CS666 – Mobile Robotics

A project report on

PATH PLANNING WITH OBSTACLE DETECTION AN D AVOIDANCE

Project by Group D

Group Members:

Deepak Yadav (234156003)

Madhusudan (234156011)

Mohd Adil Khan (234156012)

Mujahid Husain Badshah (234156013)

Reetu Raj Chauhan (23156028)

Introduction

At the heart of this project lies the seamless integration of path planning algorithms and innovative image processing techniques for the Firebird 6 robot within the Webots simulation environment. Underlining a core emphasis on obstacle detection and avoidance, the project strategically implements a robust image processing module designed to identify objects within a simulated room. Additionally, the pivotal role played by the A-star algorithm in facilitating path planning empowers the Firebird 6 robot to navigate its environment with utmost efficiency, ensuring adept avoidance of detected obstacles.

Objectives

Path Planning with A-star Algorithm:

- Develop a path planning system for the Firebird 6 robot using the A-star algorithm.
- Integrate the path planning module with the Webots simulation environment and GPS (Global Positioning System) coordinate system.

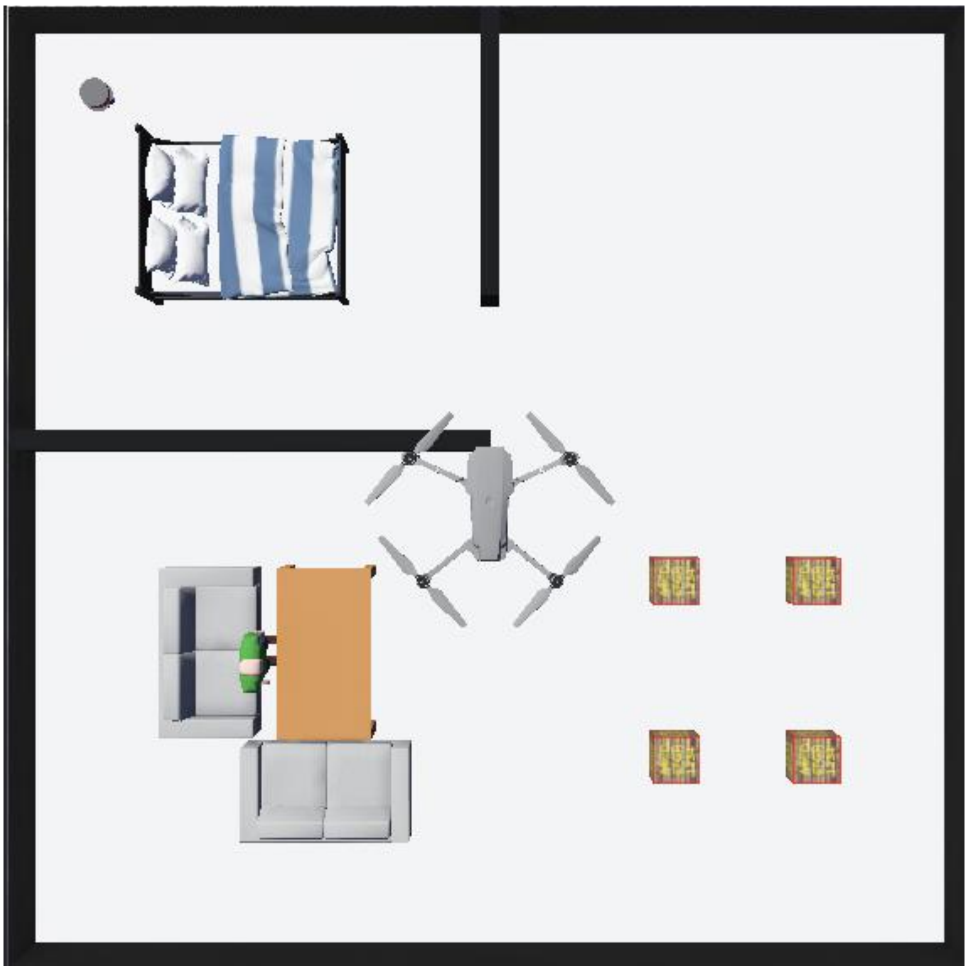
Image Processing Module:

