Обзорная часть ВКР

Отчет по учебной практике. Кирбаба Денис, группа R3438

Введение

В последнее время роботы стали важной частью нашей жизни. Они могут использоваться для различных целей в различных областях, чтобы облегчить жизнь человека.

Самым простым вариантом внедрения роботов для выполнения задач является использование стационарных роботов. Основываясь на требованиях высокой надежности, они статичны, неинтерактивны и работают в неизменяющихся условиях вдали от людей. Работа данных роботов довольна предсказуема и управление ими состоит в синтезе заранее спланированных задач.

Возможность работы роботов в условиях динамического окружения (взаимодействие с людьми, другими роботами) имеет большой потенциал для повышения эффективности их работы в различных областях. При работе мобильных роботов задача навигации имеет важное значение. Роботу необходимо обрабатывать информацию об окружении и стоить верные безопасные маршруты в реальном времени для достижения заданных целей.

Так как мобильный робот функционирует в динамической среде в непосредственной близости от людей, оборудования и других объектов, робот должен достаточно быстро реагировать на изменения и не допускать возникновение критических ситуаций. Для реализации такой высокоуровневой стратегии управления используются алгоритмы принятия решений.

Основная задача данной работы - разработка алгоритма принятия решения для задачи навигации мобильным роботом при функционировании в динамическом окружении. Разработанный алгоритм должен обеспечивать безопасное и эффективное поведение робота.

Рисунок 1 – Мобильные роботы, доставляющие детали на линию сборки автомобилей (Automated Guided Vehicle system at the automobile manufacturer Porsche Profile - MLR System GmbH (rofa-group.com))

Описание окружения

В нашей задаче окружение является динамическим, неопределенным и нагроможденным из-за присутствия людей и других мобильных объектов (другие мобильные роботы, движущиеся объекты). Движение людей и других динамических объектов не ограничено и является неопределенным.

Требования к разрабатываемому мобильному роботу

1. **Обеспечение безопасности человека** - робот должен иметь механизмы, обеспечивающие безопасность людей и другого оборудования в окружении. Для этого необходимо обнаружить препятствия/людей, присутствующих в окружении, и соответствующим образом изменить поведение/планы робота
2. **Автономная навигация в динамическом окружении** - перед выполнением любой задачи робот должен быть способен автономно перемещаться в изменяющихся условиях. Для этого необходимо:
    a. Обнаруживать препятствия, вести карту окружения и оценивать состояние робота
    b. Динамическое создание эффективных траекторий движения: робот должен уметь генерировать маршруты, которые являются эффективными, соответствуют временным ограничениям задачи и способен планировать перемещение цели (движущийся человек). Для этого необходимо будет отслеживать динамические цели.
    c. Отслеживание траекторий: мобильный робот, используя существующие алгоритмы управления, могут отслеживать желаемый план движения

Планирование пути

Планирование – поиск последовательности конфигураций робота для передвижения его из точки A в точку B.

Планирование пути – геометрический путь, в то время как планирование траектории – геометрический путь + временная информация (например, скорость).

При расчете траектории необходимо учитывать следующее:

1. Физические ограничения робота, количество степеней свободы и его габариты
2. Неопределенности при работе сенсоров (зашумленная или неполная информация) и при передвижении
3. Динамическое окружение
4. Требования к алгоритму
    a. Оптимальность (нужно найти кратчайший путь или мы будем минимизировать другой функционал)
    b. Вычислительная сложность алгоритма и затраты на память (эти пункты важны при вычислениях непосредственно на МК робота)
    c. Полнота алгоритма:
        i. По вероятности – вероятность верного решения алгоритма увеличивается с течением работы (асимптотически) – sample-based algorithms
        ii. По величине дискретизации – grid-based planners – если период дискретизации окружения достигает некоторого значения, то мы сможем найти решение
    d. Online (динамически меняем план работы) или offline (заранее вычисляем путь – работает только в статических хорошо определенных окружениях
    e. Поиск пути или траектории (если учитываем динамику робота)
    f. Точность решения

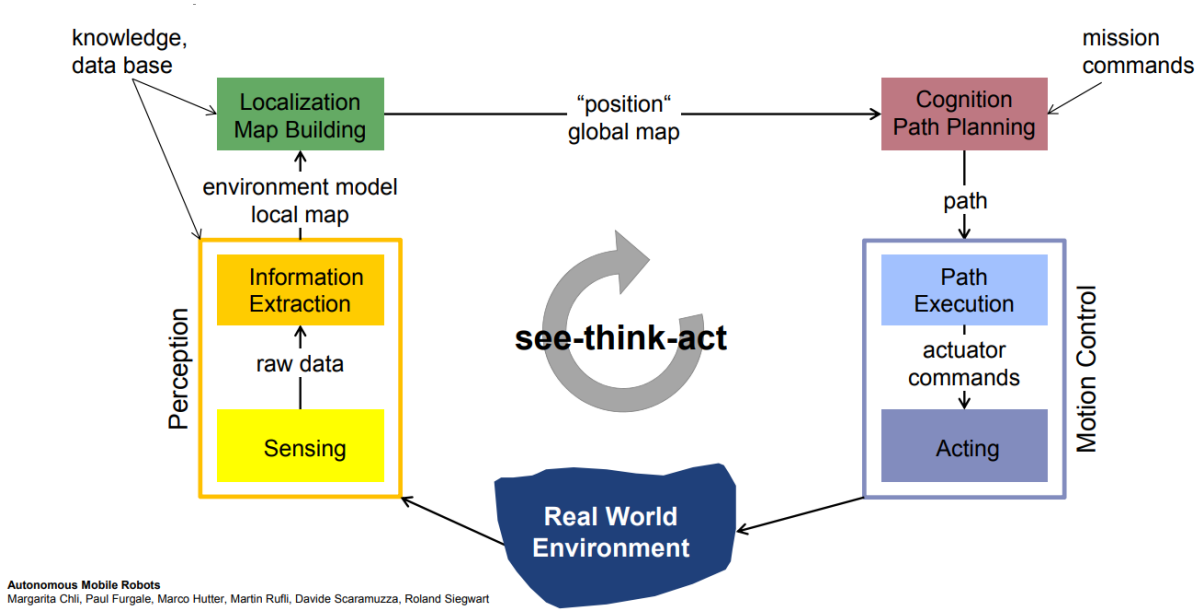

Рисунок 2 – Диаграмма работы автономного мобильного робота

В условиях неизвестной среды, мобильный робот должен одновременно производить свою локализацию и генерацию карты местности, так называемый SLAM.

Задачу планирования траектории можно разбить на слои.

- Верхний слой использует карту местности и данные с сенсора для синтеза команд, определяющих куда необходимо двигаться роботу.
- Нижний (local planner) слой отвечает за подачу команд непосредственно на двигатели, учитывая новую информацию с датчиков, таким образом генерируя корректное управление для выполнения целей поставленных global planner и объезда препятствий.

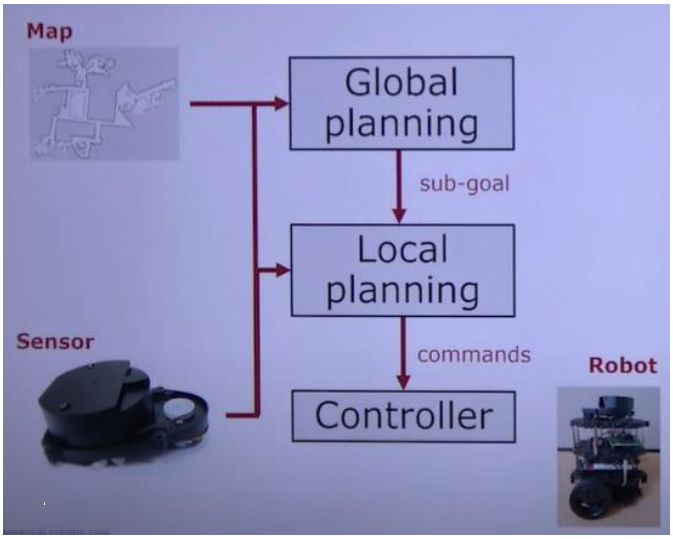

Рисунок 3 - Декомпозиция планировщика пути

Верхний слой работает на низкой частоте (порядка секунд), в то время как нижний (локальный) работает на более высокой (возможно даже на частоте сенсора).

Математическое описание планирования

Конфигурация робота – минимальное множество параметров, которые определяют степени свободы робота, относительно фиксированной системы координат. В нашей задаче конфигурация будет состоять из 3-х параметров:

$$q = (x, y, \theta)$$

Конфигурационное пространство (C-space) – пространство всех возможных конфигураций робота. В нашей задаче

$$C - space: R^2 \times SO(2D - rotations)$$

Рабочее пространство (Workspace) – набор точек, в которых робот может находиться.

Конфигурационное пространство логично разбить на 2 непересекающихся пространства

$$C_{free} = C / C_{obs}, \qquad C_{obs} = \{A(q) \cap O \neq \emptyset\},$$

где $W = R^m$ – рабочее пространство, $O \in W$ – множество точек, занятых препятствиями, и $A(q)$ – положение робота при конфигурации q.

