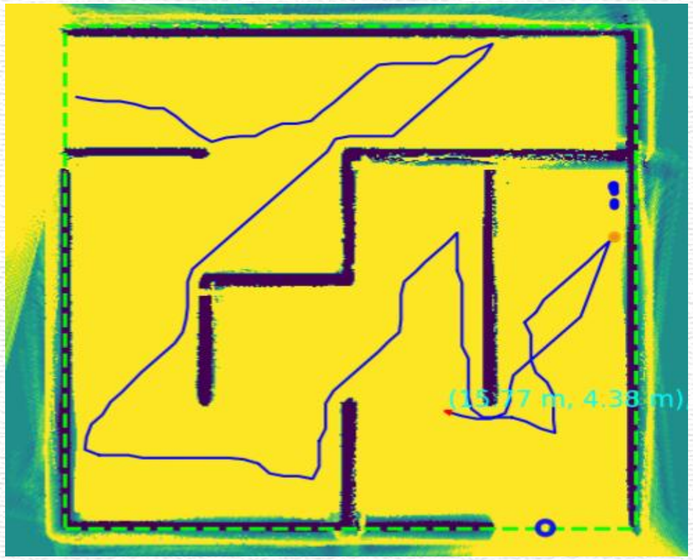
SLAM output：

The figures above demonstrate the real-time mapping and navigation results achieved by our SLAM system.

In both environments, the yellow region represents free space reconstructed from LiDAR scans, while the dark boundaries correspond to obstacles and walls. The blue trajectory traces the robot’s estimated path as it autonomously explores the unknown area, and the green dashed rectangle outlines the dynamically detected region of interest (ROI) used for efficient frontier-based planning.

output：





3. technical detail （Software）

Frontier extraction & goal biasing (exploration):

Detects frontiers—free cells whose neighborhood touches unknown—while enforcing a safety clearance from obstacles. Then, from all frontier candidates, applies a 4-stage selection pipeline:

Distance filter (ignore too close / too far from the robot).

Spatial sampling (grid sampling to keep diversity & reduce cost).

Connectivity filter (BFS from robot pose on a thick free mask to keep only reachable candidates; includes a fallback looser test if nothing survives).

Goal-directed bias (if a global exit/goal is known, prefer frontiers closer to it and directionally aligned).

code:

```python
def _filter_by_goal_proximity(self,
                              candidates: List[Tuple[int, int]],
                              curr_pix: Tuple[int, int],
                              goal_pix: Tuple[int, int]) -> List[Tuple[int, int]]:

    if not candidates:
        return []

    cx, cy = curr_pix
    gx, gy = goal_pix

    # 当前位置到终点的向量
    goal_vec = (gx - cx, gy - cy)
    goal_dist = math.sqrt(goal_vec[0] ** 2 + goal_vec[1] ** 2)

    if goal_dist < 1e-6:
        # 当前位置已接近终点，直接返回原列表
        return candidates

    # 归一化方向向量
    goal_dir = (goal_vec[0] / goal_dist, goal_vec[1] / goal_dist)

    scored_candidates = []

    for px, py in candidates:
        # 候选点到终点的距离
        dist_to_goal = math.sqrt((px - gx) ** 2 + (py - gy) ** 2)

        # 当前位置→候选点的向量
        cand_vec = (px - cx, py - cy)
        cand_dist = math.sqrt(cand_vec[0] ** 2 + cand_vec[1] ** 2)

        if cand_dist < 1e-6:
            continue

        # 方向对齐度（点积，范围[-1, 1]，1表示完全朝向终点）
        alignment = (cand_vec[0] * goal_dir[0] + cand_vec[1] * goal_dir[1]) / cand_dist

        # 综合得分：
        # - 距离终点越近越好（归一化到[0,1]）
        # - 方向越对齐越好（归一化到[0,1]）
        # - 使用可配置的权重平衡
        distance_score = 1.0 / (1.0 + dist_to_goal / 100.0)  # 距离得分
        alignment_score = (alignment + 1.0) / 2.0  # 对齐得分：[-1,1] → [0,1]

        # 综合得分（权重可调）
        final_score = (self.goal_bias_weight * distance_score +
                       (1 - self.goal_bias_weight) * alignment_score)

        scored_candidates.append(((px, py), final_score))

    # 按得分降序排序
    scored_candidates.sort(key=lambda x: -x[1])

    # 返回排序后的坐标列表
    result = [coord for coord, _ in scored_candidates]

    # MODIFICATION: 检查是否有候选点，避免在空列表上索引 [0]
    if scored_candidates:
        print(f"终点导向筛选: {len(candidates)}个候选点, "
              f"最佳得分={scored_candidates[0][1]:.3f}")
    else:
        print(f"终点导向筛选: {len(candidates)}个候选点, 没有有效得分")

    return result
```

