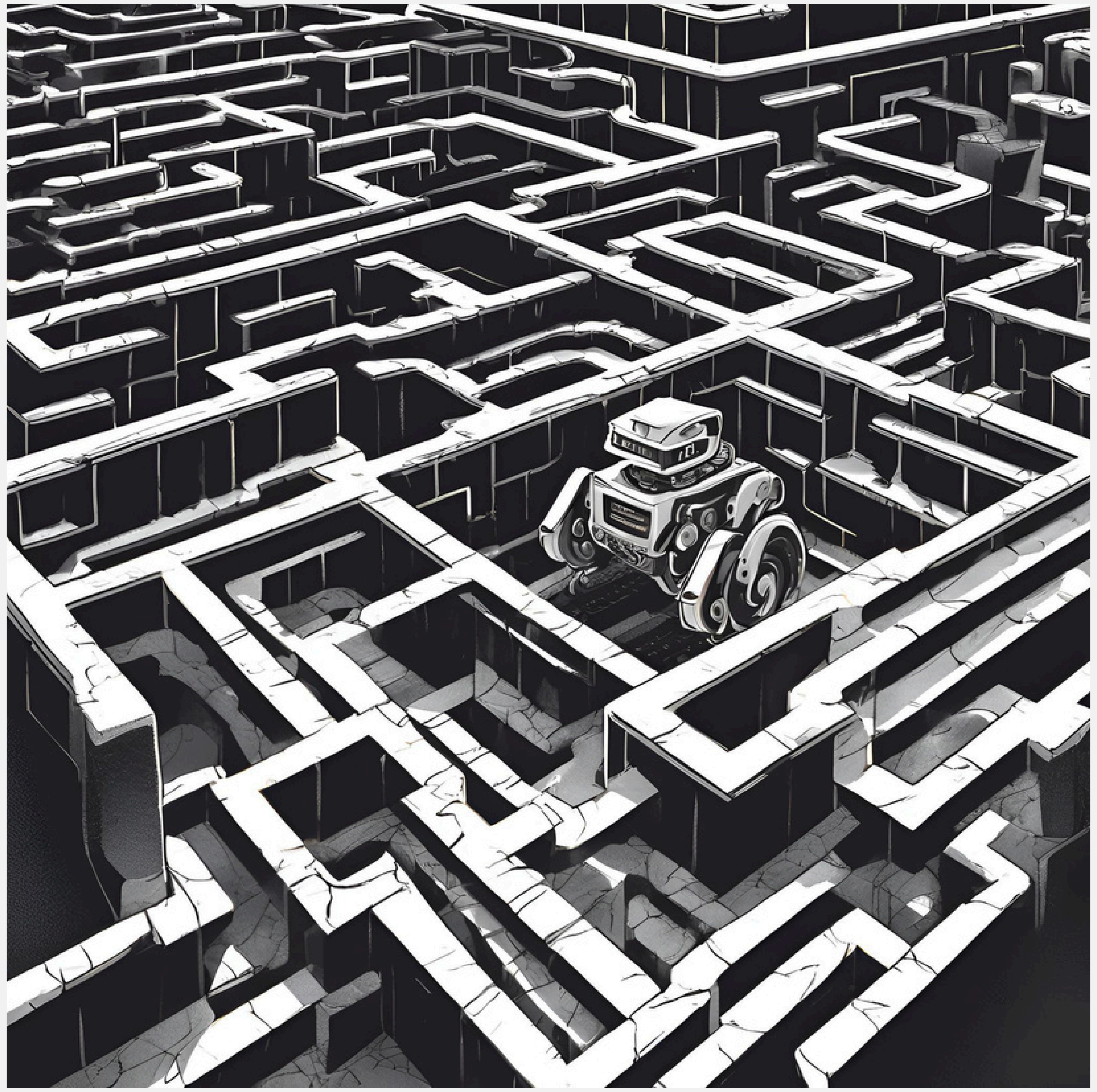
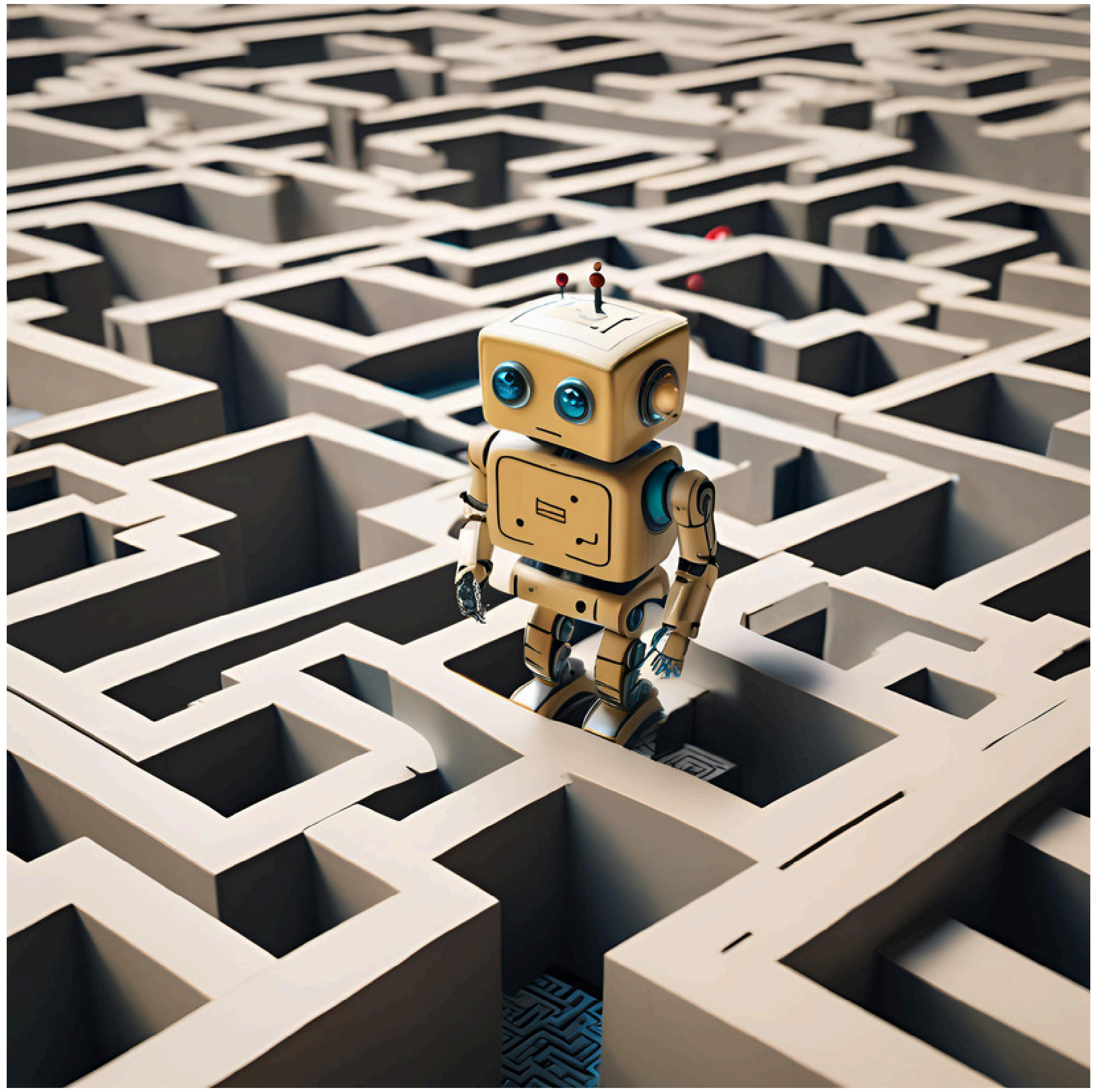
# Intelligent Mobile Robotics