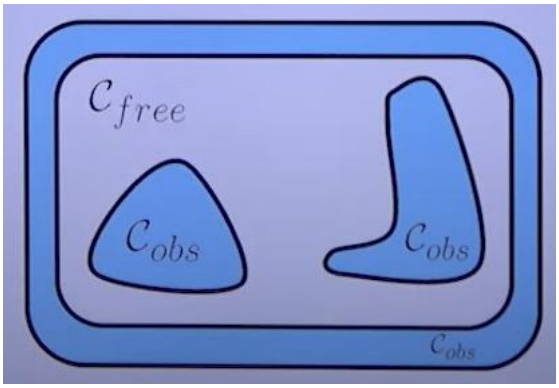

Рисунок 4 - Случай разбиения конфигурационного пространства

Итак, задача планирования заключается в поиске непрерывного пути

$$\tau: [0,1] \to C_{free}, \qquad \tau(0) = q_{start}, \qquad \tau(1) = q_{goal}$$

Виды алгоритмов планирования

1. Геометрические: visibility graphs, cell decomposition, Voronoi diagrams
2. Основанные на потенциальном поле: waveform planner, navigation function
3. Search-based: Dijkstra, A*, D*
4. Sampling-based: RRT, RRT*, PRM, BIT
5. Trajectory: minimum time/energy
6. Bioinspired: NN, genetic algorithms

Геометрические алгоритмы

Общая идея: Построение дорожной карты (roadmap) пространства C_{free}, где вершинами будут служить конфигурации в C_{free}, а ребрами – беспрепятственные пути в C_{free}. После построения данного графа применяется search-based алгоритм для нахождения кратчайшего пути.

Visibility graph

Алгоритм построения:

1. Представляем препятствия в виде многоугольников
2. Соединяем все «прямые» вершины многоугольников (включая смежные вершины) – робот может передвигаться вдоль данных ребер
3. Ищем минимальный путь в построенном графе

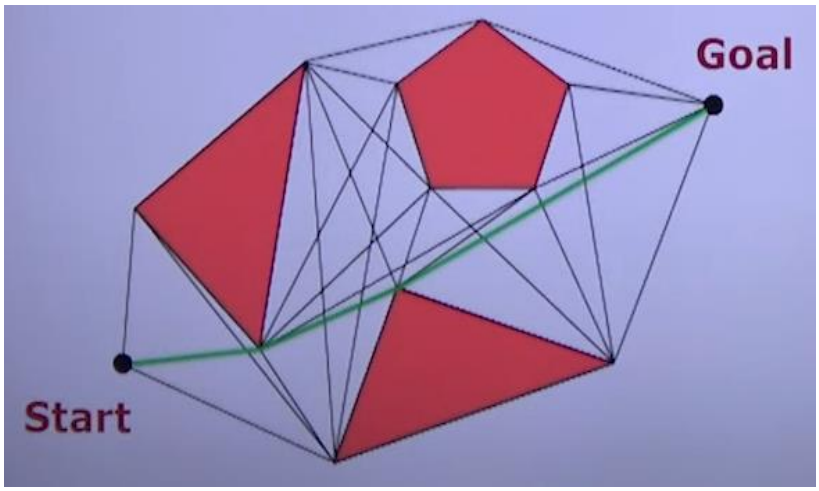

Рисунок 5 - Граф видимости

Однако в данном методе мы имеем проблему прохождения робота вблизи препятствий (а если мы имеем еще некоторый шум при подаче управления на двигатели, то может произойти столкновение). Эту проблему можно избежать, используя диаграмму Вороного.

Generalized Voronoi Diagram

Алгоритм построения:

1. Представляем препятствия в виде многоугольников
2. Соединяем точки, которые находятся на одинаковом расстоянии (equidistant) от двух ближайших препятствий, включая границы рабочего пространства
3. Ищем минимальный путь вдоль полученных ребер

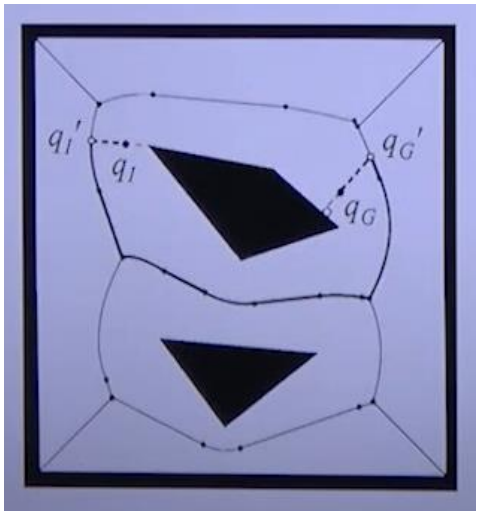

Рисунок 6 - Диаграмма Вороного

Минусы данного алгоритма:

1. Сложность вычисления при увеличении размерности конфигурационного пространства
2. «Консервативный» путь
3. Небольшие изменения в окружении приводят к сильным изменениям пути
4. Сложность при работе с сенсорами малой дальности действия (невозможность провести SLAM)

Potential Field Methods

Общая идея: представляем робота как материальную точку, которая находится под влиянием потенциального поля, которое двигает робота в направлении целевой точки.

Потенциальное поле (дифф. функция): $U: R^m \to R$, имеет две составляющие «притягивающий» к цели потенциал $U_{att}(q)$ и «отталкивающий» от препятствий $U_{rep}(q)$.

$$U(q) = U_{att}(q) + U_{rep}(q)$$

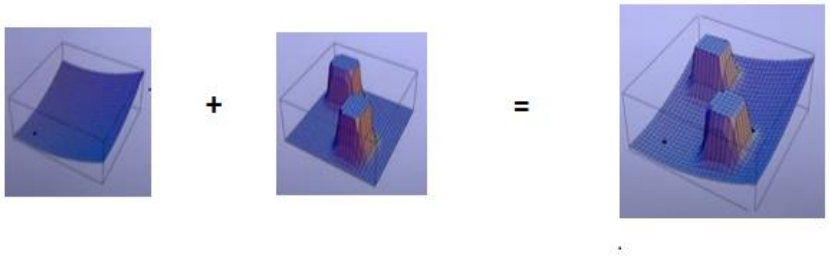

Рисунок 7 - Пример построения потенциального поля с двумя препятствиями

Потенциал можем выбрать в виде расстояния робота в каждой точке до целевой точки.

Соответственно траекторию движения можно найти, следуя за антиградиентом, используя градиентный спуск:

$$F(q) = -\Delta U(q)$$

Свойства алгоритма:

1. Возможность динамического избегания препятствий
2. Возможно попадание в локальный минимум
3. Необходимо явно промоделировать C_{obs} и C_{free}

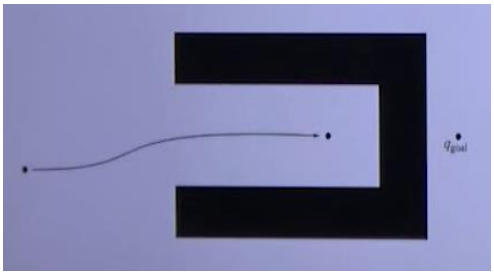

Рисунок 8 - Пример попадания в локальный минимум

Search-based Methods

Данные алгоритмы применяются только к дискретным пространствам. То есть необходимо производить дискретизацию пространства.

Решение с использованием графа (Graph-based)

Алгоритм:

1. Составляем граф $G = (V, E)$, где V – набор вершин (конфигурации), E – набор ребер (беспрепятственные пути в C_{free}).
2. Затем применяем search-based алгоритм для поиска пути (Depth-first, breadth-first, Dijkstra, A*).

Решение с использованием сетки (Grid-based)

В данном случае мы накладываем сетку на конфигурационное пространство, тем самым получаем дискретную модель пространства. Таким образом, каждая клетка является либо свободной, либо занятой препятствием.

При этом мы должны выбрать определенное разрешение сетки и связность (например, 4 neighbours или 8 neighbours). Связность определяет «гладкость» движения робота.

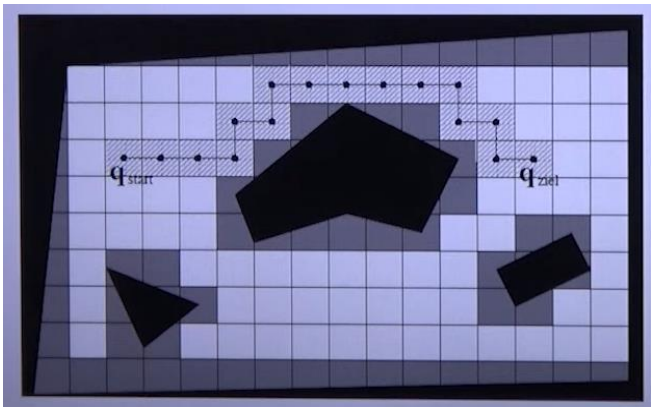

Рисунок 9 - Пример решения с использования сетки

Sampling-based Methods

Методы выше требуют явного моделирования всего окружения и препятствий внутри него. Однако это может быть трудной задачей, если препятствия имеют сложную форму и, если конфигурационное пространство имеет большую размерность. Методы нижеописанного типа позволяют избежать данную проблему.

Идея: случайным образом генерируем точки в конфигурационном пространстве и проверяем полученные ребра на возможность робота двигаться по этому отрезку (нет препятствий). В итоге получим граф, по которому далее находим путь. Другими словами, мы последовательно «изучаем» окружение, за счет случайного семплирования.