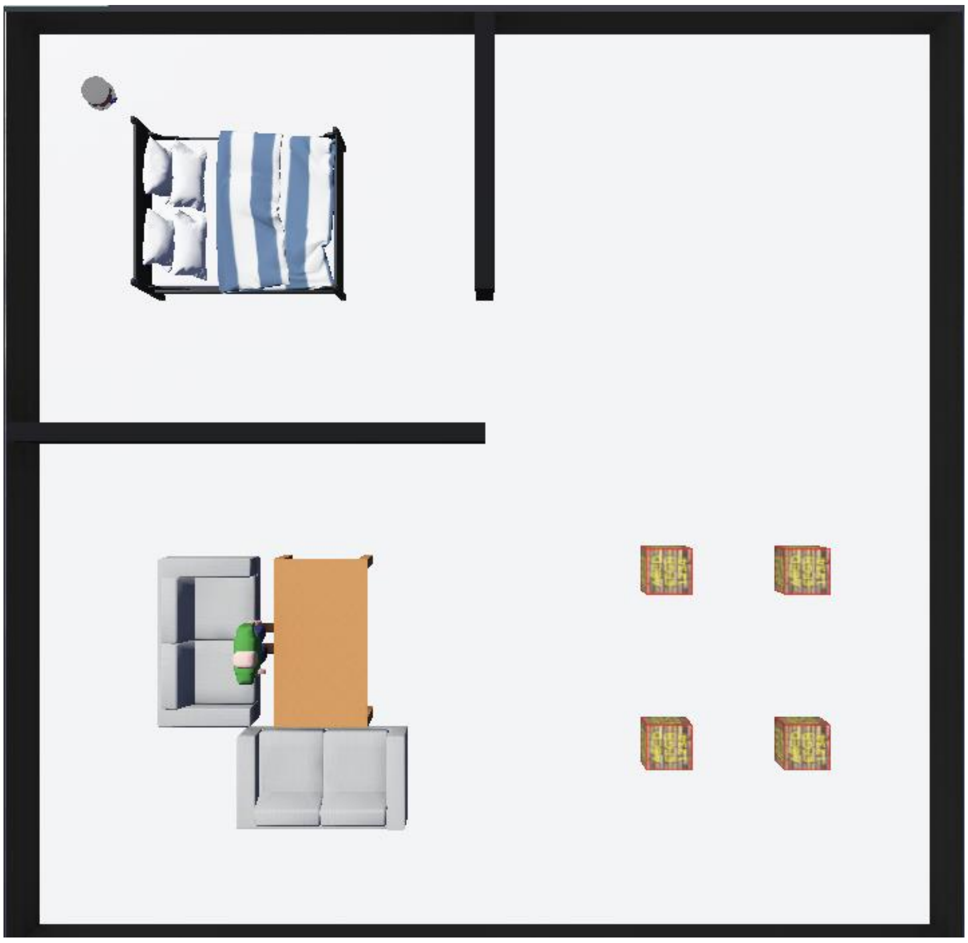
- Capture simulated images of the room using drone DJI MAVIC 2 Pro.
- Image dimension captured by the camera on the Mavic 2 pro is set as 240 x 240 px size.
- A snap of robot is saved in the directory for robot identification and removal from the environment.
- The above step is done to ensure that we do not have the robot's area populated as an obstacle resulting in proper path planning.
- Implement image processing techniques to identify and classify objects present in the captured images.
- Adding padding to the objects.

Environment Setup:

Arena:

- A rectangular arena of size 10 m by 10 m is used with obstacles placed arbitrarily in the arena.
- Walls of height 1 m are placed in the arena to create partition resembling a room.
- A bed, a few sofas, some boxes are added as obstacles in the environment.
- The obstacles are randomly arranged, but the robot's dimension is taken care of while placing it so that the navigation does not face bottlenecks due to its size.
- A drone is placed at some height in the center of the arena which is used to capture the image and use it further.