## Robotic Challenge Solver using the CiberRato Simulation Environment



**Made by:**
Tiago Mostardinha, 103944
Henrique Cruz, 103442

# Contents



# Robot Movement

### Control Mechanisms

**PD Controllers:** Large and small rotations

**P Controller:** Straight-line movement

### Robot Movement between Cells

**First:** Rotate to align with the desired direction

**Second:** Move forward while performing slight adjustments

### Error Calculations

**Large Rotations:** Difference in desired and filtered direction

**Straight-Line:** Difference in desired and current position

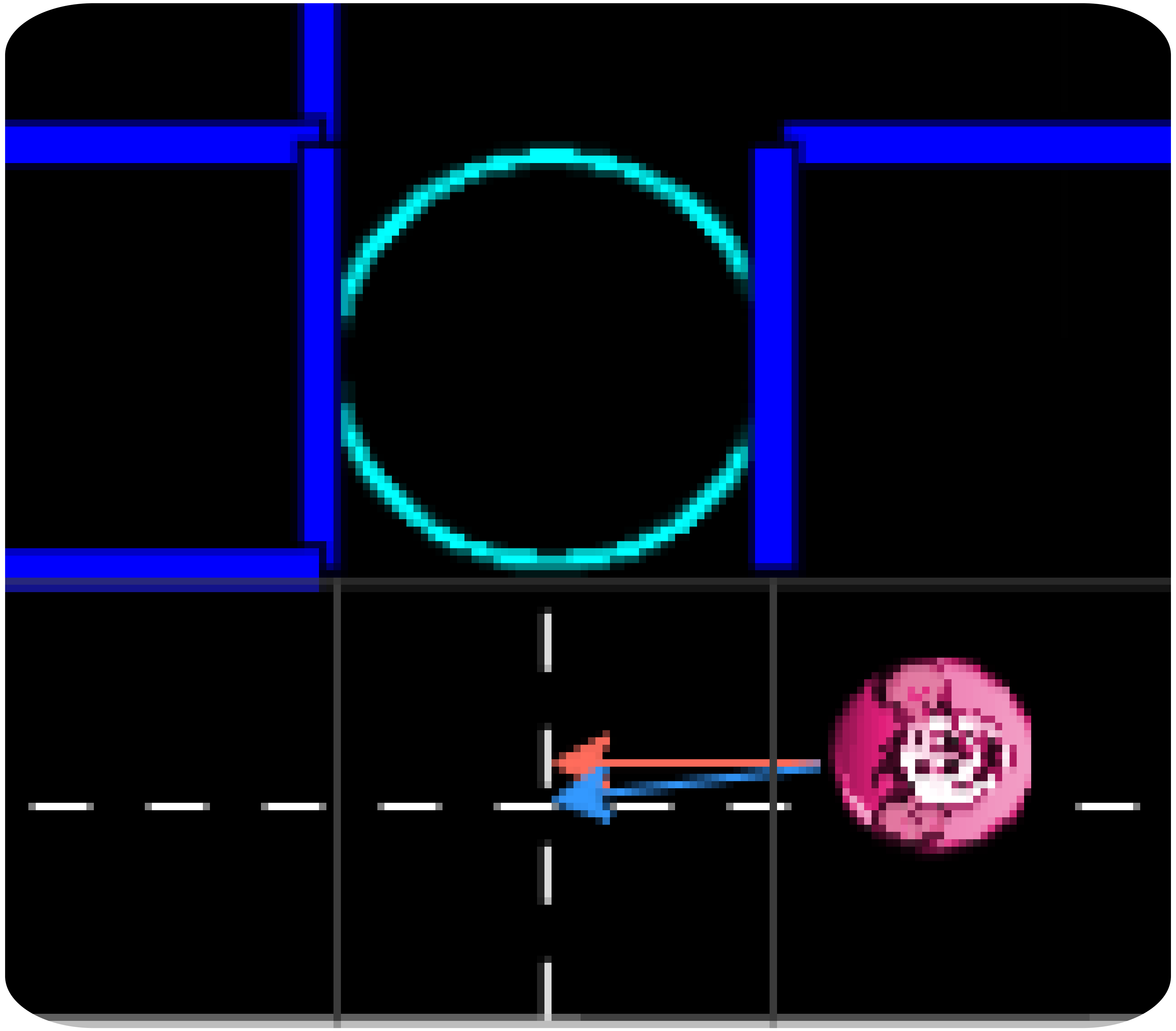
**Small Adjusts:** Difference in position along the opposite direction axis and current position

# Robot Movement

**Small Adjustments Controller**

- **Red Arrow:** Position the robot would go to **without** the small adjustments **controller**
- **Blue Arrow:** Position the robot would go to **with** the small adjustments **controller**

- **Goal:** Prevent cumulative errors



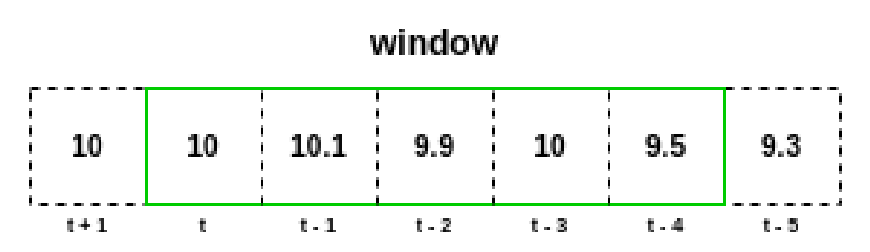
# Robot Position and Recalibration

- **Position:** Calculated using Movement Model with periodic recalibration to account for cumulative errors.
- **Recalibration Goal:** Return a reliable position value.
- **How:** Sliding window with sensor values. If the standard deviation is lower than the threshold than we have a valid direction

### Recalibrated Position Formula

Position = Cell Center + (0,4 - Distance to Wall)

### Sensor Reliability




```
window.append(value(t))
if std.dev (window) < Reliability Threshold:
    return Distance
return None
```

### Recalibration Steps

- Detect Wall
- Approach Wall
- Measure distance to the wall
- Recalibrate robot position
- Return to starting position

# Robot Direction

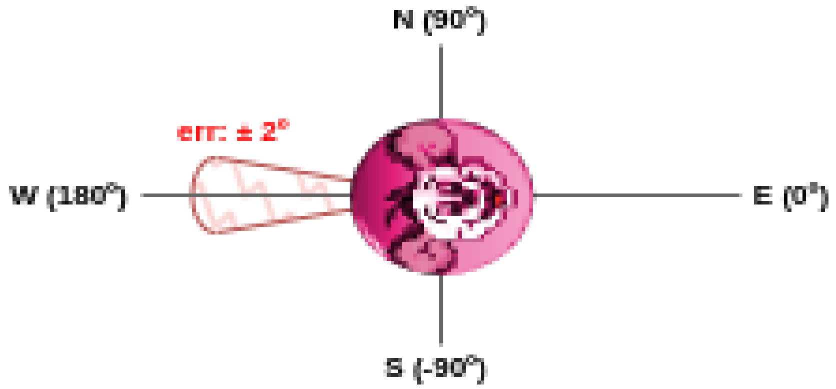
The **direction** of the robot was **determined** using **two methods:**

- **Movement Model** (Cumulative Error)
- **Compass** (Noise)

However, since both **methods** have an **error** associated, we decided to use **Kalman Filter** to mitigate them.