Probabilistic Roadmap

Данный алгоритм состоит из двух стадий:

1. Стадия обучения – сэмплирование точек в конфигурационном пространстве и проверяем на коллизии отрезков
2. Стадия запроса (Query phase) – генерация пути, используя search-based алгоритмы

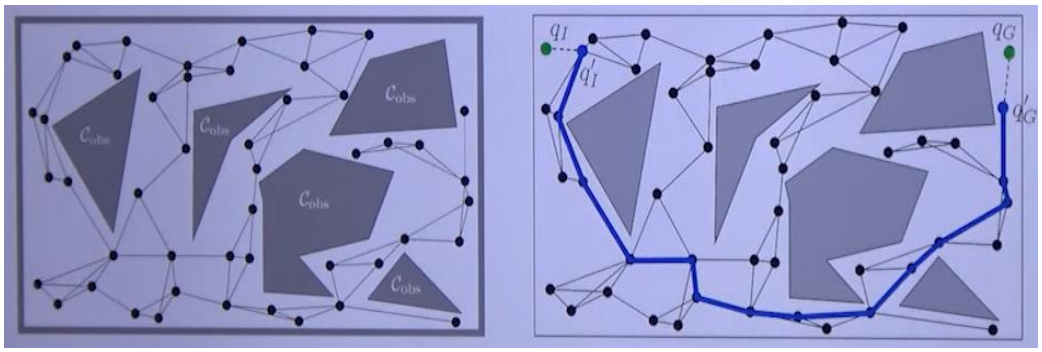

Рисунок 10 – Пример вероятностной дорожной карты

Rapidly-exploring Random Trees (RRT)

Идея алгоритма: строим дерево (граф без циклов), с parent-вершиной в начале пути, с помощью сэмплирования точек около листьев дерева и проверки на коллизии отрезков.

RRT не является оптимальный решением поиска пути, в то время как алгоритм RRT* является (там сэмплирование проводиться другим способом и дерево может перестраивать свою структуру).

Эвристические методы планирования пути

Большинство классических методов планирования пути не могут успешно работать в условиях динамического окружения. Исключением является только Artificial Potential Field.

Поэтому рассмотрим другую группу алгоритмов планирования пути, более подходящих в динамическом окружении.

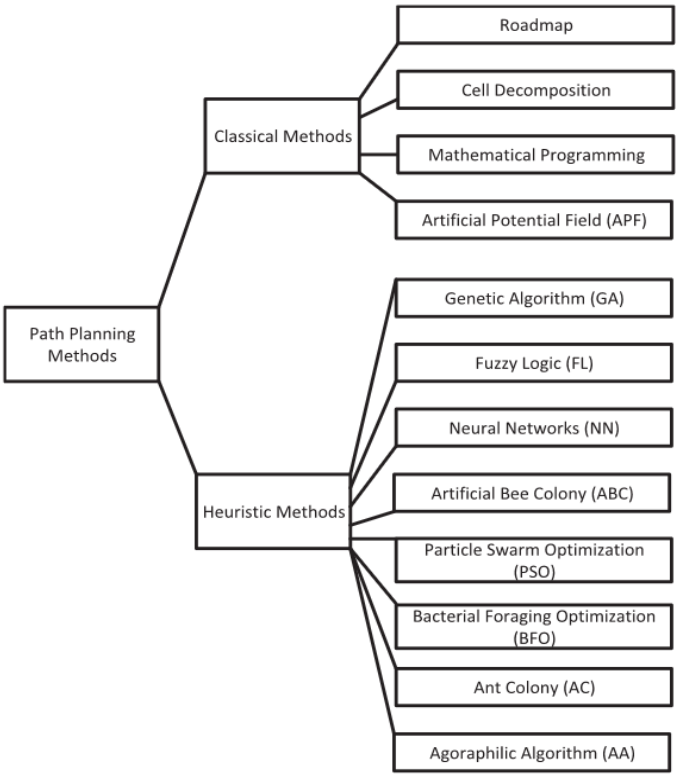

Рисунок 11 - Методы планирования пути

Также следует отметить, что методы группы heuristic methods можно использовать, если уже проведена локализация и построение карты местности, в ходе которых были определены динамические объекты.

Сравнительная таблица методов планирования пути

Method	Complete	Optimal	Scalability to higher DoFs	Comments
Visibility	Yes	Yes	No	+ Explicit free space - Poor scalability - Robot might travel close to obstacles
Voronoi	Yes	No	No	+ Explicit free space + Maximum clearance - Paths may be too conservative - Poor scalability
Potential field	Yes	No	Environment -dependent	+ Easy to implement + Can account for uncertainty - Susceptible to local minima
Dijkstra/A*	Yes	Grid	No	+ Faster than uninformed search + A* uses a heuristic function to drive the search more efficiently - Poor scalability
PRM	Yes	Graph	Yes	+ Efficient for multi-query problems + Probabilistic completeness - Jagged path
RRT	Yes	No	Yes	+ Efficient for single-query problems + Probabilistic completeness - Jagged path

Рисунок 12 - Сравнительная таблица методов планирования пути 1

Base-Algorithm	Advantages	Disadvantages
Artificial Potential Field	1. Less computational cost 2. Simple to implement 3. High adaptability	1. Tedious parameter tuning 2. Local minima problem 3. Oscillations 4. Inability to reach a goal when a large goal is nearby
Genetic Algorithm	1. Generate high quality accurate solutions	1. Can develop oscillations 2. Hard to implement in dynamic environments
Fuzzy Logic	1. Capable of representing the human thinking. 2. Can mimic the actions of a manual robot operator. 3. Fuzzy rules are transparent and clear. (As using the linguistic variables) 4. Can handle high uncertainties	1. Selecting right rules and membership functions are critical
Artificial Neural Networks	1. Nonlinear mapping capability 2. Capable of parallel processing 3. Learning ability 4. System can be retrained when the conditions are changed	1. Needs to go through a huge learning process 2. Computational cost is high 3. Impossible to come up with a transfer function
Artificial Bee Colony	1. Simplicity, flexibility and robustness 2. Ability to explore local solutions	1. Slow when in sequential processing 2. Higher number of objective function evaluation
Particle Swarm Optimization	1. Less computational complexity 2. PSO's high efficiency in terms of speed and memory requirements	1. Possibilities of getting tarped in local minimums in complex environments 2. Less accurate and practical than GA
Bacterial Foraging Optimization	1. Ability to solve the multi difficulty scheduling problem effectively	1. High computational cost minimums in complex environments
Ant Colony	1. Require less control parameters	1. Slow convergence
Agoraphilic	1. Take the optimistic approach of obstacle avoidance 2. No local minima problem 3. Less computational cost	1. Needs parameter tunning for tracking

Рисунок 13 - Сравнительная таблица алгоритмов планирования пути 2

Навигация в динамическом окружении

Описание задачи

Проблемы при навигации в динамическом окружении:

1. Motion prediction - прогнозирование поведения динамических объектов
2. Planning - планирование пути (траектории) с учетом наличия динамических объектов
3. Неопределенность в поведении динамических объектов
4. Real-time execution

Задача автономной навигации может быть разделена на 4 подзадачи:

1. Восприятие - система сенсоров робота должна получать информацию об окружении
2. Локализация и построение карты местности
3. Планирование пути - построение пути для достижения цели, избегая препятствий
4. Motion control - непосредственное управление роботом для слежения за спланированной траекторией

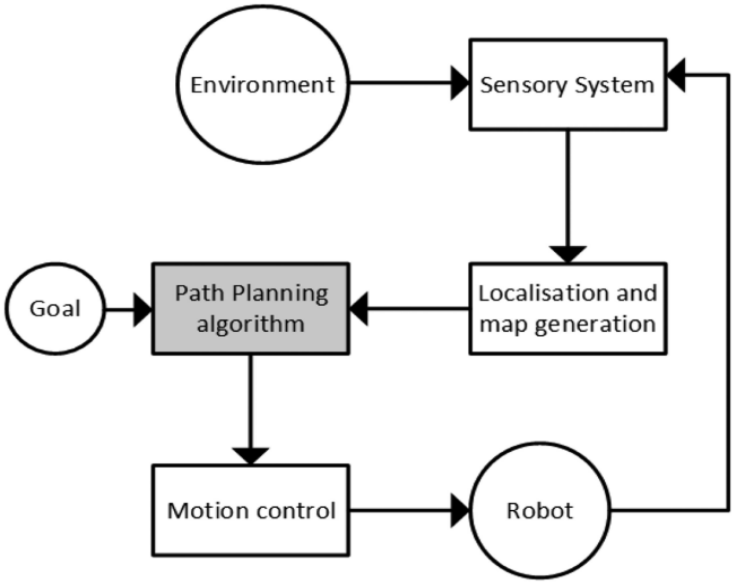

Рисунок 14 - Диаграмма задачи навигации

Создание карт динамического окружения с помощью мобильных роботов

Самый простой способ работы с динамическими объектами - использование карт “занятости” (occupancy grid algorithm). В данном алгоритме, область точно считается свободной после того, как робот наблюдал эту область свободной в течение достаточного времени (то же самое и с занятыми областями).

Другой метод - детектирование определенных особенностей, которые присущи только динамическим объектам и далее слежения за ними. Однако для реализации этого способа необходимо заранее знать особенности.

Также существует алгоритм Expectation-Maximization (EM). Его идея следующая: на этапе ожидания мы вычисляем вероятностную оценку того, какие измерения могут соответствовать статичным объектам. На этапе максимизации мы используем эти оценки для определения положения робота и карты. Этот процесс повторяется до тех пор, пока дальнейшее улучшение не будет достигнуто.

Суть этих вышеописанных методов в детектировании каким-то образом динамических объектов и фильтрации карты местности. Минус данного подхода в том, что в результате мы получаем карту местности только с изображенными на ней статическими объектами, а информация о динамические объектах фильтруется и не используется.

