Occupancy Grid Continental Task Ömer Cinar

Generated by Doxygen 1.10.0

Chapter 1

Occupancy Grid Mapping Documentation

1.1 Introduction

This project develops a software in C++ designed to create an occupancy grid, also known as an obstacle map, from detections sourced from ultrasonic sensors mounted on a robot. This map is crucial for robotic navigation and environment interaction, especially in autonomous tasks.

1.2 Project Description

This project constructs an occupancy grid by integrating readings from four ultrasonic sensors positioned at the corners of a robot's chassis, each aligned at 45 degrees with respect to the robot’s axis. These sensors measure the environment within a 30-degree wide cone by emitting sound waves and detecting their echo.

The robot is additionally equipped with a 'magical' odometry sensor that provides precise x- and y-coordinates and orientation angles (in radians), starting from zero upon activation. This data, along with timestamps and sensor readings (time of flight in seconds), is processed from a CSV file to map the surrounding obstacles.

1.3 File Overview

- `main.cpp`: Initializes the data processor and occupancy grid, reads sensor data from a CSV file, and outputs the occupancy grid to a file.
- `dataProcessor.hpp/cpp`: Defines methods for reading the CSV file and updating the occupancy grid based on sensor readings.
- `occupancyGrid.hpp/cpp`: Contains the OccupancyGrid class which manages the grid data structure and performs calculations for mapping.
- `sensor.hpp`: Defines sensor properties including angle offsets and field of view.
- `position.hpp`: Simple structure to represent 2D positions.

1.3.1 DataProcessor Class

- **CSV Data Reading**: The DataProcessor class is responsible for reading a CSV file containing the robot's sensory and positional data. The CSV file includes timestamps, robot coordinates, orientation, and sensor time-of-flight data.
- **Data Parsing and Grid Population**: After opening the CSV file, it iterates over each line, parsing the data into relevant double values. These values are then used to update the occupancy grid based on the robot's position and the sensor data.

1.3.2 OccupancyGrid Class

- **Grid Initialization**: The OccupancyGrid class initializes a 2D vector representing the grid. Each cell within this grid corresponds to a 10 cm x 10 cm area of the physical space the robot navigates.
- **Center Position Calculation**: For each cell in the grid, its center position is calculated using its row and column indices. This helps in determining the exact location of the cell in relation to the robot.
- **Distance and Angle Calculation**: For each cell, the Euclidean distance and the angle relative to the robot's orientation are calculated. These calculations are crucial for determining whether a cell lies within the sensor's effective range and field of view.

1.3.3 Detailed Algorithmic Steps

- **Converting Sensor Time-of-Flight to Distance**: The time-of-flight data from each sensor is converted into distance using the formula distance = (time_of_flight ∗ speed_of_sound) / 2. This conversion is necessary to understand how far away an object is based on the time it takes for the sound to return to the sensor.
- **Iterative Grid Update**: The entire grid is iterated through, cell by cell. For each cell, calculate its center's position relative to the robot. Calculate the distance and angle from the robot to the center of the cell. For each sensor on the robot, adjust the angle considering the sensor's orientation offset and the robot's heading. Determine if the cell lies within the field of view of any sensor by checking if the absolute angle difference is within half the sensor's beam angle. If a cell is within the sensor's range and field of view, and the calculated distance to the cell is less than or equal to the sensor's measured distance, mark it as occupied (1.0).
- **Finalizing the Grid**: After all cells have been evaluated, the grid is saved to a CSV file. This file stores the occupancy status (0 for free, 1 for occupied) of each cell, providing a visual representation of the area around the robot.

1.4 Conclusion

This approach ensures that all areas within the sensors' range are assessed for potential obstacles, providing a comprehensive map of the environment. The resulting occupancy grid is crucial for tasks such as path planning and navigation in robotics. The final grid is outputted as a CSV file, which can be used for further analysis or real-time navigation purposes. You can access the generated CSV file in the `debug` folder.

1.5 Usage

This project uses CMake as its build system. To compile and run the project, follow these steps:

1. Open a terminal in the project's root directory.
2. Create a new directory named `build` and navigate into it: `mkdir build && cd build`
3. Run CMake to generate the build files: `cmake ..`
4. Compile the project: `cmake --build .`
5. Navigate to the `debug` directory: `cd debug`
6. Run the executable with the path to the input CSV file specified: `.\main.exe`

Note: Don't forget to change the directory path of the `robot1.csv` in order to run and compile it successfully.

The output will be an occupancy grid saved in CSV format, representing detected obstacles.

To visualize the grid, you can run ./visu.py in the main directory. This script will generate a heatmap of the occupancy grid using matplotlib.

To run the test cases, after compiling the code navigate to the `debug` directory and run .\runUnitTests.exe`

Note: This project requires a C++17 or later compiler and has dependencies on Google Test, Python 3 or later, and NumPy. Make sure all dependencies are resolved before compiling. The project is configured to use the same runtime library for Google Test as your code to avoid conflicts.

