Autonomous Maze Exploration Laser Radar Vehicle Based on SLAM Compositional Approach

Group 32
Size Wang Yiyang Guo Dachuan Zhao Ze Yuan Luning Jia Yuhao Cai Dachuan Zhao Ke Ning Ruihan Gao Yicheng Wang Xianggan Xu

SoftWare Design

01

SLAM Based on LiDAR data

Map maintenance and updating

```python
def __init__(self, MAP_SIZE_PIXELS, MAP_SIZE_METERS):
    self.w = MAP_SIZE_PIXELS
    self.h = MAP_SIZE_PIXELS
    self.map_size_pixels = MAP_SIZE_PIXELS

    # 主显示地图 grid[y][x]
    self.grid: List[List[int]] = [[128 for _ in range(self.w)] for _ in range(self.h)]

    self.obstacle_votes: np.ndarray = np.zeros( shape: (self.h, self.w), dtype=np.int16)

    self.free_votes: np.ndarray = np.zeros( shape: (self.h, self.w), dtype=np.int16)

    self.locked_map: np.ndarray = np.full( shape: (self.h, self.w), fill_value: 128, dtype=np.uint8)  # 128=未锁定

    self.is_locked: np.ndarray = np.zeros( shape: (self.h, self.w), dtype=bool)

    self.vote_params = {
        'obstacle_lock_threshold': 4,  # 障碍物锁定需要的投票数
        'free_lock_threshold': 10,  # 自由空间锁定需要的投票数
        'conflict_resolve_ratio': 1,  # 冲突解决比例：一方票数需要是另一方的几倍才能获胜
        'noise_filter_threshold': 2,  # 噪声过滤：低于此票数的不参与决策
    }

    self.map_scale_meters_per_pixel = MAP_SIZE_METERS / float(MAP_SIZE_PIXELS)
    self.CurrCarPose = None
```

A voting-based grid map update system for map construction and optimization in SLAM (Simultaneous Localization and Mapping). The core idea is: each map pixel undergoes “voting” through multiple observations. When a certain type of observation (obstacle or free space) reaches a threshold, the pixel is “locked” into a definite state and remains unchanged thereafter.

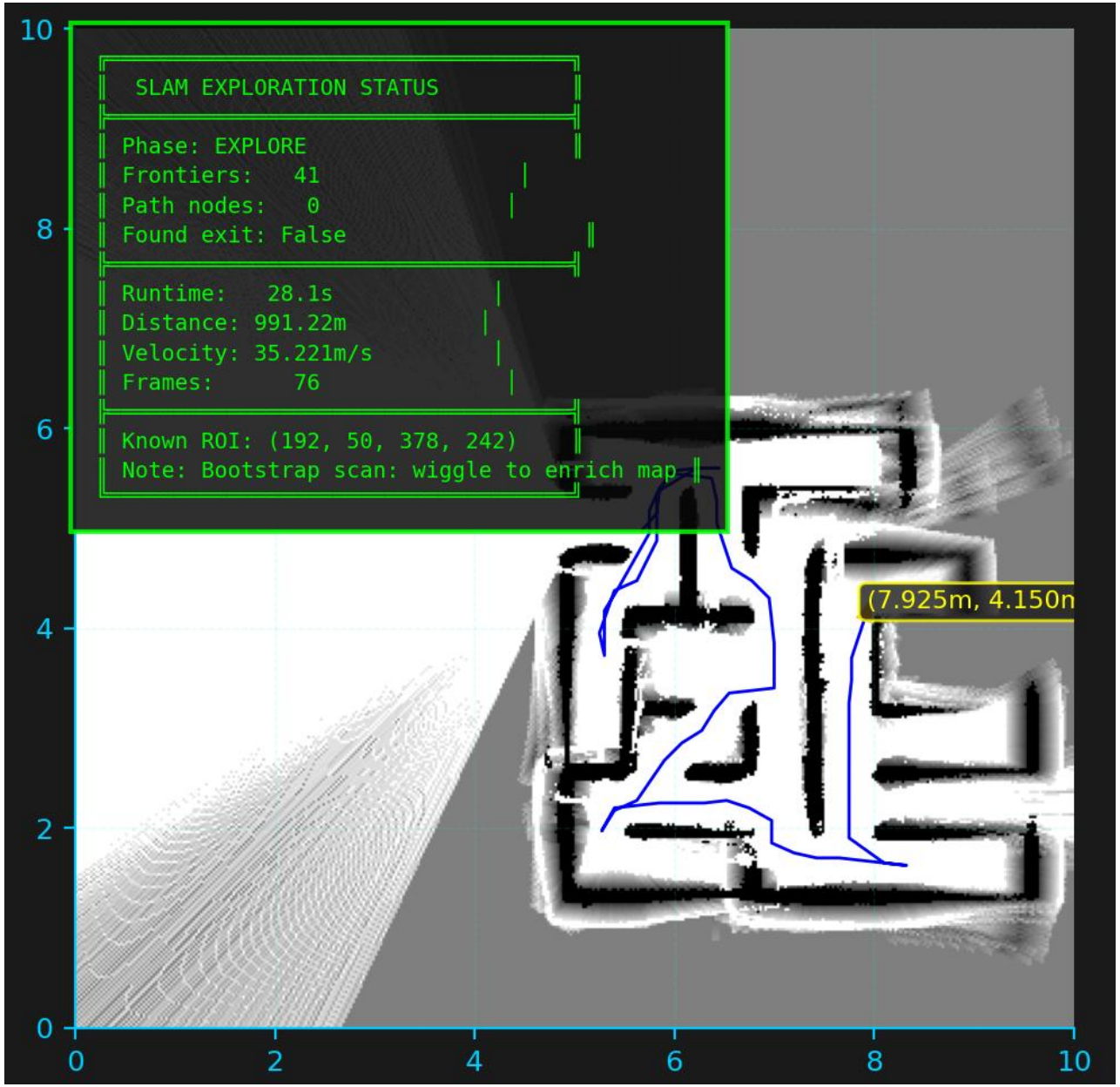
Real-time Composition

```python
def stop(self):
    self._stop_evt.set()

def run(self):
    current_cmd = 0
    while not self._stop_evt.is_set():
        frm = self.link.get_frame(timeout=1.0)
        if not frm:
            continue

        update_triger = False

        if frm.status == 2 and self.last_frm_status == 1:
            update_triger = True
            self.last_frm_status = 2

        if frm.status == 0 and self.last_frm_status == 2:
            update_triger = True
            self.last_frm_status = 0

        if update_triger or self.begin:
            self.begin = False
            frm = self.link.get_frame(timeout=1.0)
            #stm->pc , slam input
```

This code implements an autonomous exploration system based on SLAM: it receives laser and odometry data from the vehicle via serial port, employs SLAM algorithms for real-time mapping and pose estimation, and maintains a robust grid map through a voting mechanism. Concurrently, it utilizes a leading-edge point exploration strategy for path planning, issuing control commands in two phases—“rotate” and “move forward”—while visualizing the map, trajectory, and task status on the main thread. This forms a complete perception-decision-execution closed-loop system.

Ultimate result

The cart can automatically travel from the starting point to the destination.