Для обхода данной проблемы существуют подходы позволяющие моделировать динамические аспекты окружения (low-dynamic or quasi-static states). Они основываются на том факте, что многие динамические объекты появляются при наблюдении только в конечном числе конфигураций (например, открытая или закрытая дверь).

Конечный автомат

В данном разделе дадим определение конечному автомату и исследуем его в случае применения в качестве принятия решений.

Автомат конечных состояний хорошо подходит для описания последовательного поведения управляющей программы. Используя конечный автомат можно разделить задачу управления на отдельные блоки, тем самым получив наглядное представление того, в каких случаях система будет попадать в некоторое состояние.

Такая структуризация упрощает дальнейшую отладку (например, если робот застрял на определенной точке, легко определить, какой переход не работает или какое условие отсутствует).

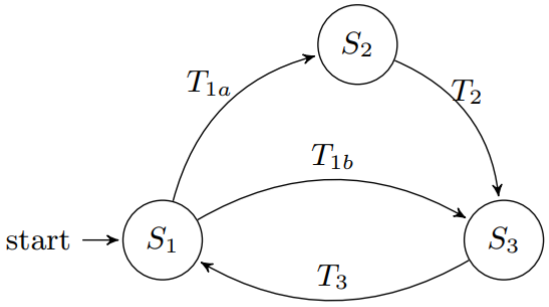

Рисунок 15 - Графическое представление конечного автомата

На Рисунке 15 представлено графическое описание конечного автомата, состоящее из дискретных состояний, представленных узлами графа и переходов, представленных ребрами графа. Когда переход срабатывает, конечный автомат изменяет свое фактическое состояние и совершает переход в другое состояние в соответствии с функцией перехода.

Формальное определение:

Конечный автомат представляется кортежем $A = (S, \Sigma, \delta, s_0, F)$:

- S — конечное непустое множество состояний

- Σ — входной алфавит (конечное непустое множество символов)

- $\delta: S \times \Sigma \to S$ — функция перехода

- s_0 — начальное состояние

- F — конечное (терминальное) состояние

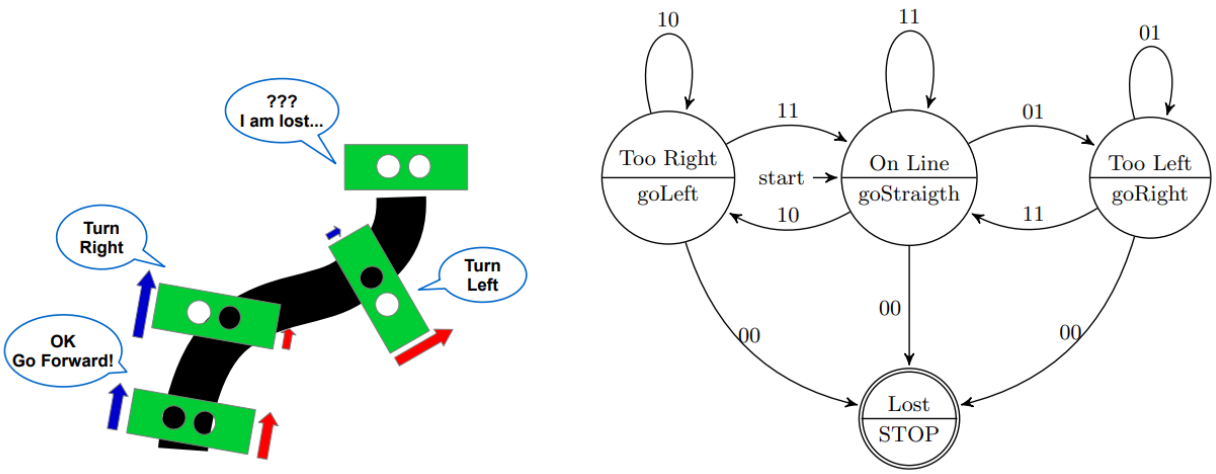

Рисунок 16 – Пример применения конечного автомата для принятия решения робота, который следует по линии

Итоги:

1. Есть набор состояний, в которых может находиться агент (робот)
2. В любой момент времени робот может находится только в одном из этих состояний
3. Выбор, продолжать ли поведение или перейти к чему-то другому, оценивается полностью внутри поведения, в котором робот находится в данный момент, т.е. нет никакой иерархии
4. Узлы тесно связаны со всеми остальными узлами, т. е. изменение одного сценария напрямую повлияет на то, будет ли он по-прежнему доступен для вызова из любого другого сценария поведения, или это может означать, что придется обновить информацию во всех других сценариях, чтобы не получить странное поведение
5. Итак, основные недостатки конечных автоматов: отсутствие модульности (степень, в которой компоненты системы могут быть разделены на составные части и повторно объединены) и недостаточная реактивность (способность быстро и эффективно реагировать на изменения)

Поведенческие деревья (behavior trees)

Поведенческое дерево – структура, используемая для переключения задач.

При использовании конечного автомата мы обычно имеем граф состояний с множественными переходами между состояниями, это может быть проблемой, так как в таком случае сложно определить какое действие будет выполняться дальше (из-за наличия большого количества циклов в графе).

Такая проблема отсутствует в структуре поведенческих деревьев, так как каждое состояние (или действие), которое представлено листом дерева имеет единственного «родителя» и возвращает ему сигнал статуса («успех/неудача/выполнение»). Заранее задается частота опроса, и на каждом «тике» частоты мы заново проходимся по дереву и узнаем состояние.

Структура

Поведенческое дерево — это дерево с направленными корнями, в котором внутренние узлы называются узлами потока управления, а узлы листьев - узлами выполнения.

Дерево начинает свое выполнение с корневого узла, который генерирует сигналы, разрешающие выполнение узла (Tick), с заданной частотой, которые посылаются его дочерним узлам. Узел выполняется тогда и только тогда, когда он получает Tick. Дочерний узел немедленно возвращает родительскому узлу сообщение Running, если его выполнение продолжается, Success, если он достиг своей цели, или Failure в противном случае.

Node type	Symbol	Succeeds	Fails	Running
Fallback	?	If one child succeeds	If all children fail	If one child returns Running
Sequence	→	If all children succeed	If one child fails	If one child returns Running
Parallel	⇉	If $\geq M$ children succeed	If $> N - M$ children fail	else
Action	text	Upon completion	If impossible to complete	During completion
Condition	text	If true	If false	Never
Decorator	◊	Custom	Custom	Custom

Рисунок 17 - Типы узлов в поведенческом дереве

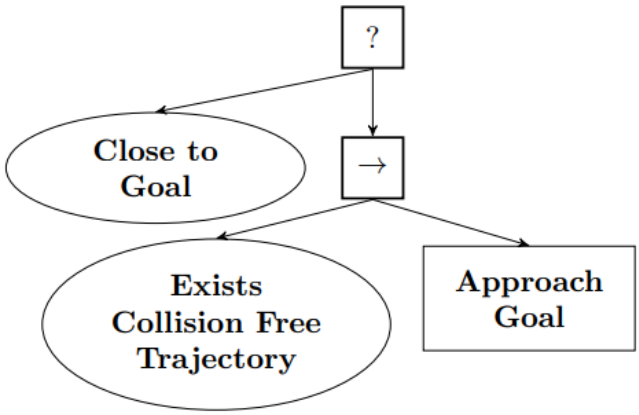

Рисунок 18 - Пример использования BT в задаче достижения роботом цели

Принципы проектирования

Явные условия

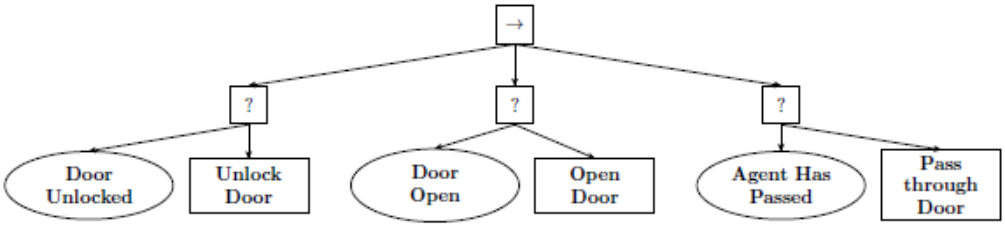

Рисунок 19 - Поведенческое дерево с применением явных условий

Добавление условий в паре с действиями производится с помощью Fallback node (?).

Инверсия действий

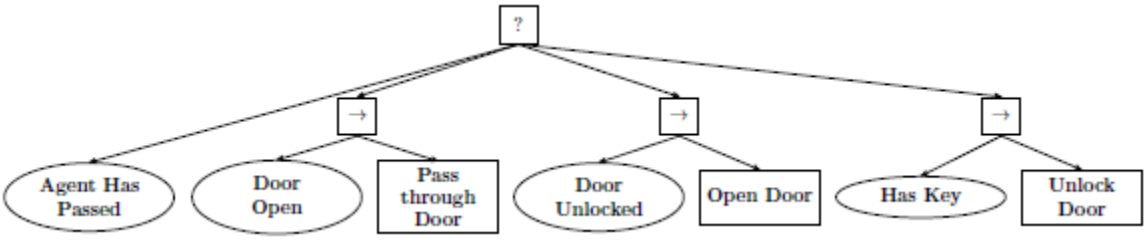

Рисунок 20 - Поведенческое дерево с применением инверсии действий

Построение неявной последовательности с помощью узла Fallback, для смены порядка действий.