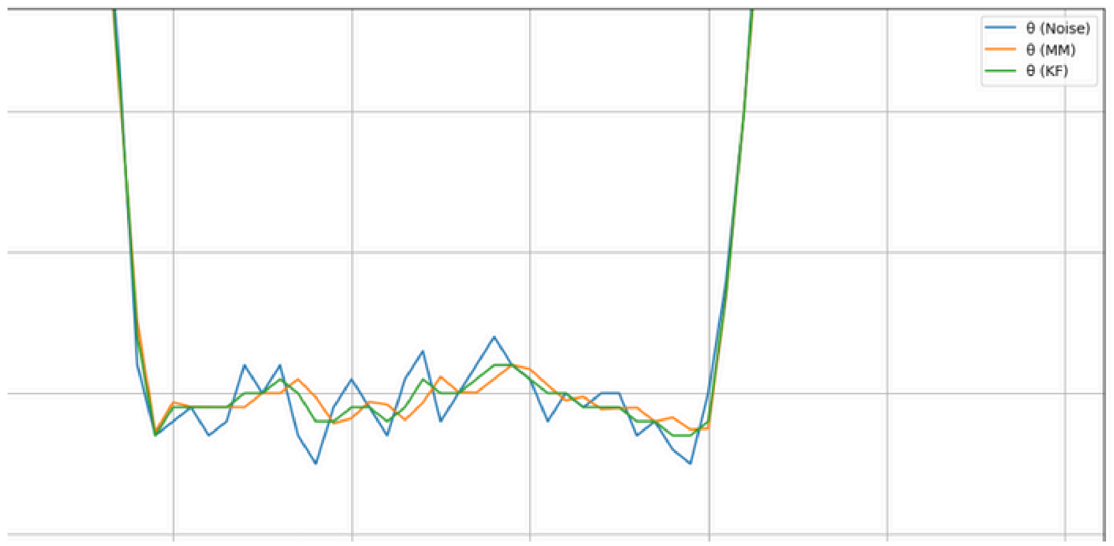
- In the Kalman filter, angles that are within the **2nd and 3rd quadrant** are **reflected** to the **1st and 4th** quadrant.
- This is due to the angles near the west direction, **fluctuate rapidly** to **positive** and **negative** values.

**Compass**




**Movement Model**

**Filtered Angle Kalman Filter**



# Kalman Filter

The Kalman Filter **estimates** the state of a dynamic system by **combining measurements** and a **prediction model.**

- ***X*** is the estimated state
- ***F*** is the state transition model
- ***B*** is the control-input model
- ***U*** is the control vector
- ***Q*** is the process noise with covariance
- ***Z*** is the measurement taken at time t
- **H** is the observation model of the state/event

**Forecast**

$$\overline{X}_t = F_t X_{t-1} + B_t U_t$$

$$\overline{P}_t = F_t P_{t-1} F_t^T + Q_t$$

**Measurement integration**

$$K_t = \frac{\overline{P}_t H_t^T}{H_t \overline{P}_t H_t^T + R_t}$$

$$X_t = \overline{X}_t + K_t (Z_t - H_t \overline{X}_t)$$

$$P_t = (I - K_t H_t) \overline{P}_t$$

# Kalman Filter

**Angle Kalman Filter**

Kalman Filter adaptation for our use case.

- **Forecast:** We used the **prediction of the angle and outputs** from the Movement Model

- **Measurement Integration:** We used the value of the **compass with noise**

To determine the filter elements, we **decomposed** the **Movement Model** expressions to determine **F**, **B**, and **Q**.

$$\theta_{t+1,t} = \theta_t + rot_{t+1,t}$$

$$\equiv \theta_{t+1,t} = \theta_t + \frac{out_t^R - out_t^L}{D}$$

$$\equiv \theta_{t+1,t} = \theta_t + \frac{\frac{input_t^R + out_t^R}{2} - \frac{input_t^L + out_t^L}{2}}{D}$$

$$\equiv \theta_{t+1,t} = \theta_t + \frac{\frac{out_t^R - out_t^L}{2} + \frac{input_t^R + input_t^L}{2}}{D}$$

$$\begin{bmatrix} \theta_{t+1,t} \\ out_{t+1,t}^L \\ out_{t+1,t}^R \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \theta_t \\ out_t^L \\ out_t^R \end{bmatrix} + \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} input_t^L \\ input_t^R \end{bmatrix}$$

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & out_t^L & 0 \\ 0 & 0 & out_t^L \end{bmatrix} * 0.015$$

$$R = \frac{2}{360}^{2}$$

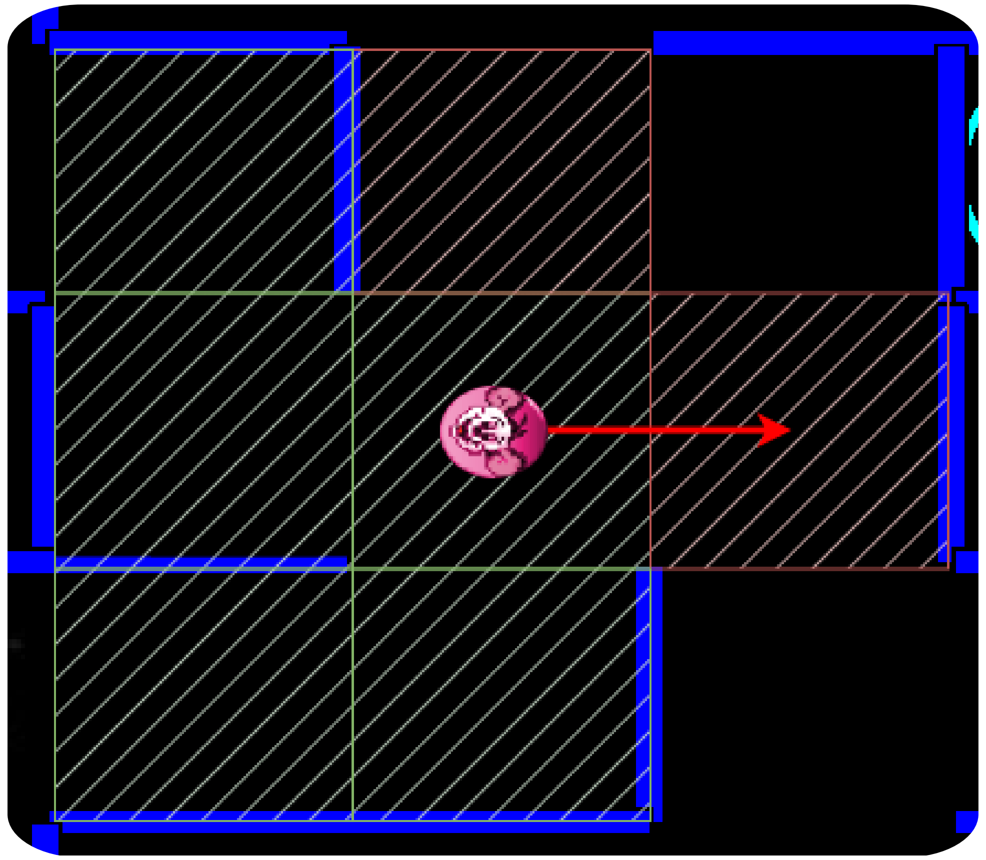
$$H = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