02

Frontier Detection and Selection

Core Strategy: Frontier-Based Exploration Overview

1. Mission Goal: To autonomously select the optimal next goal point during the MissionPhase.EXPLORE.
2. Frontier Definition: A grid cell located in Free Space (grid >= FREE_thresh=230) that is adjacent to at least one Unknown Space cell (grid == UNKNOWN_val=128).
3. Three-Stage Pipeline: The system ensures safe, efficient, and optimal exploration.

* Detection: High-speed identification of all initial frontier candidates.
* Filtering: Applying safety clearance, connectivity, and sparsity checks.
* Selection: Evaluating the final candidates using a comprehensive Cost-Gain-Bias Utility Function.

Core Strategy: Frontier-Based Exploration Overview

```python
class FrontierExplorer:

    def __init__(self,
                 free_thresh: int = FREE_thresh,
                 unknown_val: int = 128,
                 obstacle_val: int = OBSTACLE_val,
                 obs_clearance_pix: int = 10,
                 gain_radius_pix: int = 8,  # 信息增益计算半径
                 connectivity_check_radius: int = 200,  # 连通性检查半径
                 min_distance_pix: int = 30,  # 最小选择距离，避免打转
                 max_distance_pix: int = 200,  # 最大考虑距离，避免不连通
                 spatial_sample_grid: int = 10,  # 空间采样网格大小
                 max_candidates_to_eval: int = 50,
                 min_wall_distance_pix: int = 50,
                 goal_bias_weight: float = 0.6):  # 新增参数：最小墙壁距离):  # 最多评估的候选点数
```

Stage 1: High-Efficiency Detection via Vectorization

1. Technical Goal: To leverage NumPy's vectorized operations for parallel computation across the entire grid map, avoiding slow Python loops (for speed optimization).

2.Core Algorithm: Convolution-based frontier detection with safety clearance.

Free mask: grid >= FREE_thresh (254).

Unknown mask: (grid > obstacle_val) & (grid < FREE_thresh).

Neighbor check (exclude center): 3×3 ones kernel with zeroed center via ndimage.convolve.

Safety: apply safe_mask for obstacle clearance.

Frontier: frontier_mask = free_mask & has_unknown_neighbor & safe_mask.

Optimization: detect_frontiers_optimized limits work to changed_regions; otherwise runs the fully vectorized detect_frontiers_vectorized and extracts coordinates via np.where.

Stage 1: High-Efficiency Detection via Vectorization

Identifying Free and Unknown Space

```python
def _detect_frontiers_vectorized(self, grid, maze_rect):
    """使用numpy向量化操作加速前沿检测"""
    import numpy as np

    grid_np = np.array(grid)
    H, W = grid_np.shape

    # 一次性计算所有自由空间
    free_mask = grid_np >= self.free_th

    # 使用卷积快速检测边界
    from scipy.ndimage import convolve
    kernel = np.ones((3, 3), dtype=int)
    kernel[1, 1] = 0

    # 检测邻域中的未知区域
    unknown_mask = (grid_np > self.obst_v) & (grid_np < self.free_th)
    has_unknown_neighbor = convolve(unknown_mask.astype(int), kernel, mode='constant') > 0
```

Unknown Neighbor Detection

```python
# 检测邻域中的未知区域
unknown_mask = (grid_np > self.obst_v) & (grid_np < self.free_th)
has_unknown_neighbor = convolve(unknown_mask.astype(int), kernel, mode='constant') > 0
```

Stage 2: Filtering for Safety, Connectivity, and Sparsity

1. Obstacle Safety Clearance

Method: Circular radius check within obs_clearance_pix=10 pixels for each frontier candidate.

Logic: Reject points if any obstacle cell (grid <= obstacle_val) exists within the circular region.

Purpose: Ensures safe robot arrival by maintaining minimum distance from obstacles.

Implementation: Currently in safe_from_obstacle() function using nested loops for circular boundary checking.

```
def safe_from_obstacle(x: int, y: int) -> bool:
    r = self.clear_r
    rr = r * r
    for jy in range(-r, r + 1):
        for ix in range(-r, r + 1):
            if ix * ix + jy * jy > rr:
                continue
            xn, yn = x + ix, y + jy
            if 0 <= xn < W and 0 <= yn < H and grid[yn][xn] <= self.obst_v:
                return False
    return True
```

Stage 2: Filtering for Safety, Connectivity, and Sparsity

2. Region Connectivity Check:

* Purpose: Excludes frontiers that are unreachable or too far, often behind walls or unknown space.

* Method: Uses a fast BFS/Dijkstra-like search (_fast_connectivity_check) to check for a path within max_distance_pix=200. (The full system likely uses GoalChecker for a robust check).

```
def _filter_by_connectivity_simple(self, candidates, curr_pos, grid,maze_rect: Optional[Tuple[int, int, int, int]] = None):

    from collections import deque

    W = len(grid[0])
    H = len(grid)

    x0 = maze_rect[0]
    y0 = maze_rect[1]
    x1 = maze_rect[2]
    y1 = maze_rect[3]
```

Stage 2: Filtering for Safety, Connectivity, and Sparsity

3. Spatial Sparsity Sampling:

* Method: The accepted frontier points are grouped into a sparse grid (spatial_sample_grid=10), with only one representative point kept per grid cell.

* Purpose: Reduces the number of candidates (max_candidates_to_eval=50) that need expensive path planning (Cost) evaluation.

```python
def _spatial_sampling(self, candidates: List[Tuple[int, int]]) -> List[Tuple[int, int]]:
    """空间采样：在网格中每个格子只保留一个候选点"""
    if len(candidates) <= self.max_eval:
        return candidates

    # 将空间分割为网格，每个网格保留一个点
    grid_dict = {}
    for px, py in candidates:
        grid_x = px // self.sample_grid
        grid_y = py // self.sample_grid
        grid_key = (grid_x, grid_y)

        if grid_key not in grid_dict:
            grid_dict[grid_key] = []
        grid_dict[grid_key].append((px, py))

    # 每个网格选择一个代表点（可以是随机选择或者选择网格中心附近的）
    sampled = []
    for grid_points in grid_dict.values():
        # 选择网格内的第一个点作为代表（也可以用其他策略）
        sampled.append(grid_points[0])


    return sampled
```

Stage 3: Optimal Selection - Goal-Oriented Scoring and Ranking

1. Core Formula: Candidates are ranked by a final score, maximizing alignment with the mission goal:

$$S = W_{dist} \cdot \text{DistScore} + W_{align} \cdot \text{AlignScore}$$

2. Distance Score (DistScore): Measures proximity to the final mission goal, normalized as 1/(1 + d/100), favoring closer points.

3. Alignment Score (AlignScore): Measures the directional alignment (dot product) between the candidate's direction and the overall goal vector. (Mapped to the range [0, 1]).