Использование структуры решающих деревьев

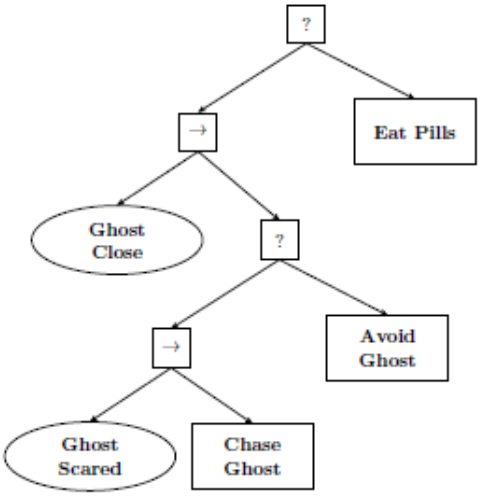

Рисунок 21 – Поведенческое дерево, имеющее структуру решающего дерева

Так как поведенческие деревья обобщают решающие деревья (decision trees), то можно использовать их структуру для обработки различных сценариев с условиями

Реализация безопасности через Sequences (→)

Для гарантирования запрета выполнения роботом определенных опасных действий (повреждения робота, наезд на людей и другие объекты) можно использовать структуру Sequence.

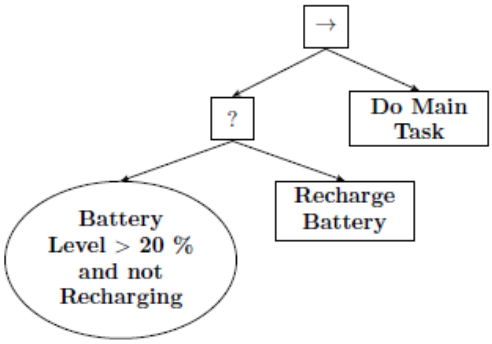

Рисунок 22 – Поведенческое дерево с гарантией безопасности

Использование Backchaining

С помощью поведенческого дерева можно реализовать структуру для агента, который будет выполнять действия в определенном порядке для достижения поставленной цели.

Для этого нужно:

1. Представить цель в виде условия

2. Синтезировать множество связанных (precondition == postcondition) поведенческих деревьев, имеющих структуру PPA (precondition, postcondition, action), реализованных для достижения цели

3. Имея такой набор, работая в обратном направлении от цели (backchaining), заменяем precondtion на PPA, имеющие соответствующее postcondition

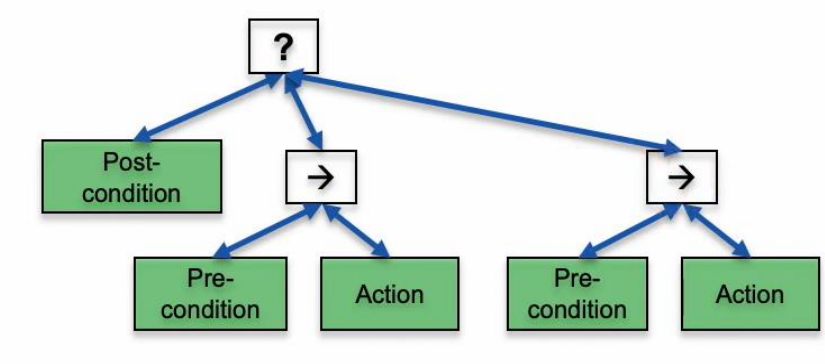

Рисунок 23 - Обобщенный формат PPA (postcondition, precondition, action)

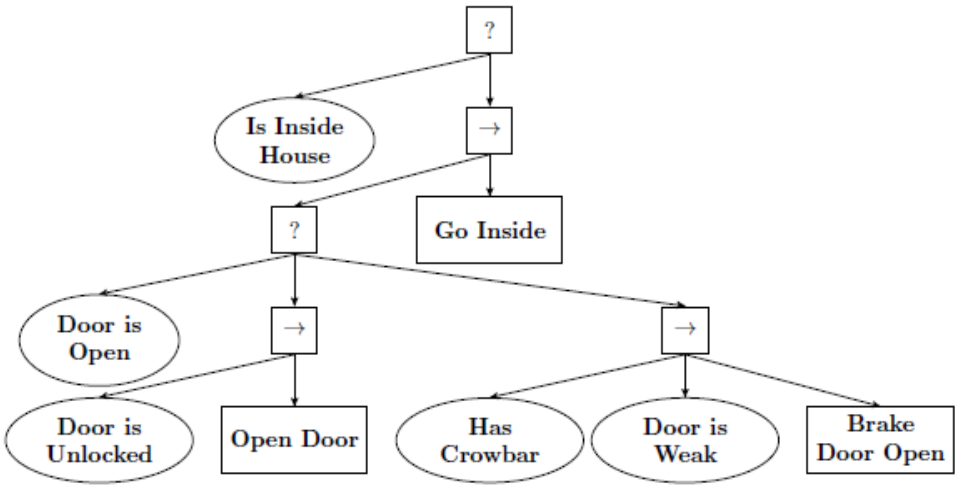

Рисунок 24 - Полученное поведенческое дерево (использовался алгоритм выше)

4 критерия для гарантирования достижения целей при использовании Backchaining

1. Действия должны достигать post-conditions за конечное время
2. Действия не должны нарушать «ключевые» цели, достигнутые в прошлом. Другими словами, действия должны достигать свои post-conditions, без нарушения важных для нас достигнутых целей (у действий будут существовать активные условия ограничений (Active Constraint Conditions) – список условий, которые нельзя нарушать ни в каком случае). Эти ограничения можно представлять в виде препятствий в конфигурационном пространстве, или в скоростном пространстве и т.д.

   Аналитически, найти такие условия можно следующим образом:

    a. Проверить все sequence узлы между действием и корнем дерева
    b. Рассматривать только потомков «слева»
3. Действия не должны нарушать условия, необходимые для выполнения будущих целей. Как найти все такие условия? Построить and-or-tree на основе исходного BT (замена sequence на and, fallback на or, действия на true (или убрать их если у них есть pre-conditions)). На основе данного дерева мы можем найти все условия, необходимые для выполнения всех поставленных целей (and-or-tree возвращает true).
   Таким образом, and-or-tree показывает какие возмущения могут быть успешно преодолены (цели будут достигнуты не смотря на них). Если возвращает True, то можем достигнуть, если False, то нет.
4. Действия не должны вызывать поддеревья (листы), которые уже вернули Failure в прошлом. Избежать вызов таких поддеревьев можно добавлением дополнительных pre-conditions к поддереву, содержащих и проверяющих данные, которые мы уже имеем. Также нужно учитывать, что pre-condition соответствующего действия могут быть нарушены в момент достижения post-condition.

Выбор подходящей детализации модулей поведенческого дерева

Так как поведенческое дерево является модульной структурой, то выбор детализации очень важен. Основной вопрос состоит в выборе того, что представлять в виде листового узла (отдельное действие или условие), а что - в виде поддерева. Можно рассмотреть два следующих случая:

1. Имеет смысл кодировать поведение в одном листе, если потенциальные части поведения всегда используются и выполняются именно в этой комбинации.

2. Имеет смысл кодировать поведение в виде поддерева, разбивая его на условия, действия и узлы управления, когда эти части, вероятно, будут использоваться в других комбинациях в других частях поведенческого дерева