# Robot Decision

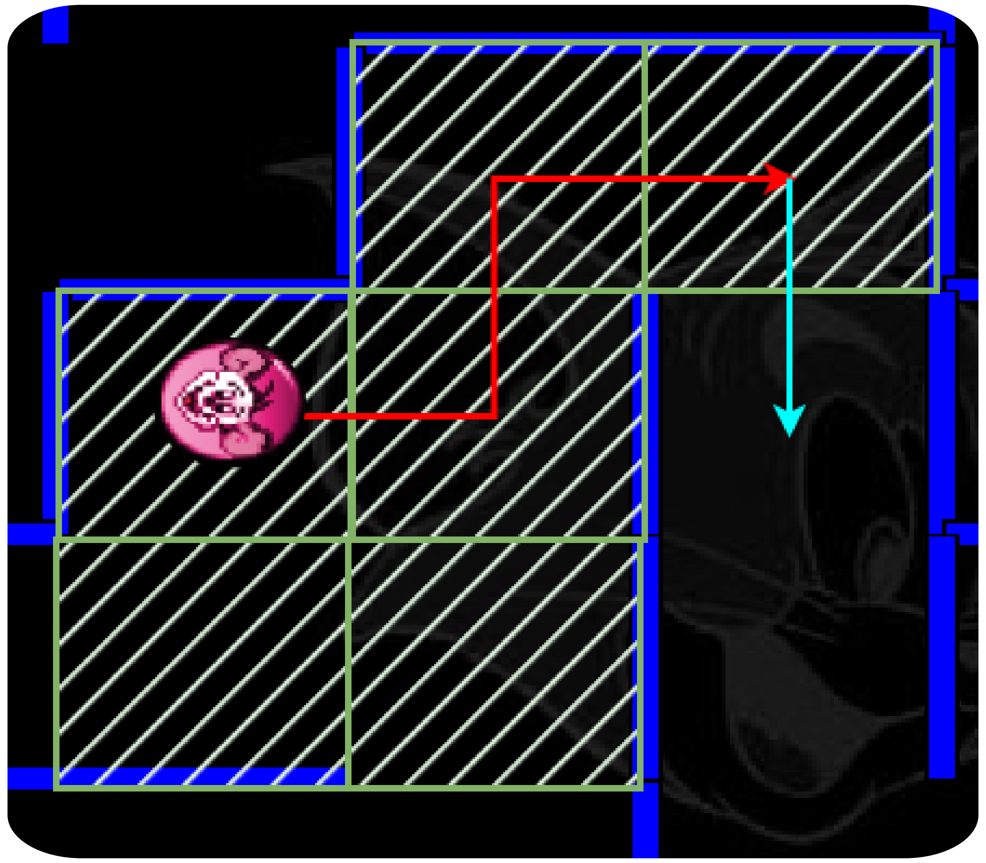
The map was divided into cells of equal dimensions.

1. **Initial Phase:**
    - The robot navigates the maze using only **obstacle sensor** data.
2. **Breadth-First Search (BFS):**
    - When all adjacent cells have been explored, the robot travels to the **nearest cell** with **unexplored neighbors**.
3. **Exploration Completion:**
    - The process continues until there are no cells with **unexplored neighbors.**

**1st Scenario**



**2nd Scenario**



# Mapping

**Maze Representation Size**

A maze representation is created with four times the size of the initial maze.

**Robot Starting Position**

The center of the maze representation corresponds to the robot starting position in the maze.

**Map Building**

As the robot travels the maze new cells are added to the maze representation.

# Planning

Planning is divided into the following 3 steps:

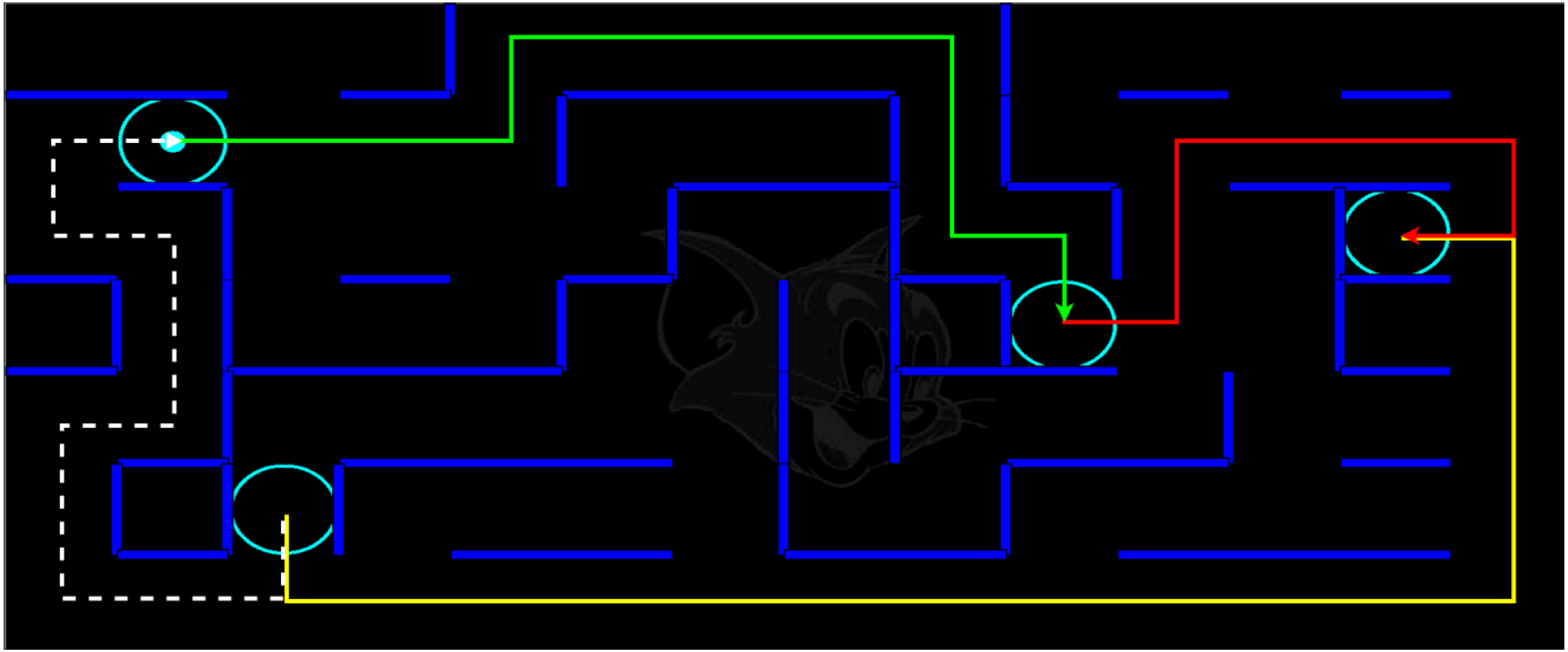
1. **Beacons Cell Identification**
    - While exploring the map, the robot registers the **cell** where it has **identified a beacon**
2. **Shortest path through Beacons**
    - A path is calculated that **passes through all beacons,** it must **start and end** with the **initial beacon**
    - Doing all **permutations of possible paths** that pass through all beacons, and using **BFS**, we can conclude which is the shortest path
3. **Write the Shortest Path in a file**
    - This path is **written to a file** containing all the cells the robot must pass through to reach its goal.