4. Weighting: The parameter W_{bias} (goal_bias_weight, e.g., 0.6) controls the balance between the two metrics: $W_{dist} = W_{bias}$ and $W_{align} = 1 - W_{bias}$

Stage 3: Optimal Selection - Goal-Oriented Scoring and Ranking

```
    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]

print(f"终点导向筛选: {len(candidates)}个候选点, "
      f"最佳得分={scored_candidates[0][1]:.3f}")

return result
```

03

User Interface

Stage 1: Initializing the visualization environment and all drawing objects

```python
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
```

Stage 2:Plotting additional frontier point sets, distinguished from the main frontier point set by different colors

```python
def _draw_frontier_points_extra(self,
                                frontier_points: Optional[List[Tuple[int, int]]],
                                best_frontier_point: Optional[Tuple[int, int]]):
    """绘制前沿点，普通前沿点用蓝色圆点，最优前沿点用红色圆点"""

    # 清除旧的前沿点
    for artist in self.frontier_artists:
        try:
            artist.remove()
        except:
            pass
    self.frontier_artists.clear()

    if self.best_frontier_artist is not None:
        try:
            self.best_frontier_artist.remove()
        except:
            pass
        self.best_frontier_artist = None

    # 绘制所有前沿点（lvse小圆点）
    if frontier_points:
        for x_pix, y_pix in frontier_points:
```

Stage 3: Initializing the visualization environment and all drawing objects

```python
def _draw_car(self, x_m: float, y_m: float, theta_deg: float):
    # 1) 箭头（为简单起见仍然移除重画；也可改成复用：更新 FancyArrow 的 verts）
    if self.car_artist is not None:
        try: self.car_artist.remove()
        except Exception: pass
        self.car_artist = None

    theta_rad = math.radians(theta_deg)
    dx = self.ROBOT_SHAFT_LEN_M * math.cos(theta_rad)
    dy = self.ROBOT_SHAFT_LEN_M * math.sin(theta_rad)

    self.car_artist = self.ax.arrow(
        x_m, y_m, dx, dy,
        head_width=self.ROBOT_HEAD_W_M,
        head_length=self.ROBOT_HEAD_LEN_M,
        fc="r", ec="r", length_includes_head=True, zorder=5
```

```python
def _pix_to_m(self, x_pix: float, y_pix: float) -> Tuple[float, float]:
    return x_pix * self.m_per_pix, y_pix * self.m_per_pix
```

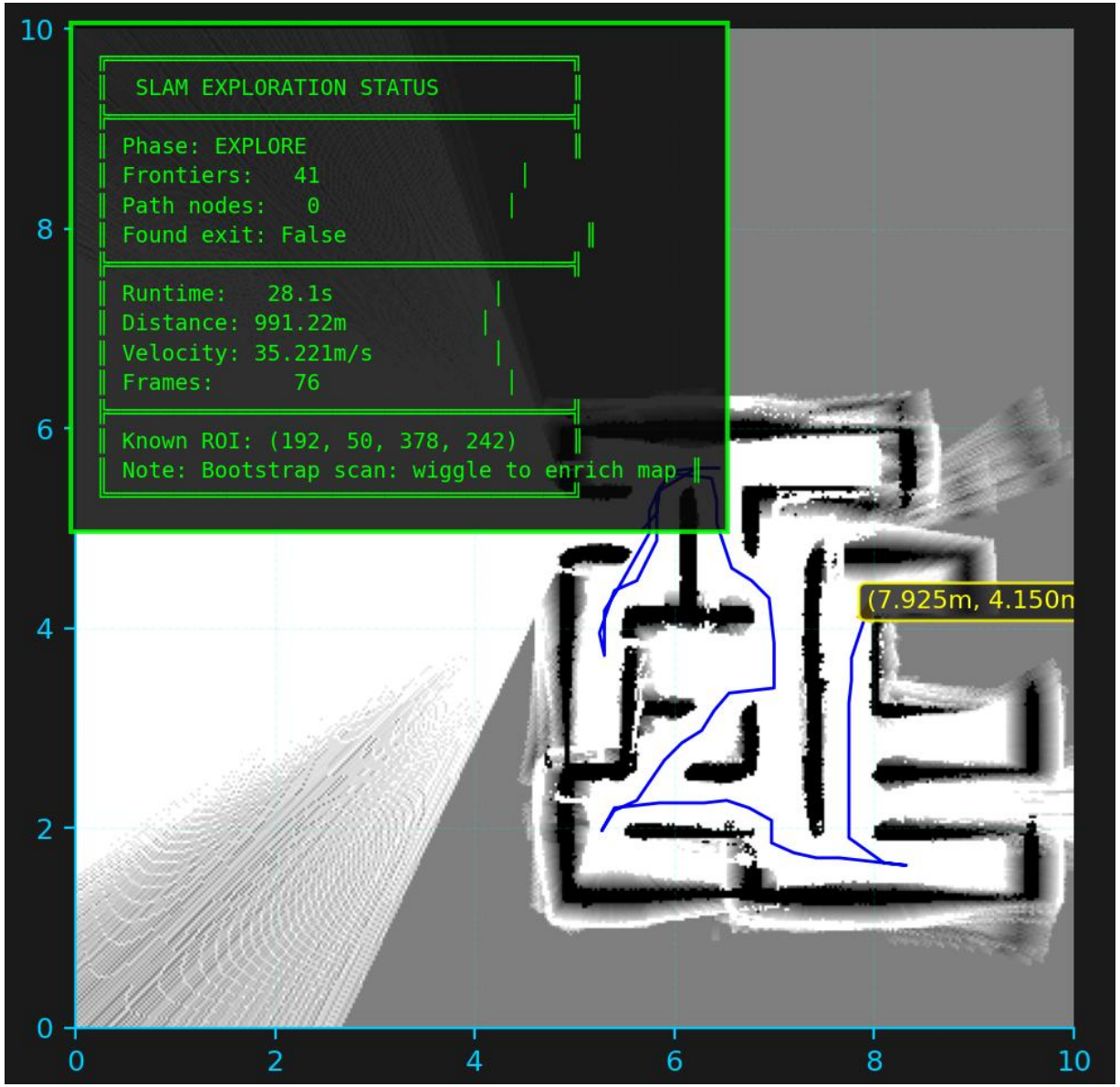
```python
def _update_traj(self, x_m: float, y_m: float):
    if not self.show_traj:
        return
    self.traj_xm.append(x_m); self.traj_ym.append(y_m)
    if self.traj_line is None:
        self.traj_line = mlines.Line2D(self.traj_xm, self.traj_ym,
                                       linewidth=1.0, color="b", zorder=3)
        self.ax.add_line(self.traj_line)
    else:
        self.traj_line.set_data(self.traj_xm, self.traj_ym)
```