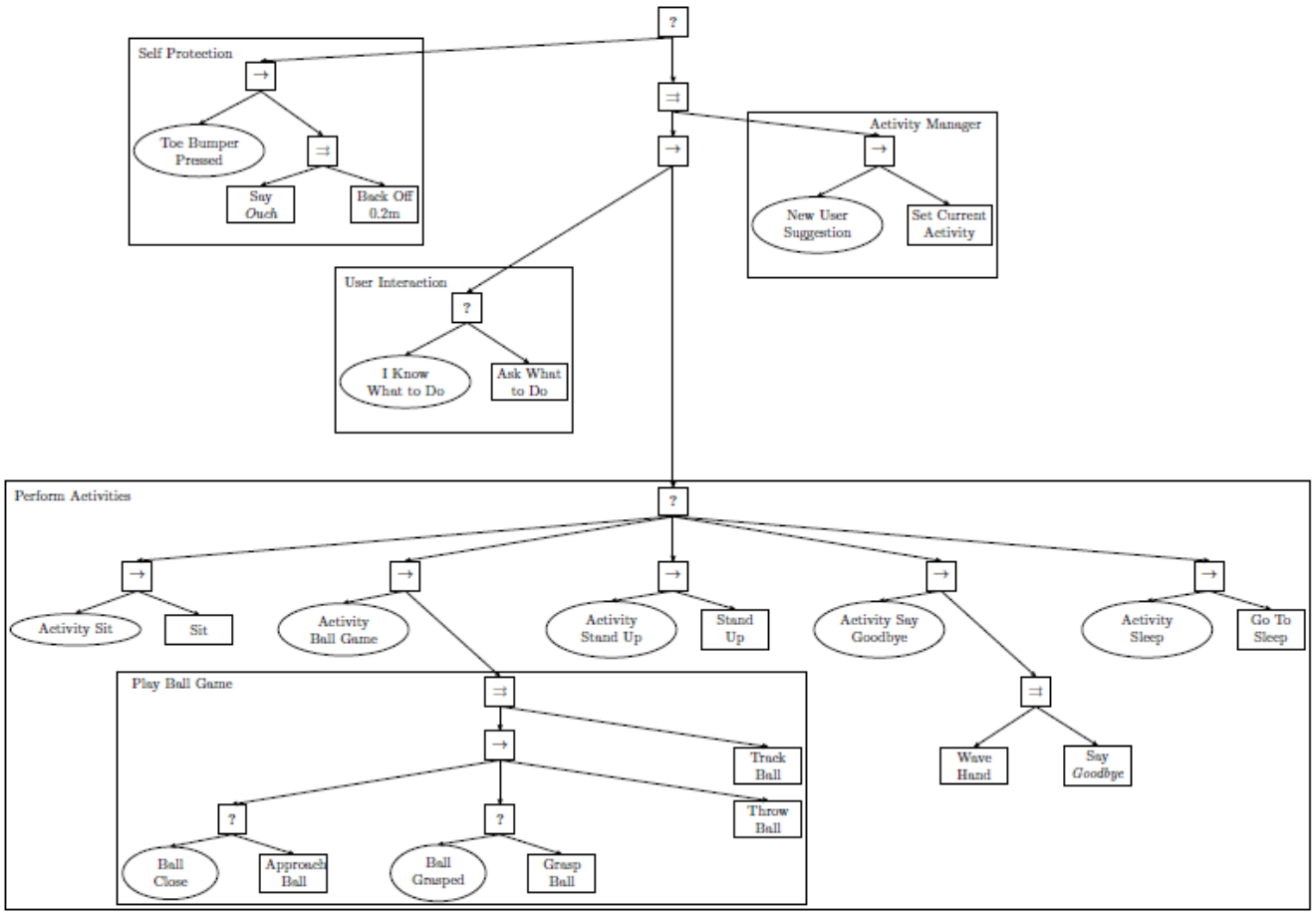

Рисунок 25 – Поведенческое дерево с определенной детализацией (произведено выделение в поддеревья отдельных модулей)

Преимущества и недостатки

Алгоритмы принятия решений можно реализовывать, используя различные архитектуры (Конечный автомат, Иерархические конечный автомат, Дерево принятия решений). Однако поведенческие деревья обобщают все эти архитектуры и имеют над ними преимущества.

Преимущества поведенческих деревьев:

1. Модульность (modularity) – малое число связей между компонентами. Это дает возможность создавать компоненты независимо друг от друга и встраивать их в систему без проблем
2. Иерархическая структура – действия подразделяются на уровни
3. Переиспользование кода – построение больших структур, используя уже реализованные малые части
4. Реактивность (отзывчивость) – возможность быстро и эффективно реагировать на изменения в среде. Поведенческие деревья являются реактивными, поскольку непрерывная генерация тиков и обход их дерева приводят к выполнению замкнутого цикла. Действия выполняются и отменяются в соответствии с обходом тиков, который

зависит от состояния возврата узлов листа. Листовые узлы тесно связаны с окружением (например, узлы условий оценивают общие свойства системы, а узлы действий возвращают статус Failure/Success, если действие не удалось/прошло успешно).

5. Наглядное графическое представление системы

Недостатки:

1. Необходимость реализовывать параллелизм - чтобы гарантировать полную функциональность, генерация и обход тиков должны выполняться параллельно с выполнением действий
2. Проверка большого количества условий при каждом новом тике

Принятие решений с помощью поведенческих деревьев

Выше мы уже рассмотрели то, каким образом организовывать структуру поведенческого дерева, однако не затрагивали именно выбор высокоуровневых целей для агента.

Можем выбирать цели, аналогично человеку (пирамида Маслоу):

1. Безопасность – избегание столкновений и нанесения ущерба объектам (в том числе самому себе)

2. Базовые нужды – например, поддержание уровня заряда для функционирования

3. Текущие задачи – выполнение поставленных задач (может потребоваться ранжирование задач по приоритету)

4. Помощь в командных задачах – помощь другим агентам для выполнения командной задачи

5. Самореализация – например, исследование поведения и среды с помощью алгоритма обучения с подкреплением

Control barrier functions

Control barrier functions (CBF) используются для проверки и обеспечения свойств безопасности управляемых объектов.

Использование CBF вместе с BT является взаимодополняющим, так как

1. BT работают таким образом, что совершают жесткий, мгновенный переход от одного состояния (от одной цели управления к другой, в таком случае могут появляться проблемы, если мы не учитываем другие цели) в другое (вначале 1, затем 2). Тогда как, CBF выполняет «мягкое», плавное слияние целей (выполняется 1, в то время как следующее принимается во внимание и обрабатывается). То есть добавляется некоторая динамика в переходных процессах
2. Сходимость BT (стремление состояния в область Success) требует не нарушение прошлых подзадач (состояние остается в некоторой окрестности, в которой прошлые подзадачи выполнены). CBF как раз таки следит за тем, чтобы это не нарушалось. Однако в случае множества решенных подзадач может быть такое, что нельзя не нарушить всех, тогда нам необходимо будет задать правильный приоритет между ними

То есть использование данной комбинации даст нам заранее определенные нами гарантии безопасности. Что очень важно для нашей задачи, так как мы работаем в динамической среде.

Итак, пусть система описывается

$$\dot{x} = f(x,u)$$

The key idea behind CBFs (26, 27) is to specify a barrier function $h : X \to \mathbb{R}$ such that the so-called *safe set* $\mathscr{C}$ is characterized by: $\mathscr{C} = \{x \in X : h(x) \geq 0\}$. Given the continuous system dynamics $\dot{x} = f_C(x,u)$, if we choose controls $u \in U_{inv}$ where

$$U_{inv} = \left\{u \in U : \frac{dh}{dx} f_C(x,u) \geq -\alpha(h(x))\right\},$$

and $\alpha \in \mathscr{K}$ is a class K functions (26), we are guaranteed to stay in the safe set $x \in \mathscr{C}$.

Выше, функция $\alpha: R_+ \to R_+, \alpha(0) = 0$ и α строго монотонно возрастает.

Мы называем множество U_{inv} инвариантным, в смысле того, что любая траектория, начинающаяся внутри инвариантного множества, никогда не достигнет дополнения этого множества (которое является «небезопасным» множеством).

Возможные требования: избегание движения в «небезопасную» область, избегание ситуации полной разрядки аккумуляторов, достижение цели за конечное время, оставаться в зоне коммуникации с оператором.

Математическое описание обработки множества условий (компромисс):

Assumption 2. *Each condition $C_i : X \to \{0,1\}$ can be formulated in terms of a CBF h_i, see Equation (35), as follows*

$$C_i = \{x \in X : h_i(x) \geq 0\}$$

Definition 2: Let

Preserves one C_i

$$K_i = \{u \in U : \frac{dh_i}{dx} f(x,u) \geq -\alpha(h_i(x))\}$$

Preserves all C_i (can be empty)

$$\bar{K}_j = \bigcap_{i=1}^{j} K_i \quad (4)$$

Preserves many C_i (never empty)

$$\hat{K}_k = \{\bar{K}_j : j \leq k, \bar{K}_j \neq \emptyset \wedge (j = k \vee \bar{K}_{j+1} = \emptyset)\} \quad (5)$$

$$u_i = \operatorname{argmin}_u ||u - k(x)||^2 \quad \text{s.t. } u \in \hat{K}_{i-1}$$

$$\implies \hat{K}_k \subset K_1$$

Итак, мы всегда гарантируем выполнение первого по приоритету условия.

$k(x)$ – какое-то управление, удовлетворяющее C_i, к которому будет стремится результирующее управление (минимизация нормы), пока не встретится какое-либо ограничение, которое необходимо обойти.

Синтезированное таким образом управления является «локальным» (получится неоптимальная траектория). Лучше всего использовать алгоритм планирования пути (например RRT), который будет также удовлетворять условиям CBF (например, нанести «опасные» множества на карту местности и считать их препятствиями). Однако в таком случае при случае пустого «безопасного» множества или динамического окружения придется перепланировать путь.

Использование генетических алгоритмов (genetic algorithms)

Описание алгоритма

Способность агента учиться на собственном опыте может быть реализована путем подражания естественной эволюции.

Генетический алгоритм — это алгоритм оптимизации, который черпает вдохновение в биологической эволюции, где набор агентов с индивидуальными поведениями эволюционирует до тех пор, пока один из них не решит заданную оптимизационную задачу достаточно хорошо.

Каждый определенный набор агентов называется поколением. На каждой итерации из предыдущего поколения создается новое. Сначала создается набор особей путем применения операций скрещивания и мутации к предыдущему поколению. Затем подмножество особей выбирается путем отбора для следующего поколения на основе заданной функции сравнения (отбора).