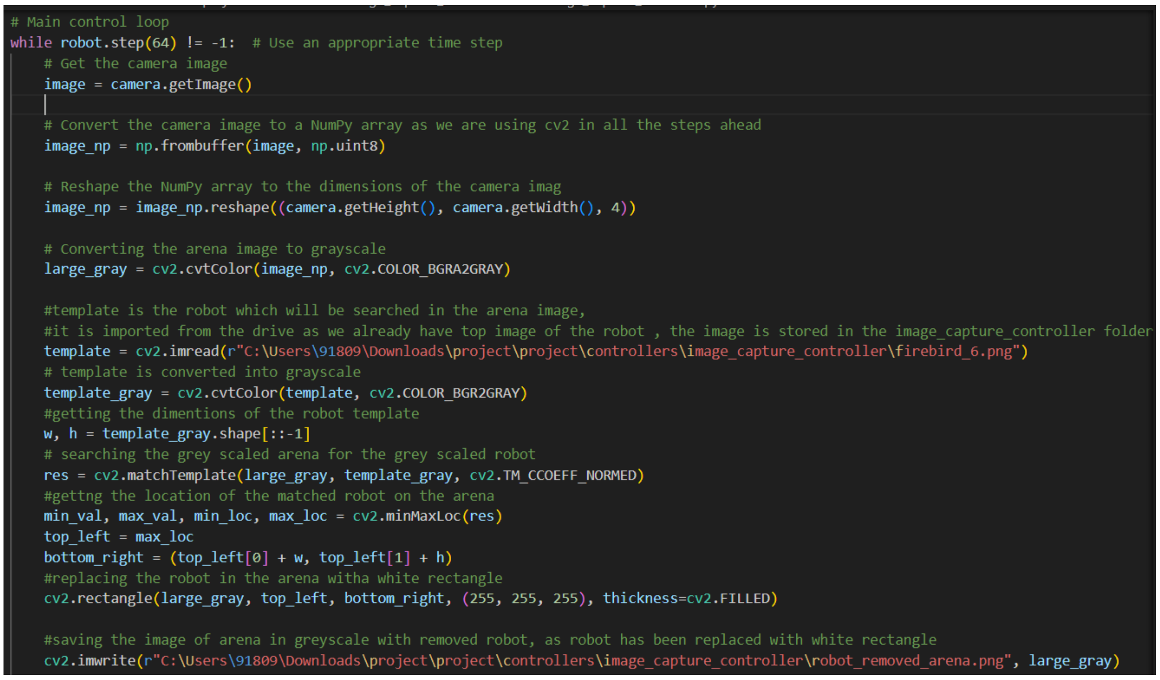
Methodology

Image Processing Module

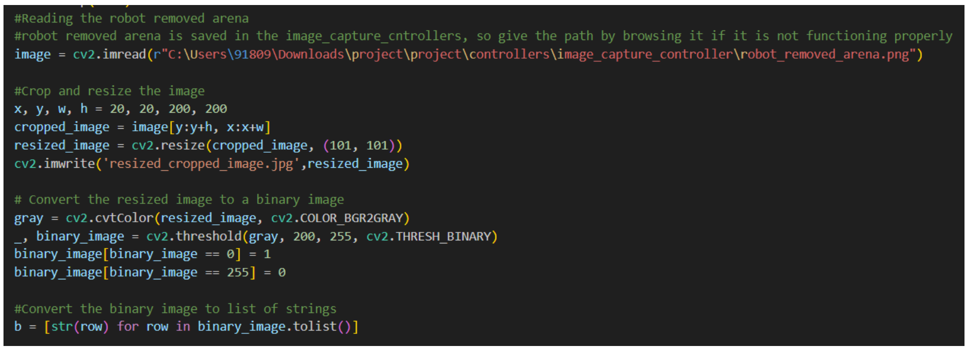
In the initial phase of the project, the image processing component took centre stage. Utilizing the OpenCV library and Python's powerful image processing capabilities, the project began by reading an image file from a specified location. The chosen image, representing a simulated environment, underwent a transformation through cropping and resizing to focus on a specific region of interest. This meticulous adjustment aimed to enhance the efficiency of subsequent processing steps.

The resized image was further processed to convert it into a binary representation. Through grayscale conversion and thresholding techniques, a binary image was obtained, where pixel values denoted the presence or absence of objects. Notably, this binary image was then transformed into a structured list of strings, providing a clear and concise representation of the image's binary pattern.

This pre-processing stage laid the foundation for subsequent tasks, setting the groundwork for the integration of obstacle detection mechanisms and path planning algorithms for the Firebird 6 robot in the Webots simulation environment.

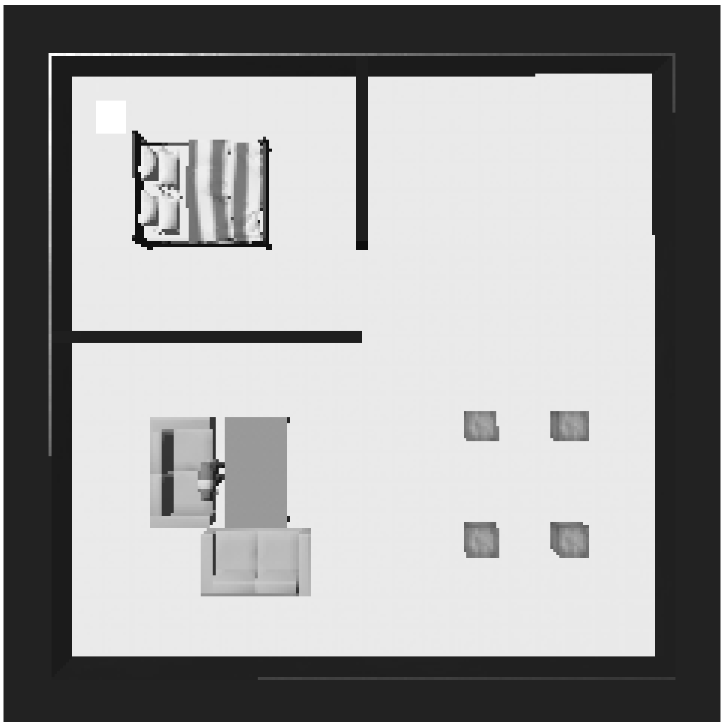


```python
# Main control loop
while robot.step(64) != -1:  # Use an appropriate time step
    # Get the camera image
    image = camera.getImage()

    # Convert the camera image to a NumPy array as we are using cv2 in all the steps ahead
    image_np = np.frombuffer(image, np.uint8)

    # Reshape the NumPy array to the dimensions of the camera imag
    image_np = image_np.reshape((camera.getHeight(), camera.getWidth(), 4))

    # Converting the arena image to grayscale
    large_gray = cv2.cvtColor(image_np, cv2.COLOR_BGRA2GRAY)

    #template is the robot which will be searched in the arena image,
    #it is imported from the drive as we already have top image of the robot , the image is stored in the image_capture_controller folder
    template = cv2.imread(r"C:\Users\91809\Downloads\project\project\controllers\image_capture_controller\firebird_6.png")
    # template is converted into grayscale
    template_gray = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
    #getting the dimentions of the robot template
    w, h = template_gray.shape[::-1]
    # searching the grey scaled arena for the grey scaled robot
    res = cv2.matchTemplate(large_gray, template_gray, cv2.TM_CCOEFF_NORMED)
    #gettng the location of the matched robot on the arena
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
    top_left = max_loc
    bottom_right = (top_left[0] + w, top_left[1] + h)
    #replacing the robot in the arena witha white rectangle
    cv2.rectangle(large_gray, top_left, bottom_right, (255, 255, 255), thickness=cv2.FILLED)

    #saving the image of arena in greyscale with removed robot, as robot has been replaced with white rectangle
    cv2.imwrite(r"C:\Users\91809\Downloads\project\project\controllers\image_capture_controller\robot_removed_arena.png", large_gray)
```



```python
#Reading the robot removed arena
#robot removed arena is saved in the image_capture_cntrollers, so give the path by browsing it if it is not functioning properly
image = cv2.imread(r"C:\Users\91809\Downloads\project\project\controllers\image_capture_controller\robot_removed_arena.png")

#Crop and resize the image
x, y, w, h = 20, 20, 200, 200
cropped_image = image[y:y+h, x:x+w]
resized_image = cv2.resize(cropped_image, (101, 101))
cv2.imwrite('resized_cropped_image.jpg',resized_image)

# Convert the resized image to a binary image
gray = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
_, binary_image = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)
binary_image[binary_image == 0] = 1
binary_image[binary_image == 255] = 0

#Convert the binary image to list of strings
b = [str(row) for row in binary_image.tolist()]
```