1.6 References

[1] "Occupancy Grids, MathWorks", `https://www.mathworks.com/help/robotics/ug/occupancy-grids.↩ html`

[2] "What is an Occupancy Grid Map?", `https://automaticaddison.com/what-is-an-occupancy-grid-map/`

[3] "Time-of-Flight principle", `https://www.terabee.com/time-of-flight-principle/#:~↩ :text=The%20Time%2Dof%2DFlight%20principle%20(ToF)%20is%20a,being%20reflected%20by%20an%`

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 DataProcessor Class Reference

A class for processing data for an occupancy grid.

```
#include <dataProcessor.hpp>
```

Public Member Functions

- void readCSVAndPopulateData (const std::string &dataFile, OccupancyGrid &grid)

  Read data from a CSV file and populate the occupancy grid.

4.1.1 Detailed Description

A class for processing data for an occupancy grid.

4.1.2 Member Function Documentation

4.1.2.1 readCSVAndPopulateData()

```
void DataProcessor::readCSVAndPopulateData (
             const std::string & dataFile,
             OccupancyGrid & grid )
```

Read data from a CSV file and populate the occupancy grid.

Parameters

dataFile	The name of the CSV file to read from.
grid	The occupancy grid to populate with data.

The documentation for this class was generated from the following files:

- include/dataProcessor.hpp
- src/dataProcessor.cpp

4.2 OccupancyGrid Class Reference

A class representing an occupancy grid for a robot.

```
#include <occupancyGrid.hpp>
```

Public Member Functions

- OccupancyGrid ()

  Construct a new OccupancyGrid object.
- Position calculateCenterPosition (int row, int column)

  Calculate the center position of a cell in the occupancy grid.
- std::pair< double, double > calculateDistanceAndAngle (double robotX, double robotY, double robotTheta, Position cellCenter)

  Calculates the distance and angle from the robot to a cell.
- void updateGrid (double robotX, double robotY, double robotTheta, const double ∗sensorData)

  Update the occupancy grid based on the robot's position and sensor data.
- void saveToFile (const std::string &filename) const

  Save the occupancy grid to a file.

4.2.1 Detailed Description

A class representing an occupancy grid for a robot.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 OccupancyGrid()

```
OccupancyGrid::OccupancyGrid ( )
```

Construct a new OccupancyGrid object.

4.2.3 Member Function Documentation

4.2.3.1 calculateCenterPosition()

```
Position OccupancyGrid::calculateCenterPosition (
            int row,
            int column )
```

Calculate the center position of a cell in the occupancy grid.

Parameters

row	The row index of the cell.
column	The column index of the cell.

Returns

The center position of the cell.

4.2.3.2 calculateDistanceAndAngle()

```
std::pair< double, double > OccupancyGrid::calculateDistanceAndAngle (
            double robotX,
            double robotY,
            double robotTheta,
            Position cellCenter )
```

Calculates the distance and angle from the robot to a cell.

Parameters

robotX	The x-coordinate of the robot's position.
robotY	The y-coordinate of the robot's position.
robotTheta	The orientation of the robot in radians.
cellCenter	The center position of the cell.

Returns

A pair containing the distance and angle from the robot to the cell.

4.2.3.3 saveToFile()

```
void OccupancyGrid::saveToFile (
            const std::string & filename ) const
```

Save the occupancy grid to a file.

Parameters

filename	The name of the file to save to.

4.2.3.4 updateGrid()

```
void OccupancyGrid::updateGrid (
            double robotX,
            double robotY,
            double robotTheta,
            const double * sensorData )
```

Update the occupancy grid based on the robot's position and sensor data.

Parameters

robotX	The x-coordinate of the robot's position.
robotY	The y-coordinate of the robot's position.
robotTheta	The orientation of the robot in radians.
sensorData	An array of sensor readings.

The documentation for this class was generated from the following files:

- include/occupancyGrid.hpp
- src/occupancyGrid.cpp

4.3 Position Struct Reference

A struct that represents a position in 2D space.

```
#include <position.hpp>
```

Public Attributes

- double x

  The x-coordinate of the position.
- double y

  The y-coordinate of the position.

4.3.1 Detailed Description

A struct that represents a position in 2D space.

4.3.2 Member Data Documentation

4.3.2.1 x

```
double Position::x
```

The x-coordinate of the position.

4.3.2.2 y

```
double Position::y
```

The y-coordinate of the position.

The documentation for this struct was generated from the following file:

- include/position.hpp

4.4 Robot Struct Reference

A struct representing a robot.

```
#include <robot.hpp>
```

Static Public Attributes

- static constexpr double robotLength = 0.1

  Length of the robot in meters.
- static constexpr double robotWidth = 0.2

  Width of the robot in meters.

4.4.1 Detailed Description

A struct representing a robot.

4.4.2 Member Data Documentation

4.4.2.1 robotLength

```
constexpr double Robot::robotLength = 0.1  [static], [constexpr]
```

Length of the robot in meters.