Stage 4:Plot predicted exit points(left) and centrally schedule all visual elements(right)

```python
def _draw_exit_pose(self, exit_pose_pix: Optional[Tuple[int, int]]):
    """绘制预测的出口点，用绿色大圆圈标记"""

    # 清除旧的出口点标记
    if self.exit_pose_artist is not None:
        try:
            self.exit_pose_artist.remove()
        except:
            pass
        self.exit_pose_artist = None

    # 绘制新的出口点
    if exit_pose_pix is not None:
        x_pix, y_pix = exit_pose_pix
        x_m, y_m = self._pix_to_m(x_pix, y_pix)
        self.exit_pose_artist = plt.Circle( xy: (x_m, y_m), radius=0.12,
                                            color='green', alpha=0.9, zorder=6,
                                            linewidth=2, fill=False)  # 绿色圆环
        self.ax.add_patch(self.exit_pose_artist)
```

```python
def display(self, grid: List[List[int]],
            car_pose_pix_deg: Optional[Tuple[float, float, float]] = None,
            extra_text: Optional[str] = None,
            roi_rect=None,
            frontier_points: Optional[List[Tuple[int, int]]] = None,
            best_frontier_point: Optional[Tuple[int, int]] = None,
            spatially_sampled_: Optional[List[Tuple[int, int]]] = None,
            exit_pose_pix: Optional[Tuple[int, int]] = None
            ) -> bool:
    # 地图
    if self.img_artist is None:
        self.img_artist = self.ax.imshow(
            grid, cmap=cm.gray, vmin=0, vmax=255,
            origin=self.origin, extent=self.extent_m, zorder=1
        )
    else:
        self.img_artist.set_data(grid)

    # 小车与轨迹
    if car_pose_pix_deg is not None:
        x_pix, y_pix, theta_deg = car_pose_pix_deg
        x_m, y_m = self._pix_to_m(x_pix, y_pix)
        self._draw_car(x_m, y_m, theta_deg)
        self._update_traj(x_m, y_m)
```

Interface Result



04

A* Path Planning

Check path accessibility

```python
class GoalChecker:


    def __init__(self,
                 free_thresh: int = FREE_thresh,
                 unknown_val: int = 128,
                 obstacle_val: int = OBSTACLE_val,
                 check_radius_pix: int = 5,  # 终点周围需要自由的半径
                 path_check_interval: int = 1):  # 路径检查间隔（帧数）

        self.free_th = int(free_thresh)
        self.unknown_v = int(unknown_val)
        self.obst_v = int(obstacle_val)
        self.check_radius = int(check_radius_pix)
        self.check_interval = int(path_check_interval)

        self._frame_count = 0  # 帧计数器
        self._last_check_result = False  # 上次检查结果（缓存）

    def _has_path_bidirectional_bfs(self,
                                    grid: List[List[int]],
                                    start: Tuple[int, int],
```

First, verify whether the target point and its surrounding area constitute free space. Then, within a defined known region, employ the A* algorithm to search for feasible paths. Performance is optimized through frame interval control and result caching, making it suitable for real-time path feasibility assessment in SLAM navigation systems.

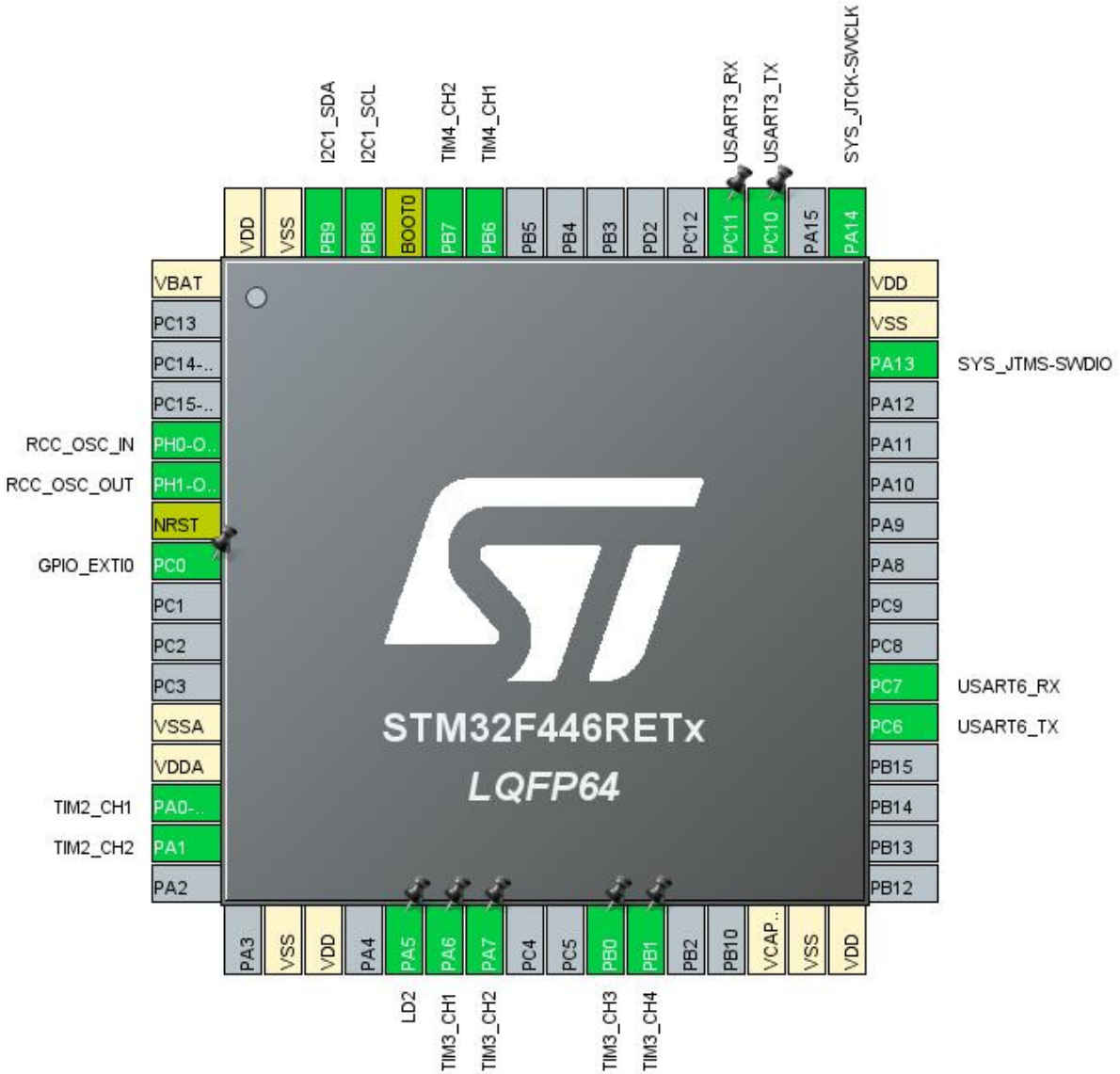
Complete A* Algorithm

```python
def get_successors(x: int, y: int, parent_x: int, parent_y: int) -> List[Tuple[int, int]]:
    """获取跳点的后继节点"""
    successors = []

    # 确定搜索方向（基于父节点）
    if parent_x == -1:  # 起点
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1),
                      (-1, -1), (1, -1), (-1, 1), (1, 1)]
    else:
        # 剪枝：只搜索"自然"和"强制"邻居
        dx = (x - parent_x) // max(1, abs(x - parent_x)) if x != parent_x else 0
        dy = (y - parent_y) // max(1, abs(y - parent_y)) if y != parent_y else 0

        directions = []
        if dx != 0 and dy != 0:  # 对角
            directions.extend([(dx, 0), (0, dy), (dx, dy)])
        else:  # 直线
            directions.append((dx if dx != 0 else 0, dy if dy != 0 else 0))

    for dx, dy in directions:
        jump_point = jump(x, y, dx, dy)
        if jump_point:
            successors.append(jump_point)
```

A complete path planner has been implemented, primarily designed to plan paths from start to end points for mobile robots or similar objects on two-dimensional grid maps. It offers multiple path planning algorithms (this project uses the A* algorithm) along with path simplification capabilities. Additionally, it supports obstacle expansion processing within the map to prevent collisions.

