[Date]

Sharnbasava University

DEPAERTMENT OF CSE CO\_ED SUK

**“DEVELOP A 2D OCCUPANCY GRID MAP OF A ROOM USING OVERHEAD CAMERAS”**

**ABSTRACT**

This report addresses the problem of developing a 2D occupancy grid map of a room using overhead cameras. Occupancy grid mapping is a fundamental technique in robotics and computer vision, employed to represent the environment by dividing the space into discrete cells and estimating the occupancy probability of each cell. The primary objective of this project is to utilize overhead cameras to capture and process real-time video data, which will be analyzed to detect and map obstacles and free space within the room. The methodology involves camera calibration, image preprocessing, object detection, and probabilistic mapping algorithms to generate an accurate and dynamic 2D grid representation. This grid map can be instrumental in various applications such as autonomous navigation, space utilization analysis, and smart home systems. The report details the theoretical foundations, technical challenges, implementation steps, and performance evaluation of the developed system, highlighting its effectiveness and potential areas for improvement.

TABLE OF CONTENTS:

|  |  |
| --- | --- |
| Sl no. | Contents |
| 1. | Introduction |
| 2. | Objectives |
| 3. | System Setup |
| 4. | Capturing and processing frame |
| 5. | Visualizing the occupancy grid |
| 6. | Source code |
| 7. | Constants and Initialization |
| 8. | Result |
| 9. | Requirements |
| 10. | References |
| 11. | Conclusion |

**1.INTRODUCTION:**

In the realm of robotics and smart environments, the ability to accurately perceive and interpret spatial surroundings is paramount. One of the fundamental approaches to achieve this is through the creation of occupancy grid maps, which provide a discrete representation of an environment, indicating the presence or absence of obstacles within specified regions. This report delves into the development of a 2D occupancy grid map of a room using overhead cameras, aiming to leverage the vantage point and broad coverage of these cameras to achieve a detailed and accurate mapping.

The primary objective of this project is to harness the capabilities of overhead cameras to capture comprehensive visual data of a room, which is then processed to detect and map obstacles and free space. This process involves several key steps: camera calibration to ensure accurate spatial representation, image preprocessing to enhance the quality of the captured data, object detection to identify obstacles within the room, and the application of probabilistic mapping algorithms to construct the occupancy grid.

**2.** **Objectives**

* Camera Calibration:
* Accurately calibrate the overhead cameras to ensure precise spatial representation and alignment with the physical dimensions of the room.
* Establish a reliable calibration procedure to minimize distortions and errors in the captured data.
* Image Preprocessing:
* Implement effective image preprocessing techniques to enhance the quality of the visual data, including noise reduction, contrast enhancement, and distortion correction.
* Ensure preprocessing methods are efficient and suitable for real-time applications.
* Probabilistic Mapping:
* Apply probabilistic mapping algorithms to generate the 2D occupancy grid, representing the presence or absence of obstacles in discrete cells.
* Ensure the mapping process accounts for uncertainties and variations in the visual data.
* Real-time Processing:
* Design the system to process visual data and update the occupancy grid map in real-time, enabling dynamic and responsive mapping.
* Evaluate and optimize the computational efficiency of the system to achieve real-time performance.

**3.System setup**

* Hardware Setup :-
* Camera for capturing frames
* Computer with python environment
* Software Tools :-
* Python: programming language used for implement.
* Opencv: Library for image processing tasks.
* Numpy: Library for numerical operations.
* Matplotlib: Library for visualizing the occupancy grid.

**4.** **Capturing and processing frameGrid configuration:-**

* + Intialize camera: Use ‘cv2.VideoCapturethe default cameras.
  + Check camera status: verify that the camera opened successfully.
  + Capture frame: Use ‘cap.read( )’ to capture a single frame from the camera.Process the frame: if the frame is captured successfully, pass it to the processing function

**5.** **Visualizing the occupancy grid**

* Displaying the Occupancy Grid:
* plt.imshow(occupancy\_grid, cmap='gray', origin='lower’)
* Adding a Colorbar:
* plt.colorbar(label='Occupancy’)
* Setting Titles and Labels:
* plt.title('Occupancy Grid Map')
* plt.xlabel('Grid cells (X direction)')
* plt.ylabel('Grid cells (Y direction)’)
* Displaying the Grid:
* plt.grid(True)
* Updating the Plot in Real-Time:
* plt.pause(0.001)

**6.** **Source code**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Constants for grid and image processing

GRID\_RESOLUTION = 0.1 # meters per grid cell

GRID\_SIZE\_X = 100 # grid cells in X direction (10 meters)

GRID\_SIZE\_Y = 100 # grid cells in Y direction (10 meters)

# Initialize occupancy grid

occupancy\_grid = np.zeros((GRID\_SIZE\_Y, GRID\_SIZE\_X), dtype=np.uint8)

# Function to process a frame and update the occupancy grid

def process\_frame(frame):

# Convert frame to grayscale

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Apply Gaussian Blur to reduce noise

gray = cv2.GaussianBlur(gray, (5, 5), 0)