4.4.2.2 robotWidth

```
constexpr double Robot::robotWidth = 0.2  [static], [constexpr]
```

Width of the robot in meters.

The documentation for this struct was generated from the following file:

- include/robot.hpp

4.5 Sensor Struct Reference

A struct representing a sensor on a robot.

```
#include <sensor.hpp>
```

Static Public Attributes

- static constexpr double sensorAngleOffset = M_PI / 4

  The angle offset of the sensor in radians.
- static constexpr double sensorAngle = 30 ∗ (M_PI / 180)

  The angle of the sensor's field of view in radians.

4.5.1 Detailed Description

A struct representing a sensor on a robot.

4.5.2 Member Data Documentation

4.5.2.1 sensorAngle

```
constexpr double Sensor::sensorAngle = 30 ∗ (M_PI / 180)  [static], [constexpr]
```

The angle of the sensor's field of view in radians.

This is the angle between the two lines that define the edges of the sensor's field of view. This value is set to 30 degrees (30 ∗ PI/180 radians).

4.5.2.2 sensorAngleOffset

```
constexpr double Sensor::sensorAngleOffset = M_PI / 4  [static], [constexpr]
```

The angle offset of the sensor in radians.

This is the angle between the direction the robot is facing and the direction the sensor is pointing. A positive value means the sensor is pointing to the right of the direction the robot is facing. This value is set to 45 degrees (PI/4 radians).

The documentation for this struct was generated from the following file:

- include/sensor.hpp

Chapter 5

File Documentation

5.1 include/dataProcessor.hpp File Reference

```
#include "OccupancyGrid.hpp"
#include <string>
```

Classes

- class DataProcessor

  A class for processing data for an occupancy grid.

5.2 dataProcessor.hpp

Go to the documentation of this file.

```
#ifndef DATAPROCESSOR_H
#define DATAPROCESSOR_H

#include "OccupancyGrid.hpp"
#include <string>

class DataProcessor {

public:
    void readCSVAndPopulateData(const std::string& dataFile, OccupancyGrid& grid);
};

#endif // DATAPROCESSOR_H
```

5.3 include/mainPageDoxy.h File Reference

5.4 mainPageDoxy.h

Go to the documentation of this file.

```
00001
```

5.5 include/occupancyGrid.hpp File Reference

```
#include <vector>
#include <string>
#include "sensor.hpp"
#include "position.hpp"
```

Classes

- class OccupancyGrid

  A class representing an occupancy grid for a robot.

5.6 occupancyGrid.hpp

Go to the documentation of this file.

```
#ifndef OCCUPANCYGRID_H
#define OCCUPANCYGRID_H

#include <vector>
#include <string>
#include "sensor.hpp"
#include "position.hpp"

class OccupancyGrid {
private:
    const double mapWidth = 10.0; // Adjust as per the room size or area in which the robot operates

    const double mapHeight = 10.0;

    const double gridResolution = 0.1;

    std::vector<std::vector<double» occupancyGrid;

public:
    OccupancyGrid();

    Position calculateCenterPosition(int row, int column);

    std::pair<double, double> calculateDistanceAndAngle(double robotX, double robotY, double
robotTheta, Position cellCenter);

    void updateGrid(double robotX, double robotY, double robotTheta, const double* sensorData);

    void saveToFile(const std::string& filename) const;
};

#endif // OCCUPANCYGRID_H
```

5.7 include/position.hpp File Reference

Classes

- struct Position

  A struct that represents a position in 2D space.

5.8 position.hpp

Go to the documentation of this file.

```

struct Position {
    double x;
    double y;
};
```

5.9 include/robot.hpp File Reference

Classes

- struct Robot

    A struct representing a robot.

5.10 robot.hpp

Go to the documentation of this file.

```

struct Robot {
    static constexpr double robotLength = 0.1;  // 10cm

    static constexpr double robotWidth = 0.2; // 20cm
};
```

5.11 include/sensor.hpp File Reference

`#include <corecrt_math_defines.h>`

Classes

- struct Sensor

    A struct representing a sensor on a robot.

5.12 sensor.hpp

Go to the documentation of this file.

```
#ifndef SENSOR_H
#define SENSOR_H

#include <corecrt_math_defines.h>

struct Sensor {
    static constexpr double sensorAngleOffset = M_PI / 4;

    static constexpr double sensorAngle = 30 * (M_PI / 180);
};

#endif // SENSOR_H
```

5.13 src/dataProcessor.cpp File Reference

```
#include "../include/dataProcessor.hpp"
#include <fstream>
#include <sstream>
#include <iostream>
```

5.14 src/main.cpp File Reference

```
#include "../include/dataProcessor.hpp"
#include "../include/occupancyGrid.hpp"
#include <iostream>
```

Functions

- int main ()

5.14.1 Function Documentation

5.14.1.1 main()

```
int main ( )
```

5.15 src/occupancyGrid.cpp File Reference

```
#include "../include/OccupancyGrid.hpp"
#include "../include/sensor.hpp"
#include <fstream>
#include <cmath>
```