Processed Image



Obstacle avoidance capabilities

To enhance the obstacle avoidance capabilities of the Firebird 6 robot within the Webots simulation environment, a crucial step involves the padding of cells surrounding obstacle coordinates in the grid. This proactive measure aims to create a buffer zone around detected obstacles, mitigating the risk of collisions during robot navigation.

Padding Algorithm:

The populate_adjacent_cells function systematically identifies obstacle coordinates within the grid and extends the influence of these obstacles to neighbouring cells. This is achieved by populating adjacent cells in all directions surrounding each detected obstacle.

Directional Considerations:

The algorithm incorporates defined directions for up, down, left, and right, ensuring a comprehensive approach to padding. Additionally, diagonal directions are considered, further fortifying the buffer zone around obstacles.

Padding Strategy:

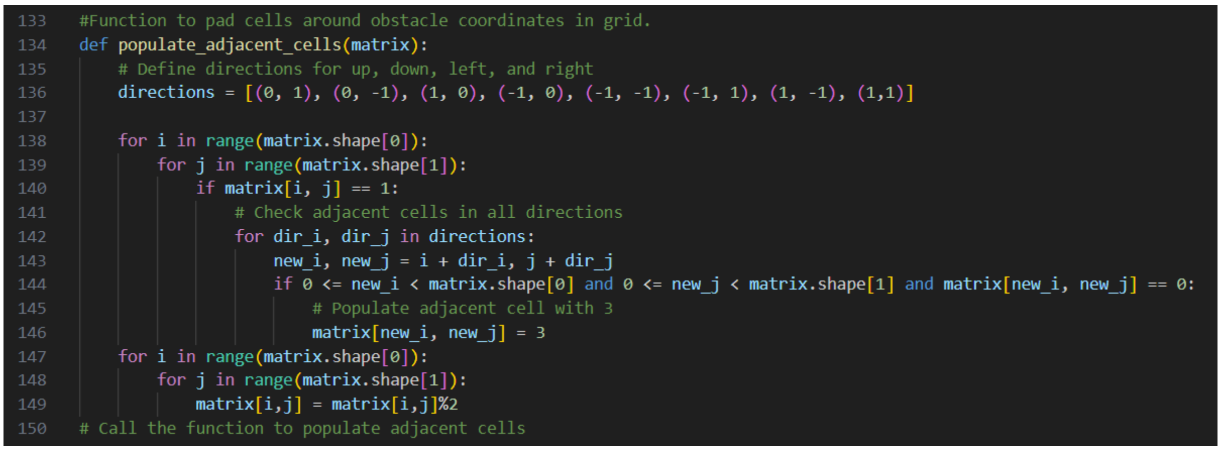
Upon detecting an obstacle at a specific grid coordinate (marked with a value of 1), the function iteratively checks adjacent cells in all directions. If an adjacent cell is within the grid boundaries and is unoccupied (marked with a value of 0), it is populated with a designated value (3 in this case) to indicate the padded region. Finally, to maintain consistency with binary obstacle representation, the values in each cell are modulo-divided by 2.

Practical Implementation:

This algorithmic strategy serves as a pre-emptive measure to prevent collisions. By strategically introducing padded regions around obstacles, the Firebird 6 robot gains an augmented awareness of its surroundings, allowing for safer and more adaptive navigation.