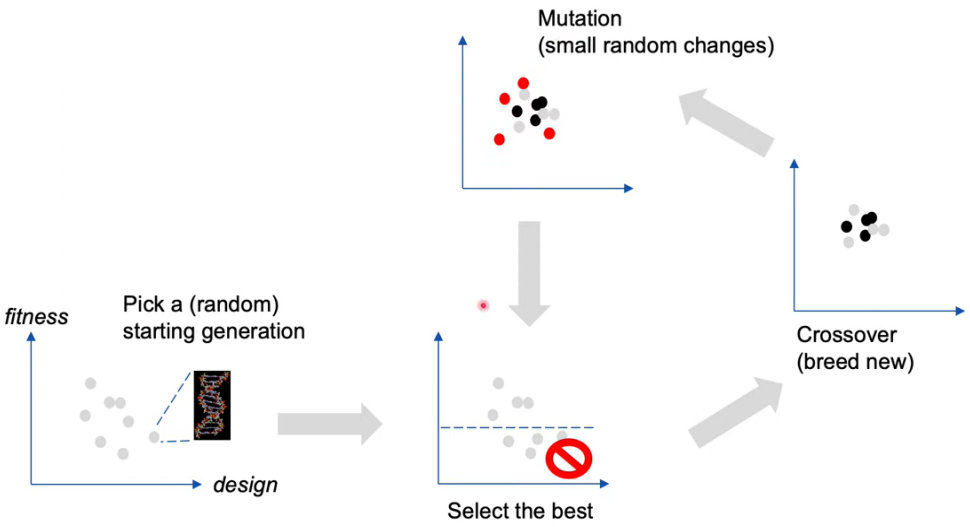

Рисунок 26 - Диаграмма генетического алгоритма

Данные операции: скрещивание (cross-over - случайная замена поддерева одного дерева на поддерево другого дерева на любом уровне), мутация (mutation (случайное «малое»

изменение) - заменяет узел в дереве другим узлом того же типа) и отбор (selection – с некоторой вероятностью отбор лучших особей, оценивая их по некоторому функционалу), достаточно просто реализовать при работы с деревьями.

Проблема при применении генетического алгоритма – чрезмерное увеличение размера дерева, без улучшения его функционирования на функции сравнения. Поэтому требуется удаление «ненужных» поддеревьев (это можно делать и в процессе поиска для ограничения размера дерева). Это можно сделать следующими способами:

1. Случайное обрезание деревьев (random pruning) используя функционал качества (функция сравнения)

2. Удаление пустых рабочих областей дерева

Использование GA с GT

Представляем дерево в виде сочетания дерева, гарантирующего безопасность, и дерева обучения, которое будет расширено в процессе обучения.

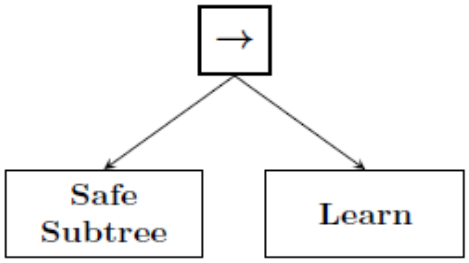

Рисунок 27 – Пример структуры поведенческого дерева с использованием генетического алгоритма

Использование обучения с подкреплением

Плюсы RL: около-оптимальное решение, «быстрое и простое» решение (не надо делать вручную, кроме гиперпараметров и функции вознаграждения).

Плюсы BT: модульность, иерархичность, гаранты безопасности и сходимости.

Если совместить два эти подхода определенным методом, то можно получить плюсы от обоих.

Как мы можем «изучить» структуру дерева:

1. Генетические алгоритмы

2. Замена части дерева (лист, поддерево, всё дерево (в этом случае будет end2end RL, BT не будет)) с помощью RL (используем модульность BT)

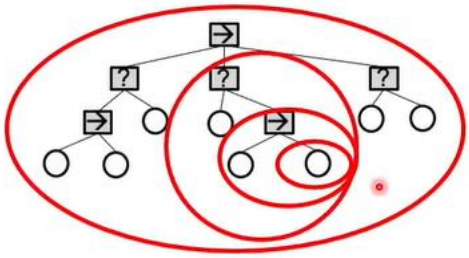

Рисунок 28 – Различные части (выделены красным), которые могут быть заменены алгоритмом обучения с подкреплением

3. Замена внутренних узлов (fallback, sequence)

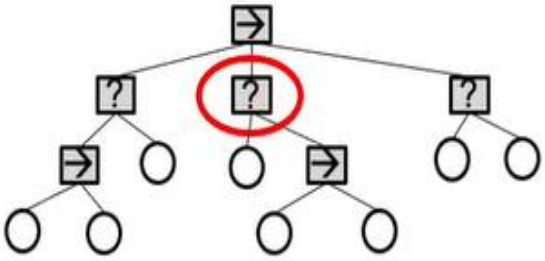

Рисунок 29 – Пример замены внутренних узлов для реализации другого типа поведения

Изучение приоритета в Fallback

С помощью RL можно изучить приоритет при выполнении Fallback узла: самым классическим способом будет применение алгоритма Q-learning, с помощью которого мы находим функцию Q(s,a) для каждого поддерева и затем сортируем их по value.

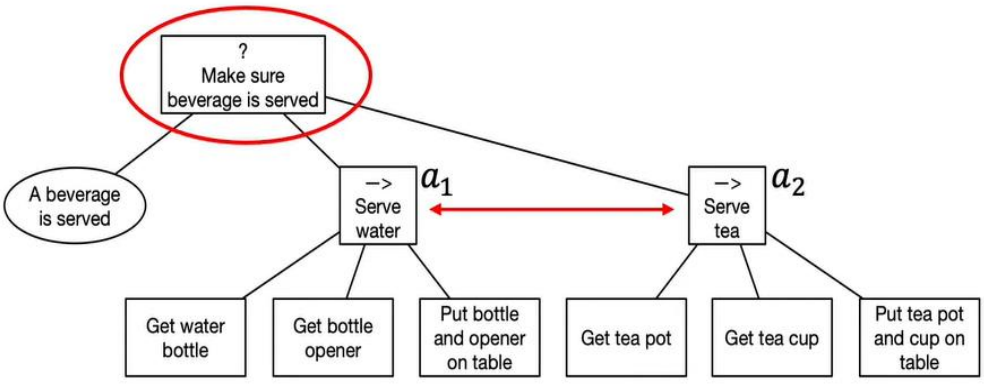

Рисунок 30 – Пример изучения приоритета Fallback с помощью RL

Изучение выполнения действия

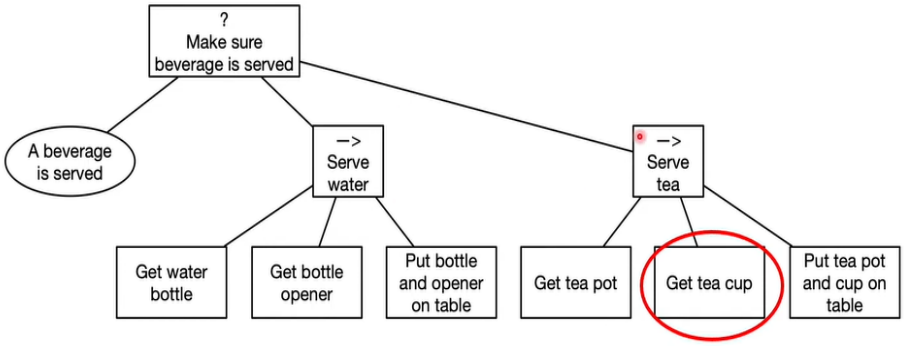

Рисунок 31 – Пример добавления отдельного листа, поведение которого изучается с помощью RL

В данном случае также можно применить алгоритм Q-learning с состояниями s, действиями A и вознаграждением r, заданными вручную.

При этом вознаграждения за действия можно выбирать на основе структуры уже существующего дерева, чтобы все переходы удовлетворяли условиям:

1. Действие достигает postcondition за конечное время
2. Действие не нарушает «ключевой» достигнутой цели в прошлом
3. Действие не нарушает условий необходимых для выполнения цели в будущем
4. Действие не вызывает поддерево, которое вернуло Failure в прошлом

Гаранты безопасности в случае применения RL можно также достичь, используя CBF. Это можно сделать, если ограничить множество действий A до множества $\hat{K}$, которое определялось как

$$K_i = \{u \in U : \frac{dh_i}{dx} f(x,u) \geq -\alpha(h_i(x))\}$$

$$\bar{K}_j = \bigcap_{i=1}^{j} K_i$$

$$\hat{K}_k = \{\bar{K}_j : j \leq k, \bar{K}_j \neq \emptyset \wedge (j = k \vee \bar{K}_{j+1} = \emptyset)\}$$

Гаранты сходимости в случае применения RL можно достичь, используя следующие условия:

1. Избегание неверных переходов (которые не приближают агента к цели)
2. Выполнение переходов за конечное время

Принятие решений (Decision-making) с использованием RL

Задача принятия решений

В отличии от задачи планирования (в которой решается проблема перемещения робота из А в В), задания принятия решений состоит в нахождении корректной конечной точки (конфигурации) В для выполнения определенной задачи. То есть в задаче планирования мы отвечаем на вопрос: как добраться из точки А в точку Б, а в задаче принятия решения: куда двигаться роботу?

Основные аспекты алгоритмов принятия решений:

1. Разнообразие целевых функций (reward functions) – так как подходы алгоритмов принятия решений можно применять к многим различных задачам, то для каждой задачи мы подбираем определенную целевую функцию, которая будет отражать то, что мы хотим от робота
2. Неопределенность (actuation and sensing) – при синтезе алгоритмов принятия решений можно использовать информацию о неопределенности в управлении или в измерениях для получения более эффективного решения. В рассматриваемых ниже методах используется только информация о неопределенности в управлении роботом при синтезе алгоритмов принятия решений.
3. Только «дискретные» проблемы – конечное число состояний и действий робота

Агент и среда (Agent and Environment)

Агент будет представлять нашего робота, которым мы можем управлять и который будет автономно изучать окружение и принимать решения, используя алгоритмы принятия решений, для выполнения поставленной задачи. У робота присутствуют двигатели, которыми можно выполнять действия в среде (влиять на нее), а также сенсоры, от которых робот получает информацию о среде.

Среда – всё остальное что окружает робота. Среда посылает роботу информацию (которую робот получает с помощью сенсоров), а также отклик за каждое действие, которое совершает робот.