HardWare Design

Main function



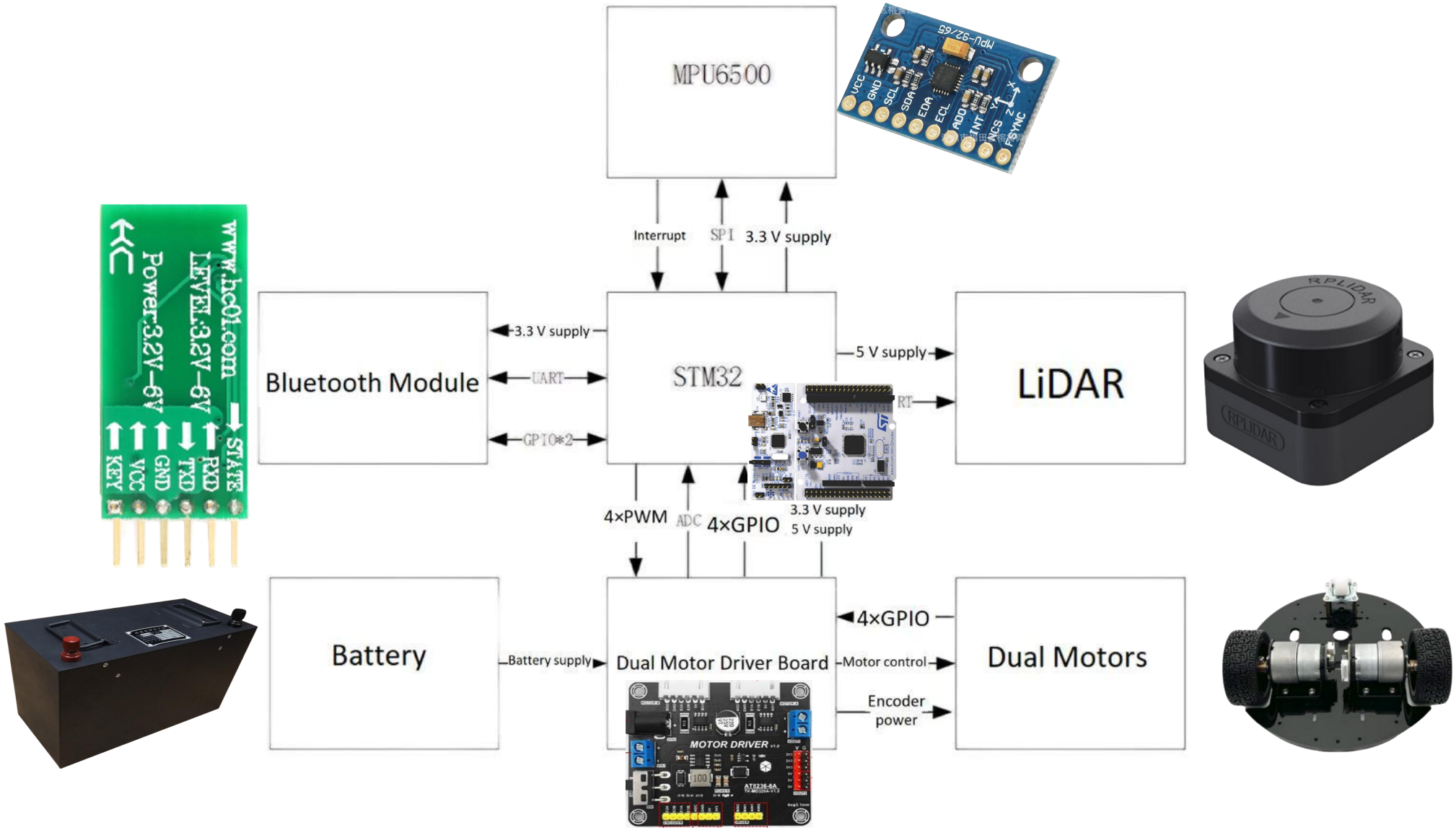
The hardware interfaces and pin assignments for this project are shown in the table below: The controller utilizes an STM32 microcontroller, with TIM3's multiple channels connected to the motor driver to form PWM control.
Encoder phases A/B are connected to TIM2/TIM4 in encoder mode to measure rotational speed and direction;
The Bluetooth module (HC-04) communicates with the host computer via USART3 for command transmission and status feedback.
RPLIDAR C1 outputs ranging data via USART6;
The IMU (MPU6500) provides acceleration and angular velocity information via I²C1 (SDA/SCL).

Device Parameter

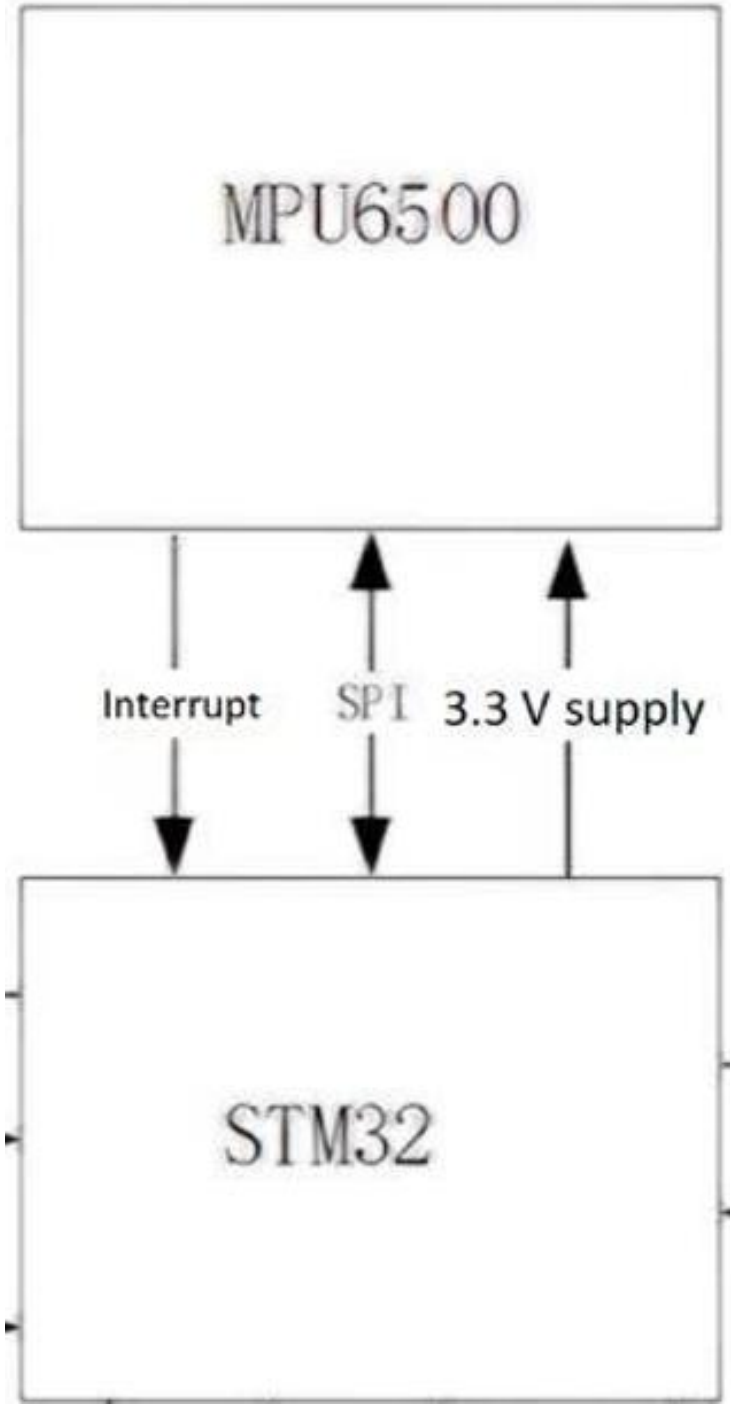
The above chain forms a closed-loop system of "sensing-control-actuation-communication": encoders and IMUs provide attitude/velocity feedback, PID calculations on the MCU generate PWM signals output to the motor driver, while serial communication with the host computer enables debugging and monitoring. This supports the vehicle's capabilities for straight-line movement, turning, navigating mazes, and returning to the starting point.

