

Goals 4 Submission

-DriveSystem.py

- Motor Class
 - Class: Controls one motor using two PWM GPIO pins.
 - `__init__(self, io, pinA, pinB)`
 - Sets up pins and initializes PWM.
 - `setLevel(self, level: float)`
 - Sets motor speed and direction.
 - `stop(self)`
 - Stops motor movement.
- DriveSystem Class
 - Class: Controls both left and right motors for movement.
 - `__init__(self, io, left_pins: tuple[int, int], right_pins: tuple[int, int])`
 - Creates left and right Motor objects.
 - `drive(self, mode: str)`
 - Drives with a named movement mode.
 - `stop(self)`
 - Stops both motors.

-Sense.Py

- IR Class
 - Class: Reads a single IR sensor.
 - `__init__(self, io, pin: int)`
 - Sets up one GPIO pin for input.
 - `read(self) → int`
 - Returns 1 if tape detected, 0 otherwise.
- LineSensor Class
 - Class: Reads three IR sensors as a unit.
 - `__init__(self, io, pin_L: int, pin_M: int, pin_R: int)`
 - Assigns pins for left, middle, right sensors.
 - `read(self) → tuple[int, int, int]`
 - Returns a 3-tuple of IR values.

- AngleSensor.Py

- AngleSensor Class
 - `__init__(self, io)`
 - Initializes the sensor interface (e.g., pigpio, I2C).
 - Sets up internal state for heading tracking and optional offset calibration.
 - `readadc(self, address):`

- Reads an 8-bit integer (0–255) from the specified ADC channel (0 or 1)
- `read_angle(self) → float`
 - Computes and returns the heading angle in degrees based on scaled ADC voltages (range: -180° to $+180^\circ$).

`street_behaviors.py`

- Behaviors Class
 - `__init__(self, io, drive, sensor, AngleSensor)`
 - Initializes motor, sensor, and angle tracking interfaces. Sets time constants and state filters.
 - `follow_line(self) → str`
 - Follows a black tape line using IR feedback.
 - Returns "intersection" if triple detection, "end" if all sensors lose tape, or continues tracking.
 - `pull_forward(self)`
 - Drives forward briefly to ensure the robot fully enters an intersection before stopping.
 - `turning_behavior(self, direction: str) → int`
 - Executes a left or right turn using angle feedback from AngleSensor.
 - Waits to leave and re-detect the tape before stopping.
 - Returns the net turn amount in 45° steps (± 1 , ± 2 , etc).

`MapBuilding.py`

- STATUS (Enum)
 - Defines the five possible states for any street direction from an intersection:
 - UNKNOWN: no information yet
 - NONEXISTENT: confirmed no street in that direction
 - UNEXPLORED: street detected but not yet driven
 - DEADEND: street ends without connecting to another intersection
 - CONNECTED: street successfully traversed in both directions
- Class Intersection
 - `__init__(self, x, y)`
 - Initializes all 8 directional streets from this intersection as UNKNOWN.
- PoseTracker Class
 - `heading_vectors = {`
 - `0: (0.0, 0.5),`
 - `1: (-0.5, 0.5),`
 - `2: (-0.5, 0.0),`

- 3: (-0.5, -0.5),
- 4: (0.0, -0.5),
- 5: (0.5, -0.5),
- 6: (0.5, 0.0),
- 7: (0.5, 0.5)
- }
- color_map = {
- STATUS.UNKNOWN: 'black',
- STATUS.NONEXISTENT: 'lightgray',
- STATUS.UNEXPLORED: 'blue',
- STATUS.DEADEND: 'red',
- STATUS.CONNECTED: 'green'
- }
- Tracks the robot's current (x, y, heading) pose and manages a dictionary of Intersection objects.
- `__init__(self)`
 - Starts the pose at (0, 0) facing north (heading 0), with an empty intersection map.
- `calcmove(self)`
 - Moves the robot one unit forward based on its current heading.
- `calcturn(self, turn_amount: int)`
 - Updates the robot's heading using 45° increments (mod 8).
- `calcuturn(self)`
 - Rotates the robot 180° (adds 4 to the heading).
- `pose(self) → tuple[int, int, int]`
 - Returns the robot's current position and heading as a tuple.
- `getintersection(self, x, y) → Intersection`
 - Returns the Intersection at (x, y); creates it if it doesn't exist yet.
- `show(self)`
 - Uses matplotlib to draw:
 - All intersections as gray dots on a grid
 - Streets extending from each intersection, color-coded by STATUS
 - The robot's current heading as a magenta arrow

– brain.py

- Main Script: Coordinates the robot's logic loop by connecting sensor readings, motor commands, navigation behaviors, and map building.
- Function:
 - Continuously prompts the user for a command ("straight", "left", or "right").