**Permutations**

$$BeaconsIDs = [0, 1, 2, 3]$$

$$path = 0 \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \\ \dots & \dots & \dots \end{pmatrix} 0$$

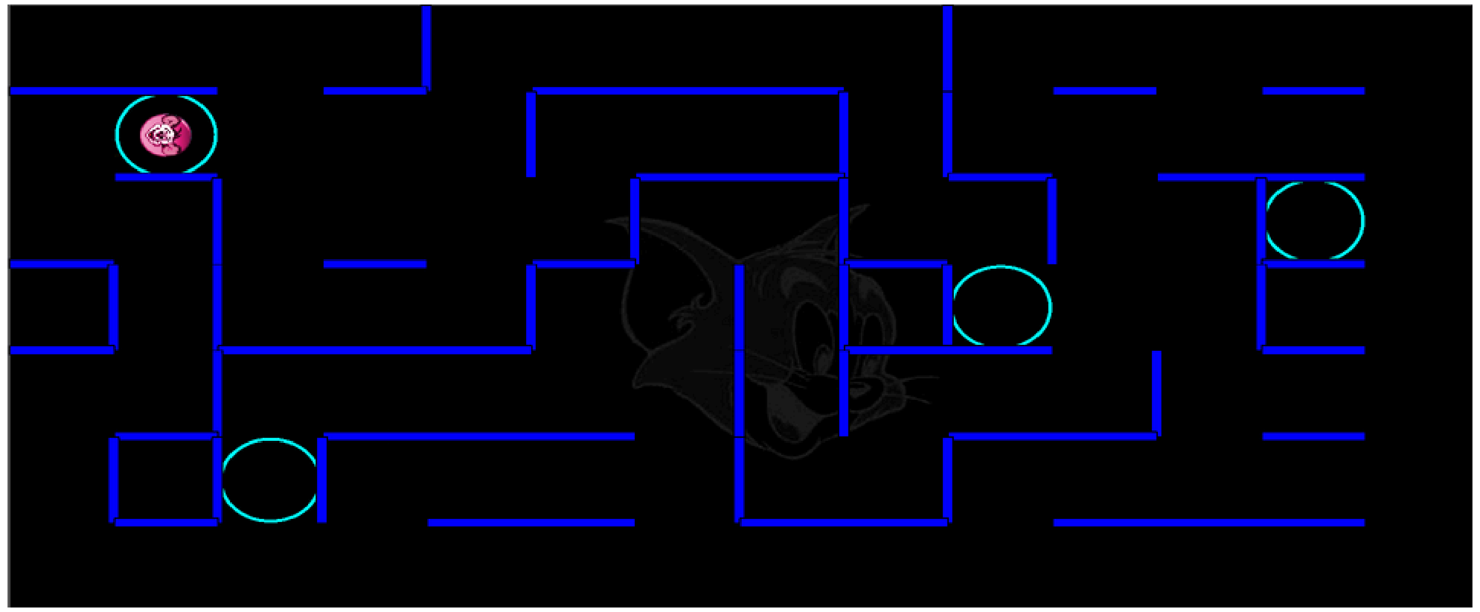
# Planning Path



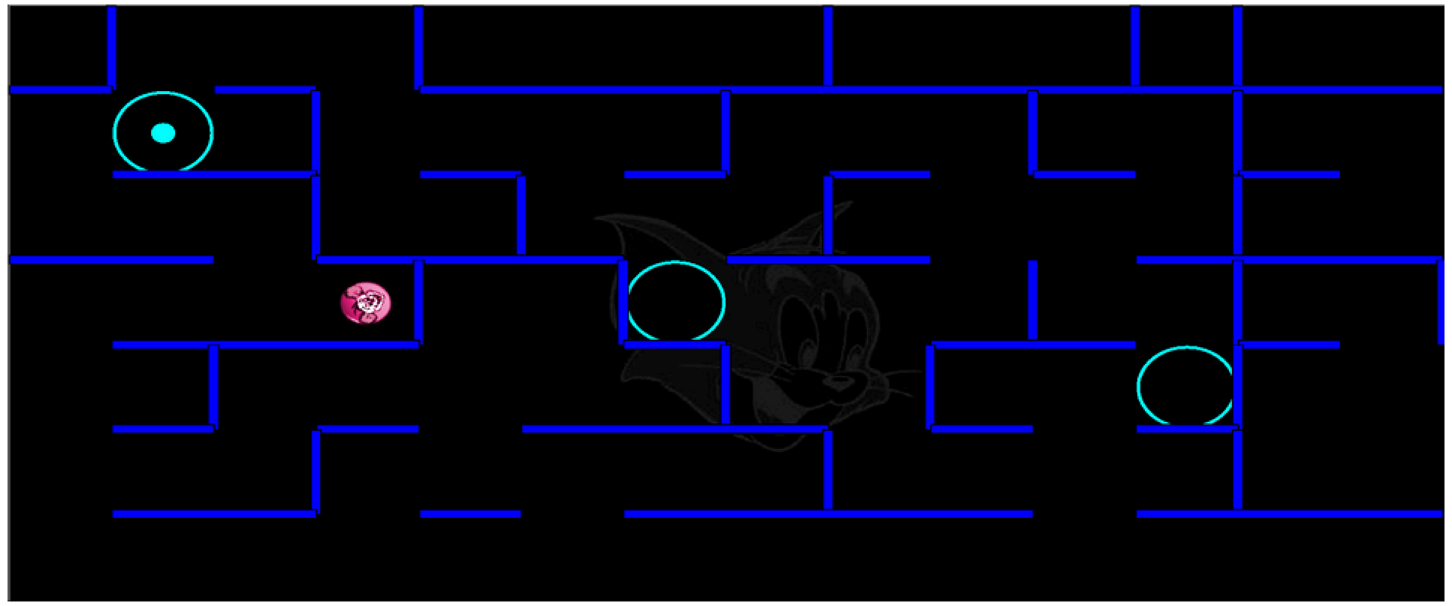
# Test and Results

**Different Map Testing**

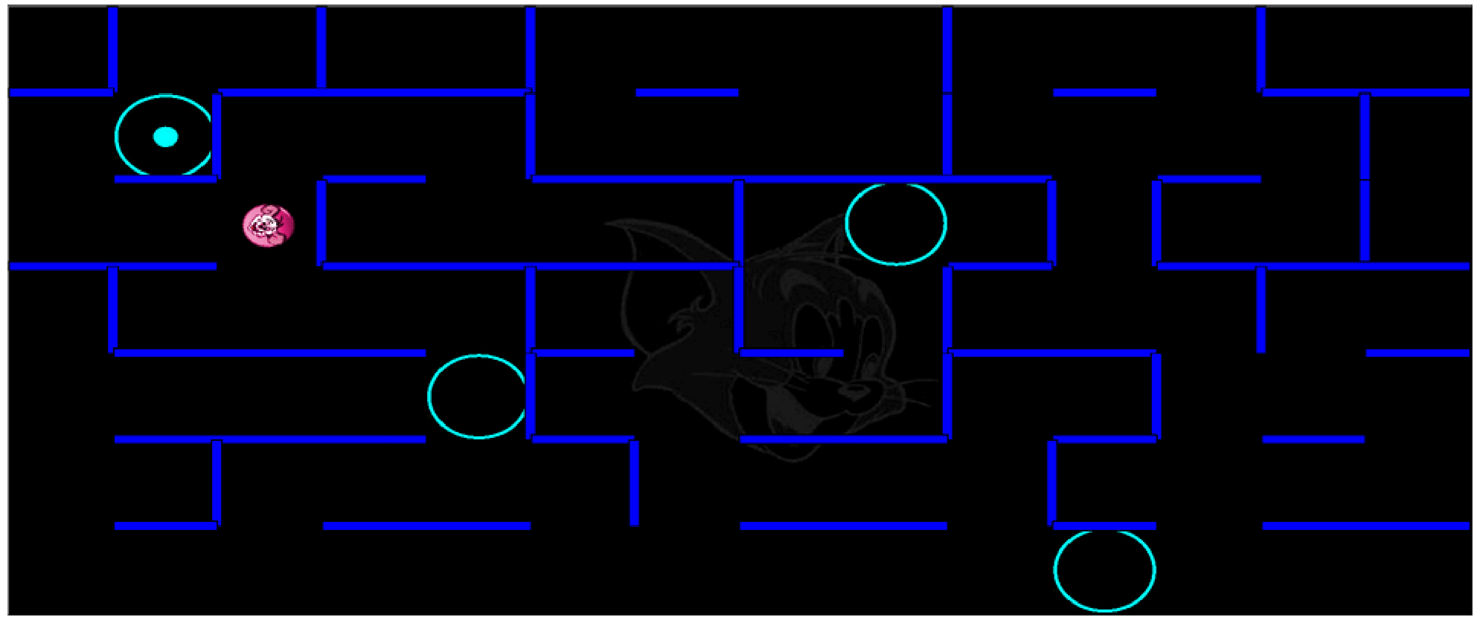
