УДК 519.6

Построение дорожного графа для маршрутизации мобильного робота в замкнутой системе коридоров

Ирванцова Виктория Сергеевна, Лазута Вероника Сергеевна,
студентки 2-го курса кафедры «Робототехнические системы»
Белорусский национальный технический университет, г. Минск
(Научный руководитель – Лебедева Г.И., канд. техн. наук доцент)

Граф — это математическая структура, которая используется для моделирования связей между различными объектами. Граф состоит из вершин и рёбер, которые их соединяют. Роботы, которые требуют предварительного программирования маршрута, обычно относятся к следующим категориям: промышленные роботы, автономные мобильные роботы (AMR), роботы для доставки, роботы-уборщики, садовые роботы, исследовательские роботы.

Программирование робота может служить множеству целей, в зависимости от его назначения и области применения. Вот несколько основных направлений: автоматизация процессов, исследования и разработки, обслуживание и помощь, образование и обучение, развлечение, безопасность и охрана, сельское хозяйство, строительство.

Каждое из этих направлений требует специфического подхода к программированию и проектированию роботов, в зависимости от их функций и условий эксплуатации.

Например:

Построение дорожного графа для маршрутизации робота-пылесоса — это важный этап в разработке его навигационной системы. Ниже описаны основные шаги этого процесса.

Первым шагом в построении дорожного графа является сканирование окружающей среды, в которой будет работать робот. Робот перемещается по помещению и собирает данные о препятствиях, стенах, мебели и других объектах. В процессе сканирования робот фиксирует расстояния до объектов, их размеры и положение. На основе собранных данных создается карта помещения. Это может быть двумерная карта или трехмерная модель.

После создания карты необходимо определить узлы графа, которые будут представлять ключевые точки в пространстве. Узлы могут представлять: перекрестки, целевые точки, объекты:

Далее надо создать рёбра графа. Рёбра представляют собой возможные пути передвижения робота между узлами. Для каждого узла необходимо

определить доступные пути к другим узлам. Это может включать прямые линии между узлами или более сложные маршруты с учетом препятствий. Проанализировав все подходящие элементы, получили дорожный граф для маршрутизации робота:

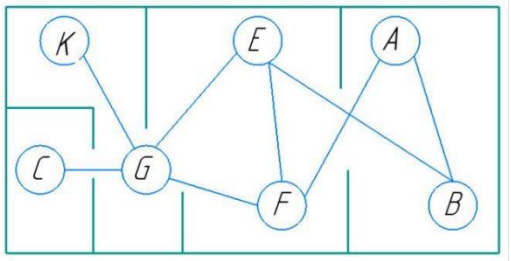

Рисунок 1 – Дорожный граф для маршрутизации робота

На следующем этапе важно учесть динамические и статические препятствия. Наш робот должен иметь возможность обновлять граф в реальном времени. Это может включать удаление рёбер, если путь становится заблокированным, или добавление новых узлов.

Так же есть поставленные задачи маршрутизации. Для решения задачи маршрутизации можно использовать различные алгоритмы, такие как A*, Dijkstra или другие методы поиска пути. Эти алгоритмы анализируют граф и выбирают наиболее эффективный маршрут на основе заранее определенных критериев. Алгоритм учитывает стоимость рёбер и выбирает последовательность узлов, которая обеспечивает наилучший маршрут к цели.

После того как маршрут был спланирован, робот приступает к его выполнению. Робот использует свои сенсоры для навигации по запланированному маршруту. Он должен постоянно отслеживать свое положение и корректировать движение в случае возникновения препятствий или изменений в окружении. Важно обеспечить обратную связь между роботом и системой управления для корректировки маршрута в реальном времени при необходимости. После этого мы получаем код:

```
class Node:
def __init__(self, name):
self.name = name self.neighbors = {} # Словарь: {соседний узел: вес ребра}
def add_neighbor(self, neighbor, weight): self.neighbors[neighbor] = weight
class Graph:
def __init__(self): self.nodes = {} # Словарь: {имя узла: объект Node} def
add_node(self, name):
if name not in self.nodes: self.nodes[name] = Node(name) def add_edge(self,
node1_name, node2_name, weight):
self.add_node(node1_name)
```

```
self.add_node(node2_name) node1 = self.nodes[node1_name]
node2 = self.nodes[node2_name]
node1.add_neighbor(node2, weight)
node2.add_neighbor(node1, weight)
def get_node(self, node_name): return self.nodes.get(node_name)
def __str__(self):
graph_str = ""
for node_name, node in self.nodes.items():
neighbors_str = ", ".join([f"{neighbor.name} ({weight})" for neighbor, weight in node.neighbors.items()]) graph_str += f"Node {node_name}: Neighbors: {neighbors_str}\n" return graph_str
graph = Graph() # Добавляем ребра, указывая примерные веса (расстояния). Важно: эти веса произвольные и должны быть настроены!
graph.add_edge("A", "B", 2)
graph.add_edge("A", "E", 3)
graph.add_edge("B", "F", 4)
graph.add_edge("E", "F", 2)
graph.add_edge("E", "G", 3)
graph.add_edge("F", "G", 1)
graph.add_edge("C", "G", 2)
graph.add_edge("K", "G", 3)
print(graph)
start_node = graph.get_node("A")
if start_node:
print(f"\nNeighbors of node A:")
for neighbor, weight in
start_node.neighbors.items():
print(f"- {neighbor.name} (Weight: {weight})")
else: print("Node A not found in the graph.")
```

После того как робот завершает уборку в целевой области, он может либо вернуться в исходную точку, либо перейти к следующему заданию. Робот может передать информацию о выполненной работе для дальнейшего анализа и оптимизации будущих маршрутов.

Построение дорожного графа для маршрутизации робота-пылесоса — это сложный процесс, который требует интеграции различных технологий и алгоритмов. Каждый этап важен для обеспечения эффективной навигации и выполнения задач по уборке.