- "straight": calls follow_line() until intersection or dead end, then optionally calls pull_forward().
- "straight" → Calls follow_line() to drive forward until it detects an "intersection" or "end".
 - If an intersection is found, it calls pull_forward() to fully enter it.
 - If an end is detected (no tape), the robot:
 - Marks the street as a DEADEND on the map.
 - Automatically performs a U-turn (left spin),
 - Drives back to the previous intersection, and
 - Updates the map to mark the return path as CONNECTED.
- "left" or "right": calls turning_behavior() to rotate the robot by a tracked angle, then updates map heading.
- Also Handles:
 - Map updates using PoseTracker and Intersection status logic (e.g., CONNECTED, DEADEND, UNEXPLORED, NONEXISTENT).
- Visualization of the map at every step using PoseTracker.show().

Changes made from Goals3 Content

- Made one Behaviors class instead of having separate classes for Turning_behaviors, pull_forward and line_following.
- Reset all of our filters by creating a reset filters function so that we do not face any problems when reaching a dead end and making a u turn back.
- Fine tuned turning by having the robot turn in the opposite direction after completing a turn till the IR sensors read (0,1,0)

Magnetometer Testing

- Channel 0: Max = 2019, Min = 95
- Channel 1: Max = 218, Min = 85
- Bringing an Iphone 16 close to the sensor, caused the sensor readings of both channels to jump by around 50-75. When i slightly shifted the position of the iphone the magnetometers readings were extremely sensitive

```

graph LR
    subgraph Low_Level_Layer [Low-Level Layer  
(Hardware Interfaces)]
        Sense_py[Sense.py  
- IR: Reads one reflectance sensor  
- LineSensor: Reads all 3 IR sensors (left, middle, right)]
        DriveSystem_py[DriveSystem.py  
- Motor: Controls one motor via two PWM pins  
- DriveSystem: Controls both motors with named commands]
        AngleSensor_py[AngleSensor.py  
- readadc(address): Reads analog input via GPIO  
- read_angle(): Computes heading angle in degrees]
    end

    Sense_py --> DriveSystem_py
    DriveSystem_py --> AngleSensor_py
    AngleSensor_py --> follow_line

    follow_line[follow_line(): Follows tape using IR] --> pull_forward[pull_forward(): Drives forward briefly]
    pull_forward --> Turn_behavior[Turn_behavior(direction): Uses Angles Sensor to Complete turn and returns ≈45° steps]
    Turn_behavior --> brain_py[brain.py  
- Prompts user input: 'straight', 'left', 'right'  
- Calls behavior methods accordingly  
- On 'end': Automatically performs a U-turn, updates map  
- On 'Intersection': Calls pull_forward()  
- Uses PoseTracker to update position and heading]
    brain_py --> MapBuilding_py[MapBuilding.py  
- PoseTracker: Tracks (x, y, heading) and manages Intersections  
- Intersection: Stores STATUS values (UNKNOWN, etc.)  
- show(): Visualizes full map and robot pose]
    MapBuilding_py --> brain_py
    MapBuilding_py --> High_Level_Layer[High-Level Layer  
(Logic + Mapping)]
    High_Level_Layer --> brain_py