Conclusion:

The successful implementation of the **populate_adjacent_cells** function contributes significantly to the project's overarching goal of enhancing the robot's obstacle avoidance capabilities. This proactive approach to padding ensures a more resilient and intelligent navigation system, aligning with the project's commitment to creating a robust and efficient robotic platform within the Webots simulation environment.



```python
#Function to pad cells around obstacle coordinates in grid.
def populate_adjacent_cells(matrix):
    # Define directions for up, down, left, and right
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0), (-1, -1), (-1, 1), (1, -1), (1,1)]

    for i in range(matrix.shape[0]):
        for j in range(matrix.shape[1]):
            if matrix[i, j] == 1:
                # Check adjacent cells in all directions
                for dir_i, dir_j in directions:
                    new_i, new_j = i + dir_i, j + dir_j
                    if 0 <= new_i < matrix.shape[0] and 0 <= new_j < matrix.shape[1] and matrix[new_i, new_j] == 0:
                        # Populate adjacent cell with 3
                        matrix[new_i, new_j] = 3
    for i in range(matrix.shape[0]):
        for j in range(matrix.shape[1]):
            matrix[i,j] = matrix[i,j]%2
# Call the function to populate adjacent cells
```

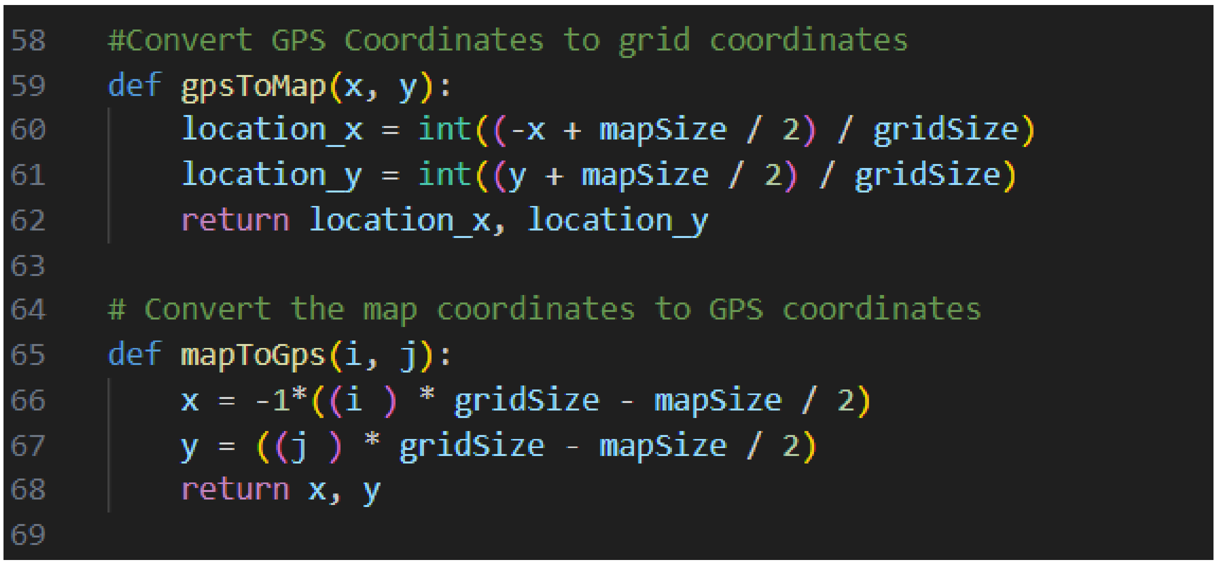
Grid Representation:

To establish a seamless integration between the simulated environment and the Firebird 6 robot's navigation system, a crucial step involved representing the environment as a grid. This grid-based representation facilitated the translation between GPS coordinates and corresponding map coordinates, forming a vital link for path planning algorithms.

Two essential functions were implemented to achieve this bidirectional conversion. The **gpsToMap** function translated GPS coordinates (x, y) into grid coordinates. The translated coordinates were determined by calculating the relative location of the robot within the simulated environment. This process involved mapping the GPS coordinates to their corresponding cell indices on the grid.

Conversely, the **mapToGps** function performed the inverse operation. Given grid coordinates (i, j), representing a cell on the navigable space grid, the function computed the corresponding GPS coordinates. This conversion was crucial for translating grid-based path planning results back into GPS coordinates that the robot could comprehend.

Together, these functions formed a pivotal component in establishing a coherent mapping system, enabling effective communication between the robot's navigation system and the grid-based representation of the simulated environment. This grid-based approach laid the groundwork for subsequent path planning algorithms, ensuring precise and accurate navigation within the designated environment.



```python
#Convert GPS Coordinates to grid coordinates
def gpsToMap(x, y):
    location_x = int((-x + mapSize / 2) / gridSize)
    location_y = int((y + mapSize / 2) / gridSize)
    return location_x, location_y

# Convert the map coordinates to GPS coordinates
def mapToGps(i, j):
    x = -1*((i ) * gridSize - mapSize / 2)
    y = ((j ) * gridSize - mapSize / 2)
    return x, y

```

A-star Algorithm Implementation:

The A-star algorithm was successfully implemented as a critical component of the project, serving as the foundation for efficient path planning for the Firebird 6 robot in the Webots simulation environment. This algorithm plays a crucial role in guiding the robot from its starting position to a specified goal while considering obstacles detected by the image processing module.