# Threshold image to create binary map (occupied vs free)

\_, binary\_map = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY\_INV)

# Resize binary map to match grid size

resized\_map = cv2.resize(binary\_map, (GRID\_SIZE\_X, GRID\_SIZE\_Y))

# Update occupancy grid based on resized binary map

for y in range(GRID\_SIZE\_Y):

for x in range(GRID\_SIZE\_X):

if resized\_map[y, x] > 0:

occupancy\_grid[y, x] = 1 # occupied

else:

occupancy\_grid[y, x] = 0 # free

# Open the camera

cap = cv2.VideoCapture(0) # Use 0 for the default camera, or replace with your camera ID

if not cap.isOpened():

print("Error: Could not open camera.")

exit()

# Create a window for the camera feed

cv2.namedWindow('Camera Feed')

try:

while True:

# Capture a single frame

ret, frame = cap.read()

if not ret:

print("Error: Failed to capture image.")

break

# Process the captured frame

process\_frame(frame)

# Display the frame

cv2.imshow('Camera Feed', frame)

# Display the occupancy grid

plt.clf() # Clear the previous plot

plt.imshow(occupancy\_grid, cmap='gray', origin='lower')

plt.colorbar(label='Occupancy')

plt.title('Occupancy Grid Map')

plt.xlabel('Grid cells (X direction)')

plt.ylabel('Grid cells (Y direction)')

plt.grid(True)

plt.pause(0.001) # Pause to update the plot

# Check for key press to exit

if cv2.waitKey(1) & 0xFF == ord('q'):

break

finally:

# Release the camera and close all windows

cap.release()

cv2.destroyAllWindows()

plt.close()

**7.Constants and Initialization:**

* Grid configuration:-
* ‘GRID\_RESOLUTION = 0.1’(meters per grid cell)
* Each grid cell represents a 0.1-meter square area.
* ‘GRID\_SIZE\_X = 100’(grid cells in X direction)
* ‘GRID\_SIZE\_Y = 100’(grid cells in y direction)
* The grid covers a 10-meter by 10-meter area.
* Data Structure:
* 2D array of size ‘100\*100’
* Initialize occupancy grid with zeros

**8.Result:**

* Description: The occupancy grid map is updated in real-time based on the camera feed.
* The grid map displays occupied (white) and free (black) cells.It provides a visual representation of the spatial occupancy.
* Visualizing the Occupancy Grid:
* The grid map is visualized using Matplotlib with the following features:
* Grayscale Color Map: To distinguish between occupied and free cells.
* Color Bar: Indicates the occupancy status.
* Grid Lines: Helps to identify individual cells.
* Axis Labels and Title: Provides context to the visualization.

**9.Requirements:**

* Hardware Requirements:
* Cameras: High-resolution overhead cameras capable of providing clear images with minimal distortion.
  + Resolution: At least 1080p.
  + Frame Rate: 30 fps or higher.
  + Field of View: Sufficient to cover the desired area.
  + Mounting Equipment: Sturdy mounts to position cameras overhead.
  + Lighting: Adequate lighting to ensure clear images in all conditions.
* Computing Hardware:
* CPU/GPU: Powerful enough to handle image processing and grid map generation in real-time.
* Storage: Sufficient storage for saving video feeds and processed data.
* Network: Reliable network infrastructure for data transmission if cameras are IP-based.
* Software Requirements
* Operating System: Compatible OS for development (e.g., Linux, Windows).
* Programming Languages:
* Primary: Python or C++ for image processing and mapping.
* Secondary: JavaScript or others for UI development if needed.
* Libraries and Frameworks:
* Image Processing: OpenCV, scikit-image.
* Machine Learning: TensorFlow, PyTorch (for any required ML models).
* Mapping: ROS (Robot Operating System) with nav\_msgs/OccupancyGrid for handling occupancy grids.
* Visualization: Matplotlib, RViz (if using ROS), or custom web-based visualizations.
* Development Tools: IDEs like PyCharm, Visual Studio Code, or any other preferred IDE.
* Data Requirements
* Image Data: Real-time video feed from cameras.
* Calibration Data: Camera calibration parameters to correct for lens distortion and to map the image to real-world coordinates.
* Ground Truth Data: For validation and training purposes (if using machine learning).

**10. REFERENCES:**

* [**https://www.w3schools.com/**](https://www.w3schools.com/)
* [**https://github.com/**](https://github.com/)
* [**https://www.geeksforgeeks.org/**](https://www.geeksforgeeks.org/)
* [**https://pypi.org/project/cryptography**](https://pypi.org/project/cryptography)

**11.CONCLUSION:**

* Real-Time Processing: Successfully implemented real-time processing of video feed to update the occupancy grid.
* Effective Visualization: Utilized Matplotlib for clear and informative visualization of the occupancy grid.
* Simplicity and Efficiency: The approach uses basic image processing techniques that are computationally efficient.
* The occupancy grid provides a clear and immediate understanding of the environment.
* It is essential for applications requiring spatial awareness, such as robotics and autonomous navigation.