3. technical detail （Software）

Path planning & path simplification (global):

Grid-based Autonomous Exploration Detection Algorithm Conditions:

Free Space Check: The cell must be categorized as free.

Unknown Neighbor Check: The cell must have at least one unknown neighbor (in 8-neighborhood).

Safety Distance Check: A circular region around the cell must be clear of obstacles.

Optimization Strategies:

Vectorized Detection: Utilizing NumPy/SciPy for accelerated, non-pixel-by-pixel processing.

Incremental Update: Only re-evaluating regions of the map that have recently changed.

code:

```python
def _is_line_clear(self, grid: List[List[int]], start: Tuple[int, int], end: Tuple[int, int]) -> bool:

    H = len(grid)
    W = len(grid[0]) if H > 0 else 0

    x0, y0 = start
    x1, y1 = end

    # Bresenham直线算法
    dx = abs(x1 - x0)
    dy = abs(y1 - y0)

    if dx == 0 and dy == 0:
        return True

    x, y = x0, y0
    x_inc = 1 if x1 > x0 else -1
    y_inc = 1 if y1 > y0 else -1

    if dx > dy:
        error = dx / 2
        while x != x1:
            # 检查当前点是否为障碍物
            if not (0 <= x < W and 0 <= y < H) or grid[y][x] <= self.obst_v:
                return False

            error -= dy
            if error < 0:
                y += y_inc
                error += dx
            x += x_inc
    else:
        error = dy / 2
        while y != y1:
            # 检查当前点是否为障碍物
            if not (0 <= x < W and 0 <= y < H) or grid[y][x] <= self.obst_v:
                return False

            error -= dx
            if error < 0:
                x += x_inc
                error += dy
            y += y_inc

    # 检查终点
    if not (0 <= x1 < W and 0 <= y1 < H) or grid[y1][x1] <= self.obst_v:
        return False
```

3. technical detail （Software）

Balancing Exploration Efficiency with Mission Goal:

Purpose: To guide exploration towards the known goal (Exit or Start Position) once it is discovered.

$$\text{final_score} = \text{goal_bias_weight} \times \text{distance_score} + (1 - \text{goal_bias_weight}) \times \text{alignment_score}$$

$$\text{distance_score} = \frac{1.0}{1.0 + \frac{\text{dist_to_goal}}{100.0}}$$

$$\text{alignment} = \frac{\text{cand_vec}[0] \times \text{goal_dir}[0] + \text{cand_vec}[1] \times \text{goal_dir}[1]}{\text{cand_dist}}$$

$$\text{alignment_score} = \frac{\text{alignment} + 1.0}{2.0}$$

Distance Score: Prioritizes frontiers closer to the final destination (Exit or Start)
Alignment Score: Measures how well the frontier's direction aligns with the straight-line vector to the final destination.

3. technical detail （Software）

Real-time Situational Awareness (PyRoboViz):

Drawer Class: Manages all visualization elements, including coordinate transformation (Pixel & Meter) and dynamic drawing.