Node Class Definition:

The A-star algorithm relies on a Node class, meticulously designed to encapsulate essential attributes. Each node is characterized by its position in the grid (x, y), a cost value, a heuristic value, and a parent attribute. The comparison function (**__lt__**) ensures the proper ordering of nodes when inserted into the priority queue, an integral part of the A-star algorithm.

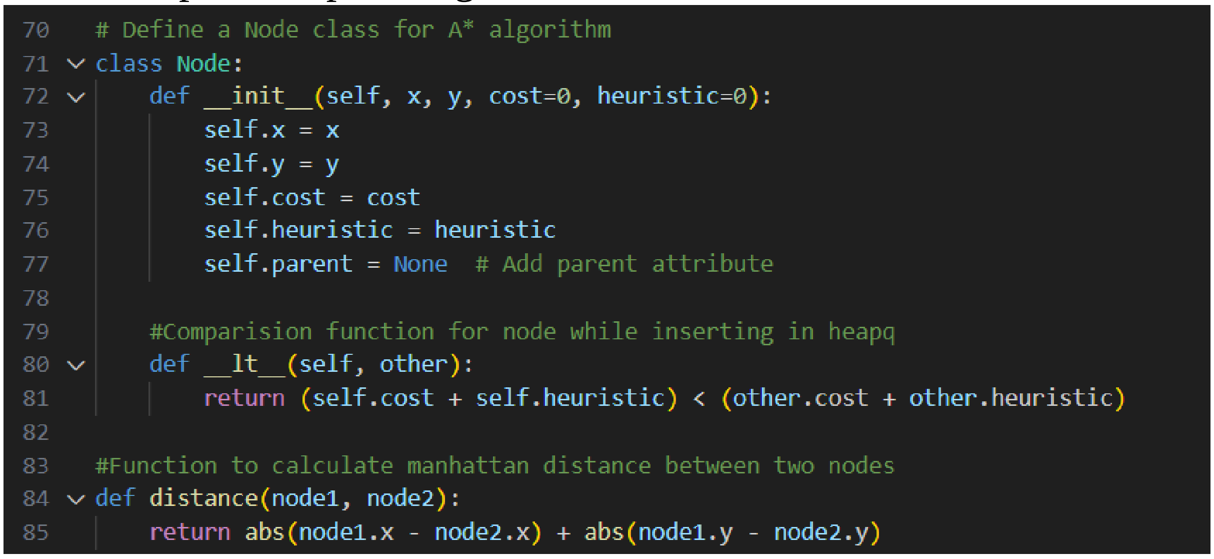
Distance Calculation:

The Manhattan distance function calculates the distance between two nodes, aiding in evaluating the cost of traversing from one node to another. This metric is pivotal in determining the optimal path during the A-star algorithm's execution.

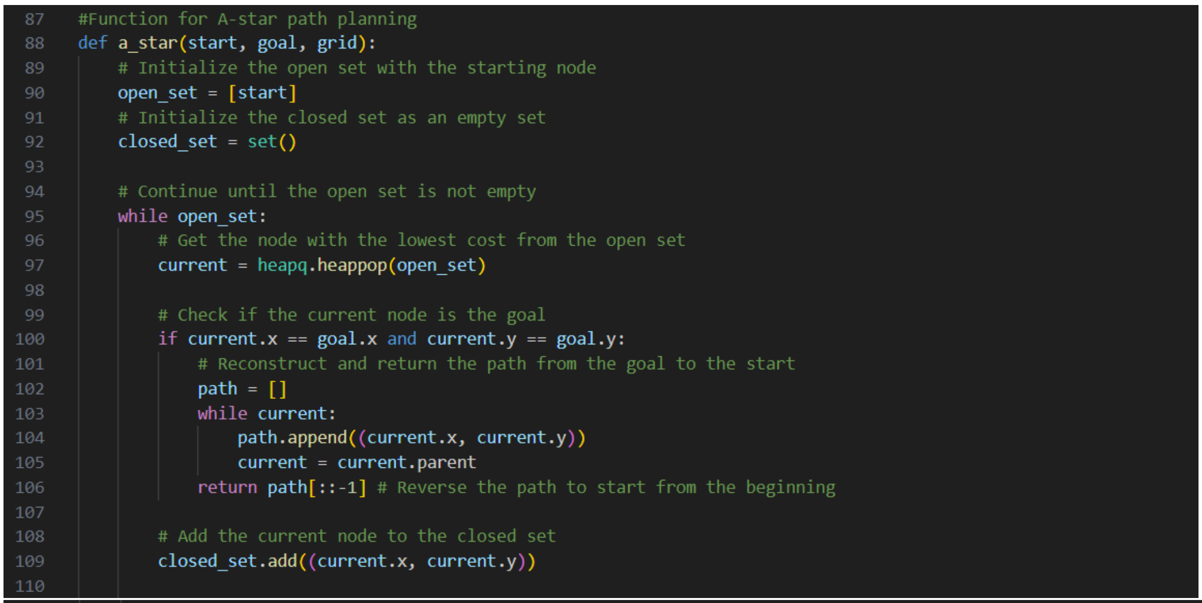
A-star Path Planning:

The A-star path planning function orchestrates the entire algorithm, utilizing a priority queue (open set) and a closed set to explore potential paths. The algorithm continues its execution until the open set is exhausted or the goal node is reached. The process involves evaluating neighbouring nodes, considering grid boundaries, obstacle presence, and updating cost and heuristic values. The resultant path is then reconstructed by backtracking from the goal node to the starting node, ensuring a comprehensive and optimal trajectory.