Module / Device	Pin	Peripheral	Function / Notes
Dual Motor Driver (MC520)	PA6	TIM3_CH1	Quadrature encoder input (A/B)
Dual Motor Driver (MC520)	PA7	TIM3_CH2	Quadrature encoder input (A/B)
Dual Motor Driver (MC520)	PB0	TIM3_CH3	Quadrature encoder input (A/B)
Dual Motor Driver (MC520)	PB1	TIM3_CH4	Quadrature encoder output (A/B)
Encoder – Quadrature A/B	PA0	TIM2_CH1	Quadrature encoder output (A/B)
Encoder – Quadrature A/B	PA1	TIM2_CH2	Quadrature encoder input (A/B)
Encoder – Quadrature A/B	PB6	TIM4_CH1	Quadrature encoder output (A/B)
Encoder – Quadrature A/B	PB7	TIM4_CH2	Quadrature encoder output (A/B)
Bluetooth (HC-04)	PC10	USART3_TX	Data out
Bluetooth (HC-04)	PC11	USART3_RX	Data in
LiDAR (RPLIDAR C1)	PC6	USART6_TX	Data out
LiDAR (RPLIDAR C1)	PC7	USART6_RX	Data in
6-Axis IMU (MPU6500)	PB8	I2C1_SDA	I²C data line; transfers IMU data (ac
6-Axis IMU (MPU6500)	PB9	I2C1_SCL	I²C clock line; provides synchronous

Hardware System Architecture

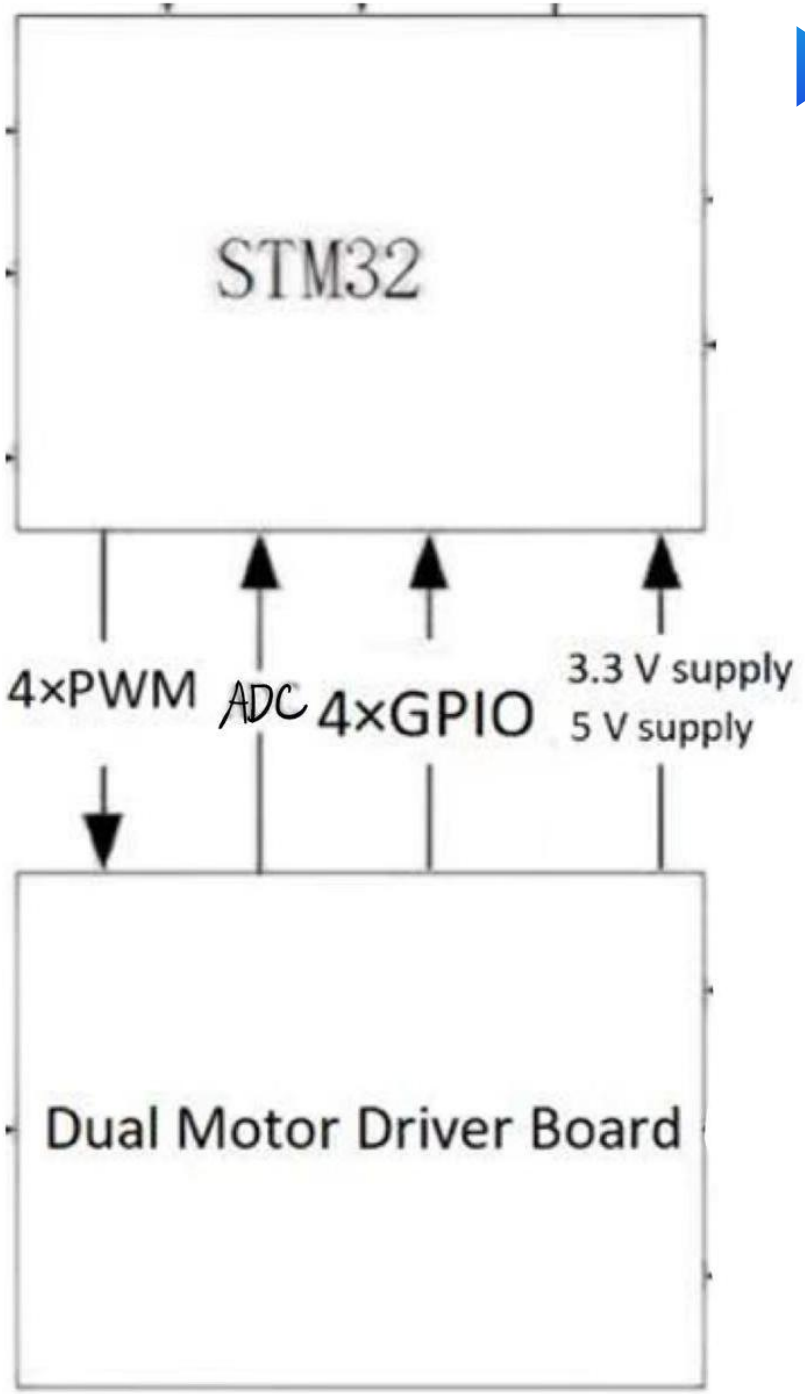


Connection between STM32 and MPU



This system achieves attitude and motion sensing through STM32 + MPU6500:
After generating the project in CubeMX, integrate the MPU6500 driver (register read/write via ReadReg/WriteReg). The main program completes device reset, clock configuration, ID verification, range setting, and interrupt enablement.