### Map 1 - 3080 Cycles



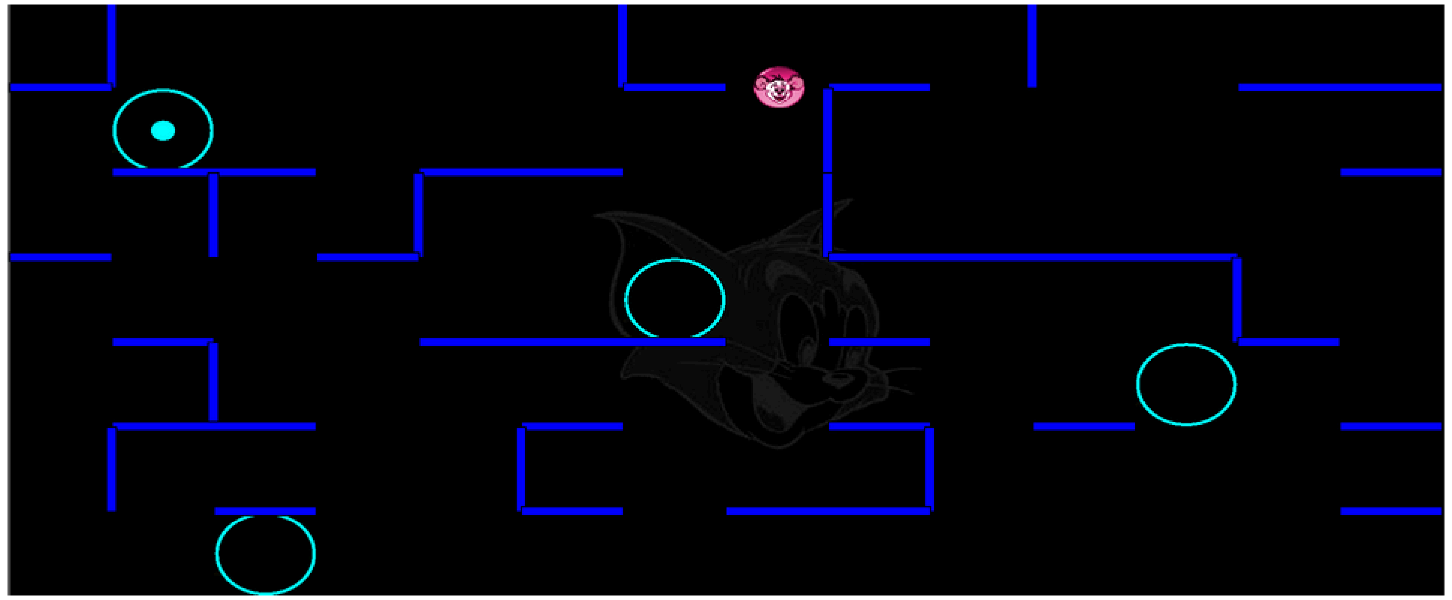
### Map 3 - 3351 Cycles



### Map 2 - 3313 Cycles



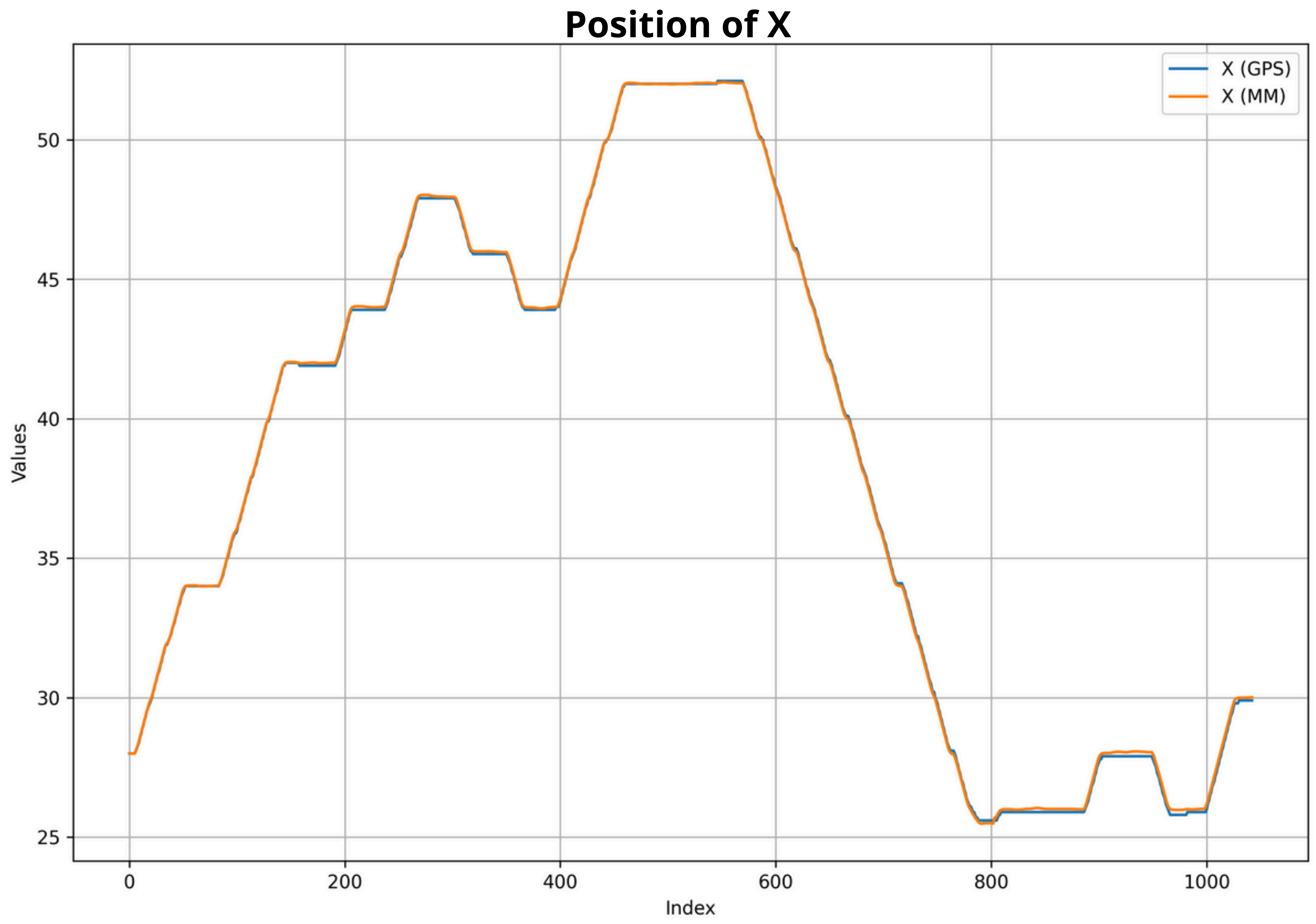
### Map 4 - 3412 Cycles

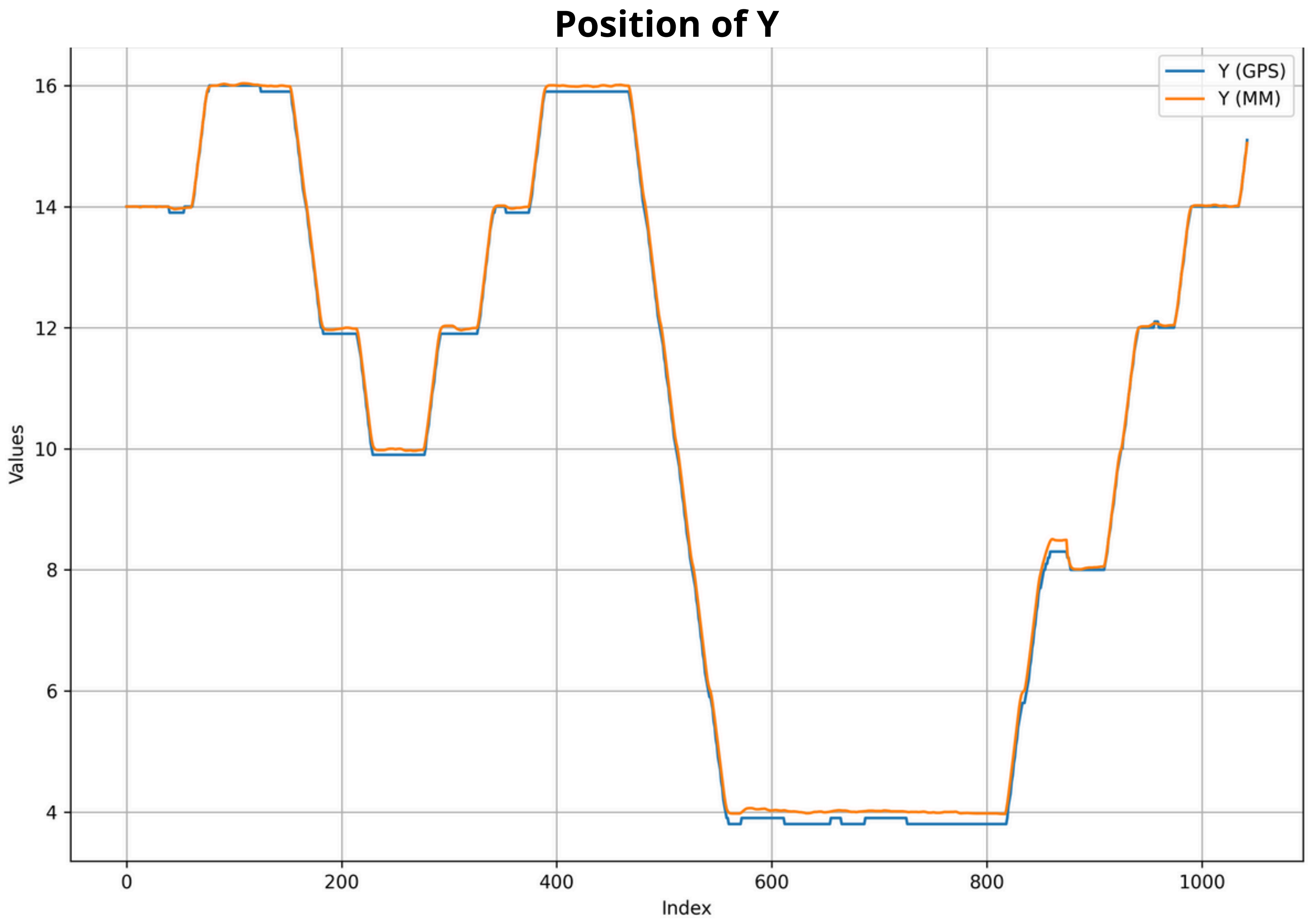


# Test and Results

## Position Comparison

- **Blue line:** x or y position from **GPS**
- **Orange Line:** position from **Movement Model**