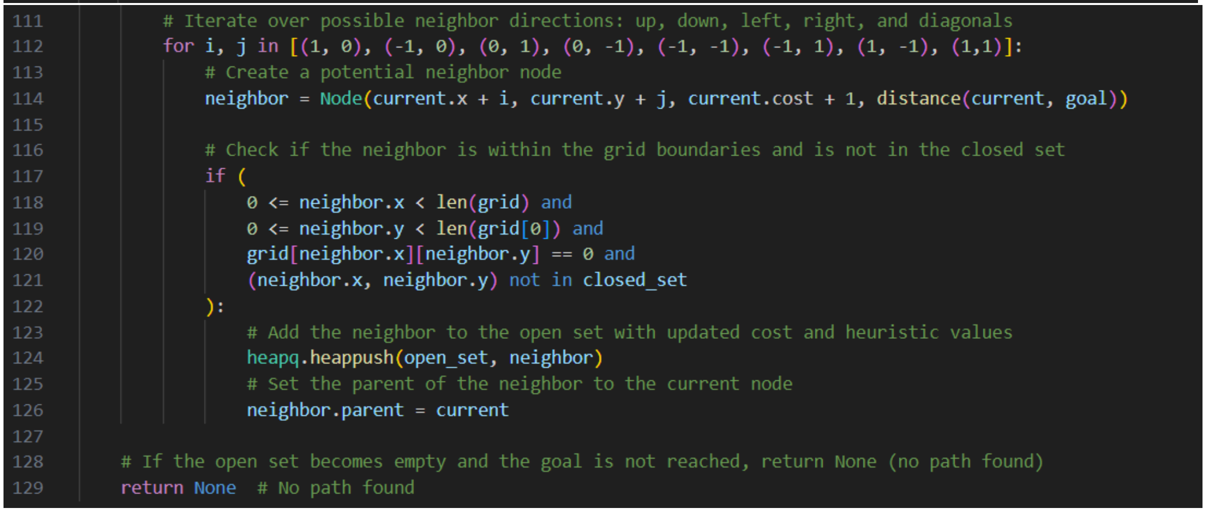
This implementation of the A-star algorithm seamlessly integrates with the project's overarching objectives, providing a robust solution for navigating the Firebird 6 robot within the simulated environment. The consideration of obstacles detected by the image processing module enhances the algorithm's adaptability and reinforces its efficacy in real-world scenarios. The successful implementation of the A-star algorithm constitutes a significant milestone in achieving precise and obstacle-aware path planning for the Firebird 6 robot.



```python
# Define a Node class for A* algorithm
class Node:
    def __init__(self, x, y, cost=0, heuristic=0):
        self.x = x
        self.y = y
        self.cost = cost
        self.heuristic = heuristic
        self.parent = None  # Add parent attribute

    #Comparision function for node while inserting in heapq
    def __lt__(self, other):
        return (self.cost + self.heuristic) < (other.cost + other.heuristic)

#Function to calculate manhattan distance between two nodes
def distance(node1, node2):
    return abs(node1.x - node2.x) + abs(node1.y - node2.y)
```



```python
#Function for A-star path planning
def a_star(start, goal, grid):
    # Initialize the open set with the starting node
    open_set = [start]
    # Initialize the closed set as an empty set
    closed_set = set()

    # Continue until the open set is not empty
    while open_set:
        # Get the node with the lowest cost from the open set
        current = heapq.heappop(open_set)

        # Check if the current node is the goal
        if current.x == goal.x and current.y == goal.y:
            # Reconstruct and return the path from the goal to the start
            path = []
            while current:
                path.append((current.x, current.y))
                current = current.parent
            return path[::-1] # Reverse the path to start from the beginning

        # Add the current node to the closed set
        closed_set.add((current.x, current.y))

```



```python
        # Iterate over possible neighbor directions: up, down, left, right, and diagonals
        for i, j in [(1, 0), (-1, 0), (0, 1), (0, -1), (-1, -1), (-1, 1), (1, -1), (1,1)]:
            # Create a potential neighbor node
            neighbor = Node(current.x + i, current.y + j, current.cost + 1, distance(current, goal))

            # Check if the neighbor is within the grid boundaries and is not in the closed set
            if (
                0 <= neighbor.x < len(grid) and
                0 <= neighbor.y < len(grid[0]) and
                grid[neighbor.x][neighbor.y] == 0 and
                (neighbor.x, neighbor.y) not in closed_set
            ):
                # Add the neighbor to the open set with updated cost and heuristic values
                heapq.heappush(open_set, neighbor)
                # Set the parent of the neighbor to the current node
                neighbor.parent = current

    # If the open set becomes empty and the goal is not reached, return None (no path found)
    return None  # No path found
```

Robot Navigation:

The culmination of the project involved the practical execution of the planned path on the Firebird 6 robot within the Webots simulation environment. This pivotal aspect, denoted by the **navigate** function, seamlessly integrates the A-star algorithm's computed path with the robot's navigation process.