MPU6500 Pin	STM32 Pin	Function / Notes
VCC	3.3V	Power supply
GND	GND	Ground
SCL / SCK	SPIx_SCK	SPI clock
SDA / SDI	SPIx_MOSI	Master Out, Slave In
AD0 / SDO	SPIx_MISO	Master In, Slave Out
NCS / CS	Any GPIO	Chip Select
INT	Any GPIO	Interrupt (data-ready)



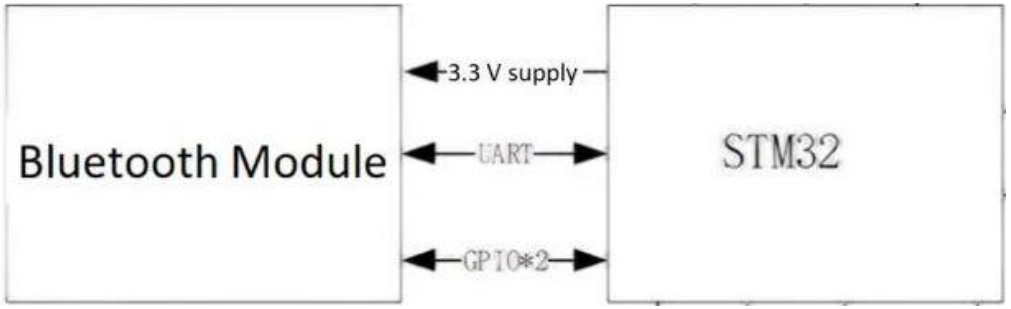
Connection between STM32 and Dual Motor Driver Board

The system implements a closed-loop motor control on the STM32. In each cycle, the main loop reads encoder pulses to compute speed and samples current/voltage via ADC.

The motor-control module then runs PID, producing a new PWM duty cycle that is applied to the PWM outputs. With encoder and current sensing as feedback, the loop repeats continuously, achieving stable and real-time control of motor speed/torque.

The data flow is: Main loop → Sensing (encoder/current) → PID computation → PWM update → Actuation → Feedback, forming a robust closed loop.

Connection between STM32 and Bluetooth Moudule



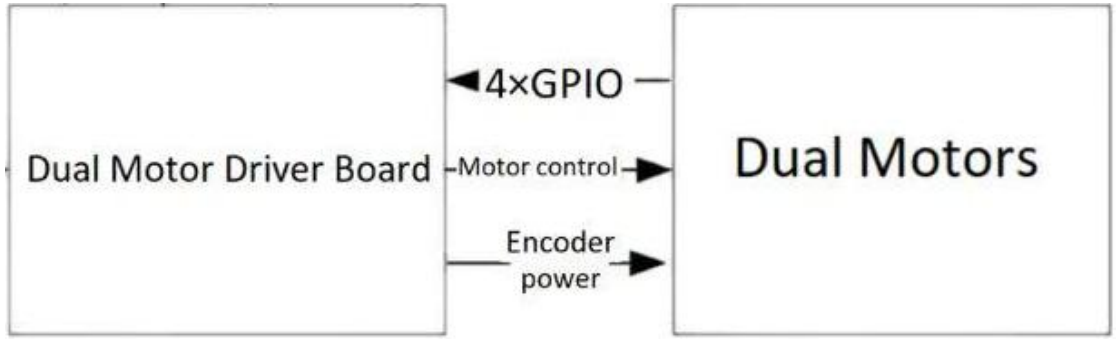
HC-04 Pin	STM32 Pin	Function / Notes
VCC	5V	Power input (4.5–6 V)
GND	GND	Ground
TXD	USARTx_RX	Bluetooth TX → MCU RX
RXD	USARTx_TX	Bluetooth RX ← MCU TX

The system interfaces with the HC-04 Bluetooth module over UART (3.3 V supply, plus two GPIO lines for control/status). During initialization, the firmware pulls the enable pin low to exit AT mode, sets the 9600 baud rate, and enables DMA/ring-buffer continuous reception. It then performs a connection handshake, with the module returning “OK”/“ERROR.”
In the steady-state loop:

Telemetry from the main program is sent to the phone via UART → DMA;

Commands from the phone are placed into the UART RX buffer and trigger an RX interrupt;

The main task parses Bluetooth data and executes control logic.
This establishes a reliable：
Main App ↔ **UART/DMA** ↔ **HC-04** ↔ **Phone** data path with low CPU overhead and real-time responsiveness.



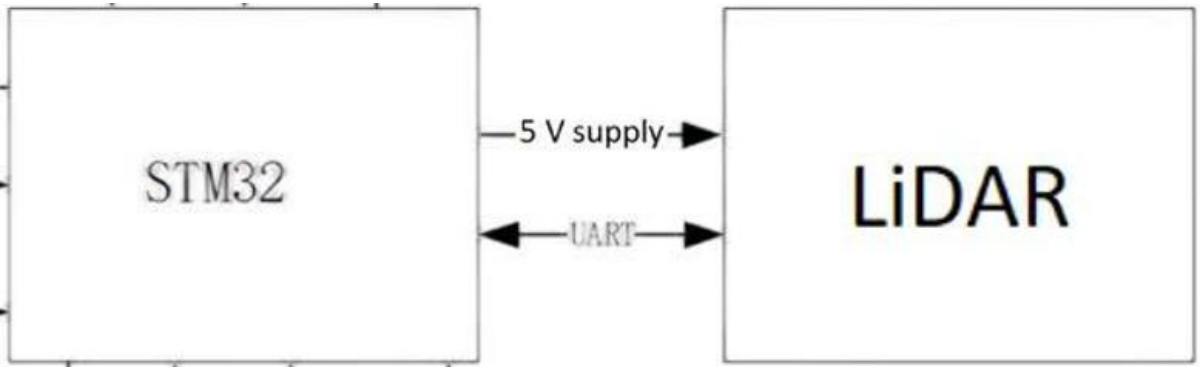
Connection between Dual Motors and Driver Board

The system implements a closed-loop dual-motor control with an STM32 and a motor driver board. After power-up, the firmware initializes PWM timers and encoder interfaces, then loads motor parameters.

In each control cycle, the main loop reads encoder pulses, converts them to speed, runs PID in the PWM-control module to compute a new duty cycle, and updates PWM outputs to the driver. Fault/status checks run in parallel.

The driver actuates two motors, while encoder A/B signals feed back to the MCU, forming a robust loop of **sense → compute → actuate → feedback.**

Motor Driver Board Pins	STM32 Pins	Function / Notes
PWM1A/B	TIMx_CH1/CH2	Motor 1 PWM control
PWM2A/B	TIMy_CH1/CH2	Motor 2 PWM control
EN1/2	GPIO Output	Motor enable control
DIR1/2	GPIO Output	Motor direction control
ENC1A/B	TIMz_CH1/CH2	Motor 1 encoder inputs
ENC2A/B	TIMw_CH1/CH2	Motor 2 encoder inputs
5V / 3.3V	Power Management	Encoder power supply
GND	GND	Ground



Connection between STM32 and LiDAR

RPLIDAR C1 Pin	STM32 Pin	Function / Notes
VCC (5V)	5V Power	Power (external supply)
GND	GND	Ground
RX	USARTx_TX	Lidar receive ← MCU tra
TX	USARTx_RX	Lidar transmit → MCU re

The RPLIDAR C1 connects over a USART + DMA high-speed serial link. After power-up, the firmware initializes the lidar and queries device info, then enables DMA ring-buffer reception. Once the start-scan command is issued, the lidar streams point frames continuously; DMA writes them to the buffer and raises an interrupt. The main task performs packet parsing → coordinate transformation → application logic (e.g., obstacle avoidance/mapping). Pin & power mapping: **VCC 5V, GND, RX ← MCU USARTx_TX, TX → MCU USARTx_RX**. This design delivers high throughput with low CPU load and stable timing—ideal for real-time use.