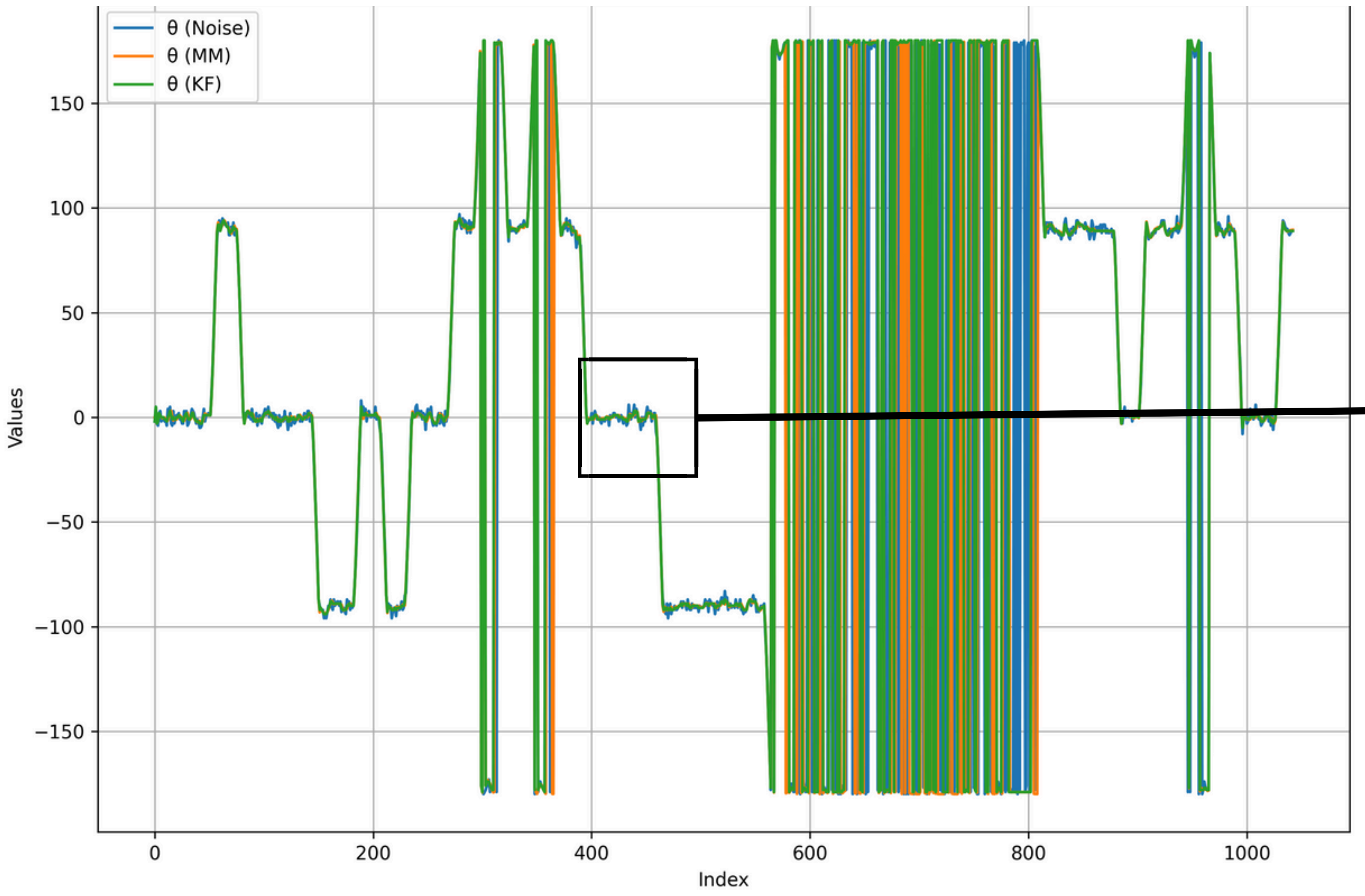


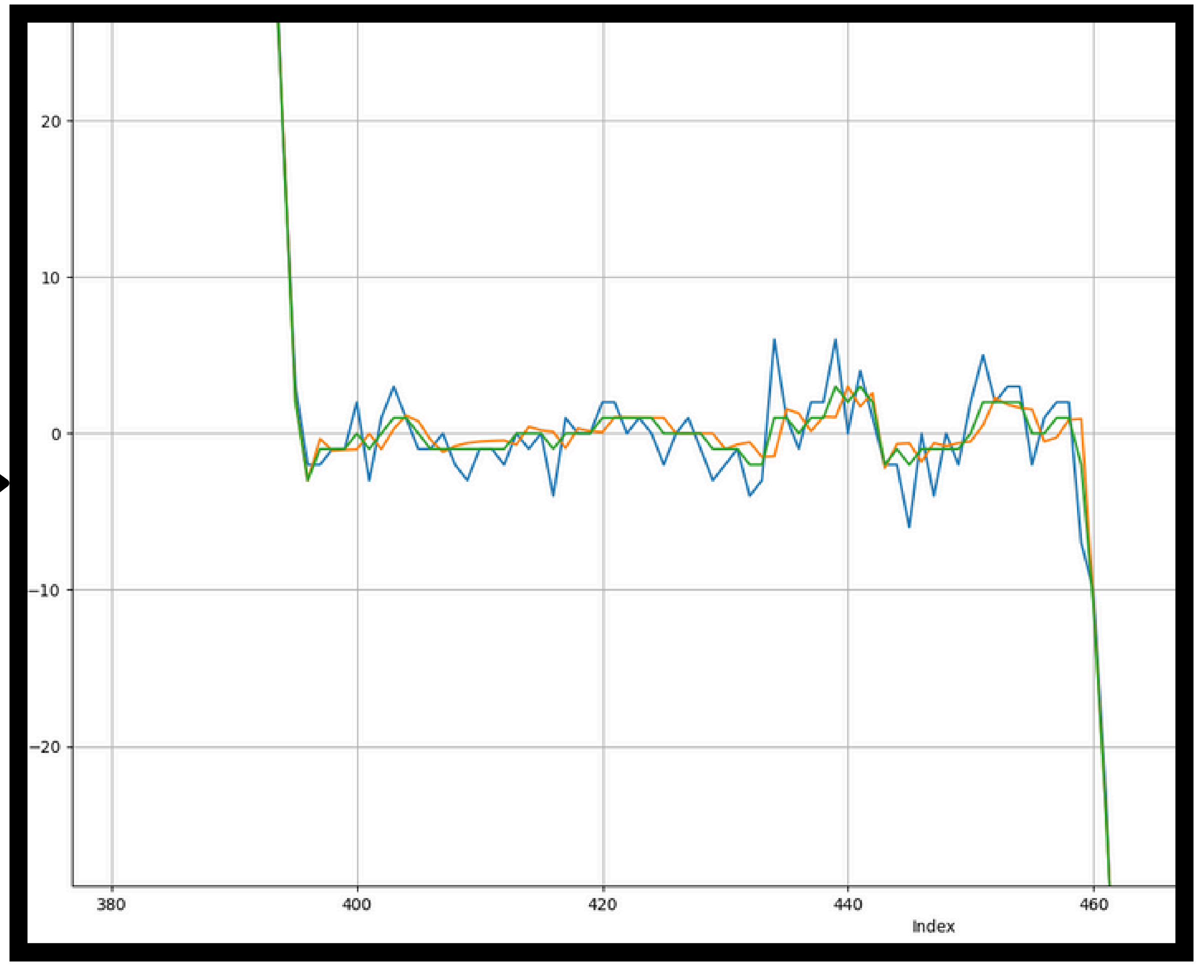



# Test and Results

**Angle Comparison**

- **Blue line:** Angle from **Compass**
- **Orange Line:** Angle from **Movement Model**
- **Green Line: Angle Filtered**



# Test and Results

## Constants Optimization

A total of 370 tests were conducted, each using a different constant value.