Navigation Process:

The **navigate** function orchestrates the movement of the Firebird 6 robot along the planned path, ensuring precise traversal within the simulated environment. This process involves a series of coordinated actions, including orientation adjustments and forward movements, executed with precision to achieve the specified destination.

Orientation Adjustment:

The function commences by determining the robot's initial heading and computing the angle to the target direction using compass values. A rotation mechanism is then implemented to align the robot with the target angle. The algorithm intelligently selects the shorter rotation direction, optimizing the orientation adjustment process.

Forward Movement:

Following successful orientation, the robot embarks on a forward movement towards the destination. The navigation process continually monitors the robot's position using GPS coordinates, translating them into grid coordinates for enhanced spatial awareness. The function incorporates Euclidean distance calculations to assess proximity to the destination, ensuring the robot halts when within the defined distance threshold.

Iterative Execution:

To enhance the robustness of the navigation process, the function iteratively checks for specific conditions, such as reaching designated waypoints. This iterative approach allows for adaptive responses to varying scenarios, ensuring the Firebird 6 robot successfully navigates the entire planned path.

Reporting and Debugging:

The function incorporates comprehensive reporting mechanisms, providing real-time updates on the robot's position, grid coordinates, and relevant milestones. Debugging information, including the current and target degrees, aids in monitoring and validating the navigation process.

Conclusion:

The successful implementation of the **navigate** function represents a significant achievement, aligning with the project's overarching goals of seamlessly integrating obstacle detection, path planning, and robot navigation within the Webots simulation environment. This critical component contributes to the project's overall success by demonstrating the practical application of the A-star algorithm for precise and adaptive robot navigation.

Challenges Faced

1. Speed and Dynamics Calibration Based on Arena Size

One of the primary challenges encountered during the project involved calibrating the speed and dynamics of the Firebird 6 robot based on the size of the simulated arena. Achieving optimal robot performance required meticulous adjustments to the speed parameters, considering the dimensions of the environment. Striking the right balance between speed and precision was crucial for ensuring efficient path execution within the designated arena.

2. Configuration of Obstacle Resolution in Grid Based on Robot Dimensions

Configuring obstacle resolution within the grid presented a notable challenge, particularly concerning the accurate representation of obstacles based on the dimensions of the Firebird 6 robot. Determining an effective grid resolution that aligns with the robot's physical characteristics was imperative for precise obstacle detection and subsequent path planning. Striving for an optimal configuration demanded a thorough understanding of the robot's geometry and its interaction with the grid.

3. Tackling Erroneous Obstacle Removal Due to Pixelated Image on Resizing

The image processing module faced a challenge related to pixelation when resizing the captured image. This introduced the risk of erroneous obstacle removal, as pixelation could obscure smaller obstacles or create inaccuracies in the representation of the environment. Mitigating this challenge involved implementing strategies to preserve the integrity of obstacle data during image resizing, ensuring a reliable and accurate detection process.

4. Alignment of Image Coordinate System with GPS Coordinate System and Applying A* on That Coordinate System

Integrating data from different coordinate systems posed a substantial challenge. Aligning the image coordinate system with the GPS coordinate system required careful consideration of transformations and calibrations. Applying the A-star algorithm on this unified coordinate system demanded meticulous synchronization to ensure seamless navigation. Overcoming this challenge involved robust coordination between image processing outputs and GPS data.

5. Computation of Distant Path Not Completing as Simulation Exits

The computation of distant paths faced an obstacle in the form of simulation exits. This challenge hindered the completion of path planning for longer distances. This happens due to the limitation of webots simulation as it terminates the controller which takes more than 1 second to generate an output. This problem can be bypassed by dividing the path traversal for distant points in portions.

In overcoming these challenges, the project team demonstrated resilience and problem-solving skills, refining the system for enhanced obstacle detection, path planning, and navigation of the Firebird 6 robot within the Webots simulation environment.

Conclusion

The integration of image processing and path planning modules in a Webots-based environment demonstrates the potential for real-world applications. The successful implementation of object detection and A-star algorithm highlights the feasibility of autonomous navigation for robotic systems in complex environments.

Future Enhancements

Potential future enhancements include:

- **Advanced Object Detection:**
- Implementing deep learning techniques for more robust and accurate object detection.
- **Dynamic Path Planning:**
- Adapting the path planning algorithm to handle dynamic changes in the environment.
- **Multi-Robot Collaboration:**
- Extending the project to support collaboration between multiple Firebird 6 robots.