Team Assignments

Dachuan Zhao | 2023213254

Integrated Planning

Define the roadmap and milestones; split hardware work into four phases: Module Research → Code Design → Bring-up & Integration → Joint Interop Testing. Lead consolidation of MPU6500, HC-04, RPLIDAR C1.

Core Module Requirements

Specify technical baselines for LiDAR, STM32, AT8236 (motor driver), MC520 (motor) to guide performance targets and ownership.

Team Synergy & Execution

Organize by module with clear ownership + flexible collaboration.

Prioritize bring-up/compatibility; keep progress synced, collect issues quickly, and unblock with resources (equipment, data, staffing).

Resource & Scope Balance

Balance limited hardware debug time with algorithm needs; coordinate interfaces early to avoid churn.

Deliverables

Compile final PPT and written report, integrating all members' contributions into a single, coherent presentation.

Team Assignments

Yicheng Wang | 2023213063

Low-Level Drivers & Motor HW Support

Implement encoder driver (A/B capture, pulse count, direction) and compute real-time speed.

Develop motor driver PWM outputs for start/stop and forward/reverse control.

Speed Closed-Loop (PID)

Feed encoder data into pid.c/h and help tune parameters.

Debug and resolve control issues to improve response and speed-tracking accuracy.

Hardware Setup & Debug

Build the motor control rig (motors, encoders, driver).

Solder MCU/peripherals; verify wiring with multimeter/oscilloscope; fix shorts/opens/signal issues to meet driver requirements.

Team Assignments

ZeYuan | 2023213184

Ownership

Lead dual-motor drive + LiDAR software.

Driver Modules

PWM speed + direction control for two DC motors.

Real-time dual-motor speed acquisition/processing.

UART/DMA driver—receive, parse, convert raw frames to angle–distance packets.

Key Integrations

Close the speed loop using encoder feedback (PID-ready).

Execute motor speed/direction commands precisely.

Deliver stable, reliable LiDAR data streams.

Integration & Debug

Co-debug on the robot platform (HW + SW).

Test and fix motor response, LiDAR accuracy, encoder reliability.

Optimize drivers for performance and stability to meet system targets.

Team Assignments

Ruihan Gao | 2023213290

Ownership

Dual-motor drive development + hardware integration support.

Core Drivers

motor.c/h: PWM generation via timers for two DC motors; independent speed + fwd/rev control.

encoder.c/h: High-precision A/B incremental encoder capture (IC/EXTI), edge & direction detect, quadrature count.

Motion Estimation

Compute real-time speed (RPM/rad·s^{-1}) and distance from pulses for velocity control & navigation.

Closed-Loop Speed Control

Use encoder speed as feedback vs. target.

Adjust PWM duty based on speed error to regulate torque, achieve precise dual-motor control and disturbance rejection.

Hardware Build & Debug

Assist chassis assembly and main control wiring.

Plan/layout critical circuits (motor driver, encoder, sensors) for EMI robustness.

Fine-pitch soldering of driver ICs/MCU peripherals; bring-up & verification with DMM/scope.

LiDAR Integration

Integrate RPLIDAR: power & UART/SPI check, point-cloud acquisition, and fusion with encoder odometry for reliable perception.

Team Assignments

Xianggan Xu | 2023213306

Ownership

Inertial navigation & sensing (IMU + LiDAR) development and integration.

Core Drivers & Processing

MPU6500.c/h: I²C/SPI driver; real-time gyro (°/s) & accel (g) readout with unit conversion.

Static calibration: zero-bias estimation/compensation for gyro & accel.

rplidar.c/h: UART protocol; receive/verify/parse frames into angle–distance(–intensity) point clouds.

State Estimation & Perception

Accel-based displacement (Δx, Δy) via time-stamped integration (noting drift; to be fused later).

Assist with PC-side visualization for real-time point clouds (mapping/obstacle-check/debug).

Hardware Setup & Sensor Debug

IMU installation & calibration; optimize mounting and damping to cut vibration noise.

Static/dynamic tests; analyze noise & temp drift; tune calibration parameters.

Use scope/logic analyzer to debug comms & data integrity.

LiDAR Integration

Ensure stable power & UART settings; verify point-cloud quality (range/resolution/noise).

Tune scan rate/angle resolution; test under varied conditions (lighting/reflectivity).

Try basic time-sync / fusion with IMU/encoder for robust perception.

Team Assignments

Yiyang Guo | 2023213593

Ownership

Led Bluetooth comms + motion control; supported HW integration & validation.

Bluetooth Comms (bluetooth.c/h)

Built bidirectional link (BLE/SPP) for stable command channel.

Parsed remote commands: speed, direction (fwd/rev/steer), mode select.

Telemetry uplink: periodic/event frames (wheel speeds, distance, PWM duty, battery voltage, status flags).

Motion Control (pid.c/h, control.c/h)

Dual-motor PID speed control (independent L/R loops).

Differential-drive logic: straight, reverse, in-place turn, arbitrary-arc turns.

Constant-Velocity (CV) mode; seamless switch between RC and CV.

Tuning & Results

Step/ramp tests; hand-tuned Kp/Ki/Kd.

Better straight-line stability, faster steering response, higher cruise accuracy, near-zero steady-state error.

Hardware Integration

Assisted robot assembly; verified wiring for MCU, Bluetooth, driver, encoders.

Continuity/level/SI checks with DMM/scope; debugged wiring/interference issues to ensure reliable comms & control.

Team Assignments

Size Wang | 2023213482

Specific Tasks:
①Implement the ListGridMap class, responsible for map initialization, LiDAR data processing (obstacle and free space voting mechanisms), and map locking logic (stable region determination).

②Develop coordinate conversion (meter-pixel mutual conversion), map display functions, and map state statistics (locked pixel ratio, counts of obstacles/free spaces, etc.).

③Interface with path planning modules to provide grid data for passability judgment.

Team Assignments

Ke Ning | 2023213647

①Implement the A* algorithm supporting 4-directional/8-directional movement, design heuristic functions (Manhattan distance/octile distance), and manage search nodes via a priority queue.

②Implement the Jump Point Search (JPS) algorithm to optimize A* efficiency on grid maps, including jump point judgment and path interpolation (Bresenham algorithm).

③Develop auxiliary functions such as path simplification and start/goal point snapping (to solve boundary issues) to ensure planned paths are smooth and feasible.

Team Assignments

Luning Jia | 2023213351

①Design the navigation state machine (MissionPhase), implementing switching logic for phases like exploration (EXPLORE), return to start (RETURN_HOME), and go to exit (GO_TO_EXIT).

②Develop core navigation functions: waypoint generation and tracking , in-place swing mapping , and waypoint .

③Define configuration classes (Pose, WheelGeom, Tolerance, etc.) to unify system parameter formats and coordinate data interaction between modules.

Team Assignments

Yuhao Cai | 2023213416

①Implement frontier point detection and filtering : select optimal exploration targets from candidates via distance filtering, spatial sampling, and connectivity checks.

②Develop goal reachability detection : use an A* variant to determine if a path exists from the current position to the target, including boundary checks and obstacle avoidance.

③Maintain valid map regions : dynamically update bounding boxes of known areas to ensure navigation stays within explored ranges.

④Handle serial communication : implement data reception threads, process serial read/write operations, frame parsing, and queue management to ensure reliable transmission of sensor data and control commands.