**Straight Line Controller:**

- Proportionality constant (KP)
- Derivative Constant (KD)

**Small Adjustments Controller:**

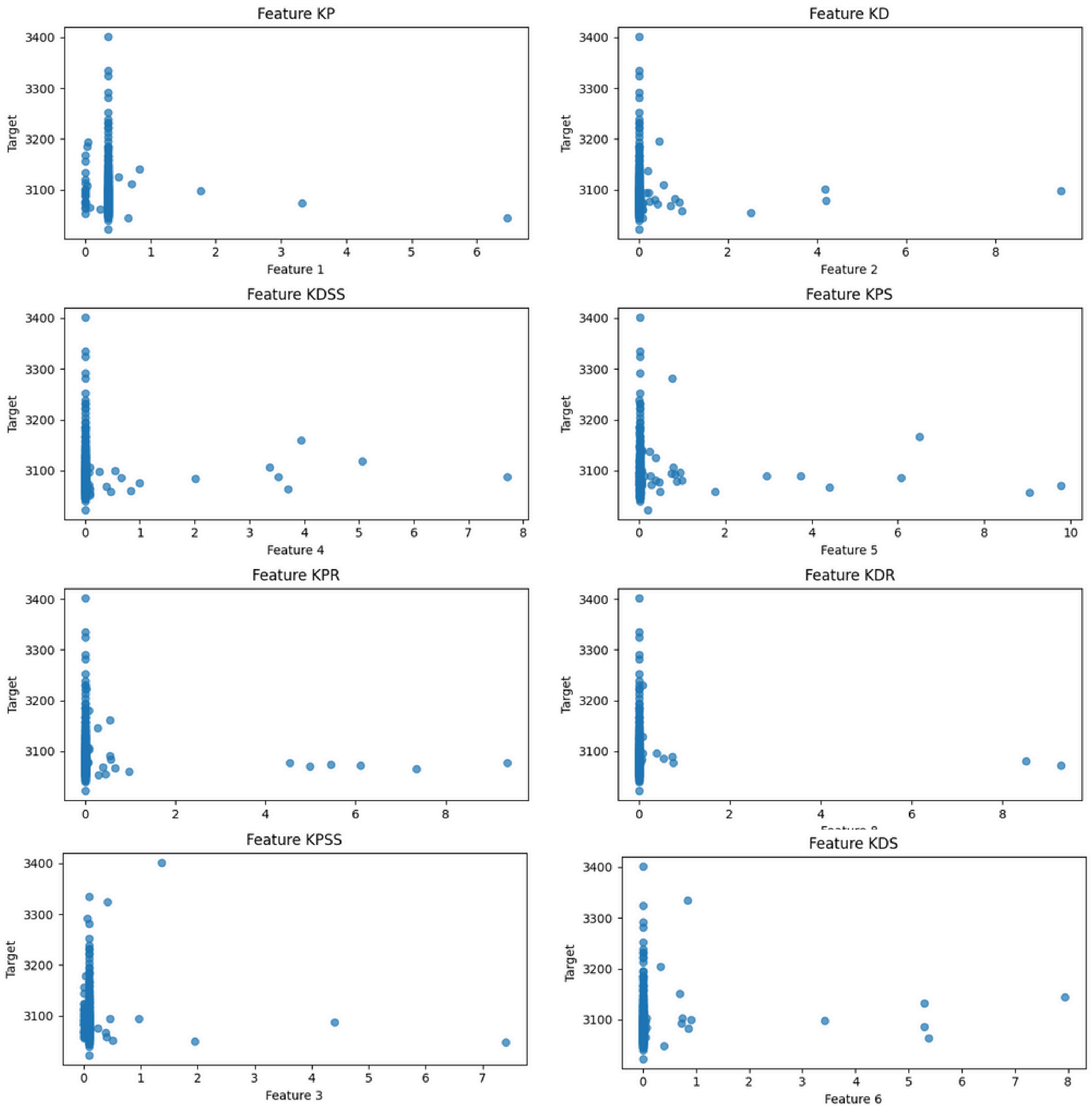
- Proportionality Constant (KPSS)
- Derivative Constant (KDSS)

**Rotation Controller:**

- Proportionality Constant (KPS)
- Derivative Constant (KPS)

**Recalibration Controller:**

- Proportionality Constant (KPR)
- Derivative Constant (KDR)



# Any Questions?