Key Display Elements:

Occupancy Grid Map: Real-time representation of the Free, Occupied, and Unknown spaces.

Robot & Trajectory: A red arrow showing the current pose (x, y, theta), alongside the accumulated path history.

Exploration Targets: Markers for Optimal Frontier Points (current exploration goal) and Detected Exit Point.

code:

```python
class Drawer:
    ROBOT_HEAD_LEN_M  = 0.09
    ROBOT_HEAD_W_M    = 0.06
    ROBOT_SHAFT_LEN_M = 0.05

    def __init__(self, map_size_pixels: int, map_size_meters: float,
                 title: str = "ListGridMap Viewer",
                 show_trajectory: bool = True,
                 origin_lower_left: bool = True):
        self.N = map_size_pixels
        self.map_size_meters = float(map_size_meters)
        self.m_per_pix = self.map_size_meters / float(self.N)
        self.title = title
        self.show_traj = show_trajectory
        self.origin = "lower" if origin_lower_left else "upper"

        self.fig, self.ax = plt.subplots(figsize=(8, 8))
        try: self.fig.canvas.set_window_title("SLAM")
        except Exception: pass
        self.ax.set_title(self.title)
        self.ax.set_xlabel("X (m)"); self.ax.set_ylabel("Y (m)")
        self.ax.set_aspect("equal")
        self.extent_m = (0.0, self.N*self.m_per_pix, 0.0, self.N*self.m_per_pix)
        self.ax.set_xlim(self.extent_m[0], self.extent_m[1])
        self.ax.set_ylim(self.extent_m[2], self.extent_m[3])

        self.img_artist = None
        self.car_artist = None
        self.text_artist = None          # <<< 新增：复用的文字对象
        self.traj_line: Optional[mlines.Line2D] = None  # <<< 新增：复用的轨迹对象
        self.traj_xm: List[float] = []
        self.traj_ym: List[float] = []
        self.hud_artist = None
```

4. function realization

01 Autonomous Exploration

The system continuously detects “frontier” regions—boundaries between known and unknown areas—and selects the most informative next goal.

Obstacle Avoidance 02

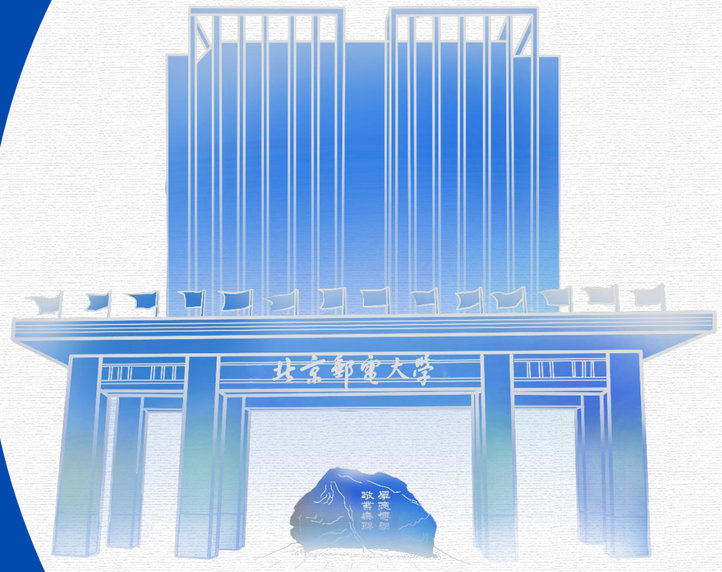
Ensures safe navigation by detecting obstacles in real time through LiDAR data and encoder feedback, adjusting motion before collisions occur.

03 Map Construction

Performs real-time map building and localization using LiDAR scans and odometry data, forming a 2D occupancy grid of the environment.

Automatic Return-to-Base 04

After exploration, the system automatically navigates back to its starting point (home) using the previously constructed map and stored trajectory.

北京郵電大學
Beijing University of Posts and Telecommunications

THANK YOU FOR WATCHING