```

```

graph TD
    Start([Start Cycle]) --> Visualize[Visualize current map and robot pose]
    Visualize --> Prompt[Prompt user for command: 'straight', 'left', or 'right']
    Prompt --> CommandLeft{Command == 'left' or 'right'?}
    CommandLeft -- Yes --> StoreHeading[Store current heading]
    StoreHeading --> GetIntersection[Get current Intersection]
    GetIntersection --> CallTurn[Call turning_behavior() and get turn_amount]
    CallTurn --> UpdateHeading[Update heading using pose.calcturn(turn_amount)]
    UpdateHeading --> WasNewHeading{Was new heading UNKNOWN?}
    WasNewHeading -- Yes --> MarkUnexplored[Mark it as UNEXPLORED]
    MarkUnexplored --> PerformUturnLeft[Perform U-turn with turning_behavior('left')]
    PerformUturnLeft --> CallFollowLineReturn[Call follow_line() to return to last intersection]
    CallFollowLineReturn --> MarkReturnPathConnected[Mark return path as CONNECTED]
    MarkReturnPathConnected --> EndCycle([End of Cycle])
    WasNewHeading -- No --> MarkDeadend[Mark current heading as DEADEND]
    MarkDeadend --> PerformUturnLeft
    PerformUturnLeft --> CallFollowLineReturn
    CallFollowLineReturn --> MarkReturnPathConnected
    MarkReturnPathConnected --> EndCycle
    UpdateHeading --> AnySkippedHeadings{Any skipped headings?}
    AnySkippedHeadings -- Yes --> MarkNonexistent1[Mark as NONEXISTENT]
    MarkNonexistent1 --> Continue[Continue]
    AnySkippedHeadings -- No --> WereSkippedHeadings{Were skipped headings UNKNOWN or UNEXPLORED?}
    WereSkippedHeadings -- Yes --> MarkNonexistent2[Mark as NONEXISTENT]
    MarkNonexistent2 --> Continue
    WereSkippedHeadings -- No --> Continue
    Continue --> Prompt
    CommandLeft -- No --> CommandStraight{Command 'straight'?}
    CommandStraight -- Yes --> CallFollowLine[Call follow_line()]
    CallFollowLine --> ReachedIntersection{Reached 'Intersection' or 'end'?}
    ReachedIntersection -- Reached --> CallPullForward[Call pull_forward() to enter intersection]
    CallPullForward --> UpdatePose[Update pose with pose.calcmove()]
    UpdatePose --> MarkStreetConnected[Mark street as CONNECTED]
    MarkStreetConnected --> EndCycle
    ReachedIntersection -- No --> UpdateHeading
  
```

Outcome A – In-Place Turn

- Called `turning_behavior()` to get the turn amount (+/- 1)
- Updated the robot's heading using `pose.calcturn(turn_amount)`.

- The position (x, y) stays the same.

Outcome B – Straight Movement to Next Intersection

The robot followed the line and reached a new intersection.

- After reaching an intersection via `follow_line()`, called `pose.calcmove()` to update (x, y) based on current heading.
- The heading remains unchanged.

Outcome C - The robot found a dead-end, made a U-turn, and returned to the same intersection but in the opposite heading

- Marked the current heading as DEADEND.
- Called `pose.calcturn(turn_amount)` to perform a 180° turn.
 - where `turn_amount = behaviors.turning_behavior("left")`
- Waited until the robot re-crossed an intersection before calling `calcmove()` to update (x, y) again.

(4) how you updated the map (intersections) for the outcomes A/B/C in Sect. 5.

Outcome A – Turned in Place

The robot remained at the same intersection but changed heading.

- The robot remains at the same intersection (x, y) but changes its heading.
- No new intersection is added.
- The robot marks the new facing direction as `STATUS.UNEXPLORED` if not previously explored.
- Other directions that were not seen can be marked as `STATUS.UNKNOWN`
- No edge is added to the map since there was no movement.

Outcome B – Drove to a New Intersection

The robot moved one unit forward and reached a new intersection.

- The robot moves one unit in the direction of its current heading, arriving at a new intersection (x', y').
- A new Intersection object is created at the new position
- The previous intersection's street in the heading direction is set to `STATUS.CONNECTED`.
- The new intersection's street in the reverse direction is also marked `STATUS.CONNECTED`.
- adds an edge between the two intersections.

Outcome C – U-Turned at Dead-End

The robot reached a dead-end, U-turned, and returned to the previous intersection.

- The robot reaches a dead-end, then performs a U-turn and returns to the same intersection.
- The street it attempted to drive into is marked as STATUS.DEADEND.
- The robot's heading is reversed.
- No new intersection is added, but the map now reflects that direction as a dead end.