$Action\ A$: действие, которое предпринимает робот на основе алгоритма принятия решения

$Observation\ O$: информация, которую агент получает от среды

$Reward\ R_t$: скалярный сигнал обратной связи, который характеризует каким-либо образом цель работы робота. Эта величина определяет, насколько «хорошо» робот справляется с поставленной задачей в определенный промежуток времени t

Цель: максимизация ожидаемой кумулятивной (сумма наград за последовательность выполненных действий) награды

Агент и среда взаимодействуют в каждый интервал времени t.

Агент в каждый момент времени t:

1. Выполняет действие A_t
2. Получает наблюдение O_t
3. Получает награду R_t

В то время как среда:

1. Получает действие от агента A_t
2. Отправляет наблюдение O_{t+1}
3. Отправляет награду R_{t+1}

Заметим, что мы будем считать каждый «тик» времени тогда, когда среду отправляет какие-то данные.

Пример: Одноэтапное принятие решения

Рассмотрим пример, в котором робот может выполнить только одно действие для максимизации ожидаемой награды

$$a^* = argE\ (A = a)$$

Однако в реальных задачах роботу необходимо совершить некоторую последовательность действий для решения поставленной перед ним задачи. При этом возникают следующие трудности:

1. Награда может быть отложенной
2. Последующие действия будут зависеть от текущих наблюдений

Переходные состояния (State transitions)

$State\ S$: вся информация о конкретном сценарии принятия решений (текущее положение, прошлые действия, история наград, история наблюдений, неопределенности в управлении и тд), которую агент должен использовать для выполнения нового действия.

Модель перехода $T(s_{t+1}, a_t, s'_t) = p(s_t, a_t)$: вероятность того, что конкретное действие a в состоянии робота s приведет в состояние s'

Свойства Маркова (Markov Property): вероятность модели перехода зависит только от текущего состояния и действия

$$p(a_0, a_1, \dots, a_t, s_0, s_1, \dots, s_t) = p(a_t, s_t)$$

Переходные состояния бывают двух типов:

1. Детерминированные:

$$p(s, a) = \delta_{s_{next}}(s') = \{1, \quad s' = s_{next}\ 0, \quad otherwise$$

2. Стохастические: мы не уверены, что переход из одного состояния в другое при определенном действии произойдет 100% (тут мы и используем неопределенность при управлении роботом). То есть у нас есть некоторое распределение вероятностей, которое

определяет какое состояние будет после выполнения действия (а так как мы тут рассматриваем случаи с конечным числом состояний и действий, то у нас будет pmf, которое мы можем представить в виде таблицы)

s_t'	$T(s, a, s')$
s_1	0.1
s_2	0.8
s_3	0.1

Марковские процессы принятия решений (Markov Decision Process)

Марковский процесс принятия решений (МПП) — это стохастический процесс управления с дискретным временем. Он обеспечивает математическую основу для моделирования принятия решений в ситуациях, когда результаты частично случайны, а частично находятся под контролем агента, принимающего решение.

Марковские процессы принятия решений являются продолжением идеи цепей Маркова; разница заключается в добавлении действий (позволяющих выбирать) и вознаграждений (дающих мотивацию). В простейшем случае, если для каждого состояния существует только одно действие (например, "ждать") и все вознаграждения одинаковы (например, "ноль"), марковский процесс принятия решений сводится к марковской цепи.

Допущения:

1. Полностью наблюдаемая среда: полностью «доверяем» нашим наблюдениям (в системе нет неопределенностей при работе сенсоров)
2. Markov dynamics: марковское допущение (история агента независима от его будущего)
3. Стохастические переходы: существует pmf будущих состояний агента
4. Стационарность: действия и принимаемые решения роботом не зависят от момента времени

Математическая формулировка MDP

Определяется кортежем: $\langle S, A, P, R \rangle$

- S: конечное множество состояний
- A: конечное множество действий
- P: вероятности переходов состояний (state transition probabilities) (это можно представлять как вероятностное отображение текущего состояния в следующее состояние)

$$T(s_{t+1}, a_t, s_t') = p(s_t, a_t)$$

- R: функция награды

$$E(R_{t+1} | S_t = s, A_t = a)$$

Рассмотрим два типа задач:

1. Эпизодическая задача (Episodic task) – конечная задача, в которой мы имеем конечную цель, она будет занимать T временных шагов (например, задача достижения роботом определенного местоположения). Тогда максимизируемая ожидаемая кумулятивная награда (возврат G_t):

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_{t+T}$$

2. Продолжительная задача (Continuing task) – никогда не заканчивается, тогда возврат будет вычисляться в виде суммы ряда:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$

Как видим, в таком случае мы будем получать бесконечные возвраты, поэтому вводится коэффициент дисконтирования γ. Данный гипер-параметр позволяет настраивать алгоритм на «жадность».

$\gamma \to 0$ «близорукий» («жадный») агент

$\gamma \to 1$ «дальнозоркий» агент

Стратегия (Policy)

Стратегия π определяет поведение агента. Это отображение состояний в действия (states to actions). Цель MDP найти подходящее отображение π, которое максимизирует возврат G_t.

Есть два типа стратегий:

1. Детерминированная $a = \pi(s)$. Эту функцию можно представить как LUT (таблица поиска) – нет никакой неопределенности.
2. Стохастическая $\pi(s) = p(S = a)$. Отображение можно представить в виде условной вероятности.

Value functions

1. State-value functions: ожидаемый возврат, если робот начинает в состоянии s и следует стратегии π:

$$v_\pi(s) = E_\pi[S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} | S_t = s]$$

2. Action-value functions: это почти то же самое, только теперь начиная в некотором состоянии s робот еще выполняет какое-то конкретное действие a и далее следует стратегии π:

$$q_\pi(s,a) = E_\pi[S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} | S_t = s, A_t = a]$$

Уравнение Беллмана

Основывается на идее рекурсивной декомпозиции.

Итак, мы определили функцию стоимости (значимости) состояния: её можно разбить на 2 части: мгновенная награда + сниженная функция стоимости последующих состояний. Это и будет являться нашей рекурсивной декомпозицией.

Для наглядности, распишем это

$$\begin{aligned} v(s) &= E[S_t = s] \\ &= E[S_t = s] \\ &= E[S_t = s] \\ &= E[S_t = s] \\ &= E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s] \end{aligned}$$

Таким образом, можно рекурсивно проводить декомпозицию.

Аналогично можем сделать и с функцией стоимости действия:

$$q(s,a) = E[R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\ ]$$

Оптимальность

1. $v_*(s) = max_\pi v_\pi(s)$ – оптимальное значение для состояния
2. $q_*(s,a) = max_\pi q_\pi(s,a)$ – оптимальное значение для пары состояние-действие
3. π_* - оптимальная стратегия тогда и только тогда, когда:

$$\pi_*(a) > 0 \text{ только в случаях } q_*(s,a) = max_b q_*(s,b)\ \ \forall s \in S$$

Динамическое программирование

Динамическое программирование это способ решения последовательных задач. Основная идея заключается в разбиении сложной задачи на подзадачи, решение подзадач и сливание решений в единое.

Для применения данного метода необходимы следующие условия:

1. Выполнение принципа оптимальности: наша задача должна иметь оптимальную подструктуру (например, если мы имеем решение задачи планирования от промежуточной точки до финальной, то это решение будет являться под-решением задачи планирования пути от начальной точки до финальной через промежуточную точку)

2. Перекрывающиеся подзадачи: подзадачи повторяются много раз (например, если мы решили задачу планирования от промежуточной точки до финальной, то это решение можно использовать для любой задачи планирования пути от любой начальной точки до финальной через данную промежуточную)

MDP удовлетворяет этим условиям.

Поиск оптимальной стратегии

Наивный подход:

$$v_*(s) = max_\pi v_\pi(s)$$

$$v_*(s) = max_\pi [S_t = s]$$

$$v_*(s) = max_{a \in A} R_s^a + E_{p(s,a)} \gamma v_*(s')$$

Так как данное решение нелинейно, то будем использовать итеративные методы:

- Value iteration
- Policy iteration

Value iteration

В данном методе функция π не используется при (однако она вычисляется по найденной $v(s)$ если необходимо).

Данный метод проходит по всем возможным value для каждого состояния, до тех пор, пока последовательность не сойдется.

Иными словами, имеем

$$v_*(s) = max_{a \in A} R_s^a + E_{p(s,a)} \gamma v_*(s')$$

Начинаем с начального предположение функции v_0 при $k = 0$ и повторяем пока последовательность не сойдется:

$$v_{k+1} = max_{a \in A} R_s^a + E_{p(s,a)} \gamma v_k(s')$$

Policy iteration

Так как агенту необходимо найти только оптимальную стратегию (а не значения value function для каждого state), то логично будет итеративно искать policy

Тут мы проделываем аналогичные шаги:

1. Задаем начальное предположение π_0
    a. Оценка стратегии (policy evaluation):

$$v_k(s) = v_{\pi_k}, \ \ s \in S$$

можем это рассчитать, так как у нас уже есть стратегия и мы знаем какое действие необходимо произвести, а соответственно и сможем вычислить value function для текущего состояния и действия.

b. Улучшение стратегии (policy improvement):

$$\pi_{k+1} = argmax_{a \in A} q_{\pi}(s, a)$$

изменяем нашу стратегию, путем вычисления оптимальной action-value function

Повторяем до тех пор, пока стратегия не сойдется.

Заключение

Построение поведенческого дерева для решения задачи

Построим поведенческое дерево, используя изученную теорию

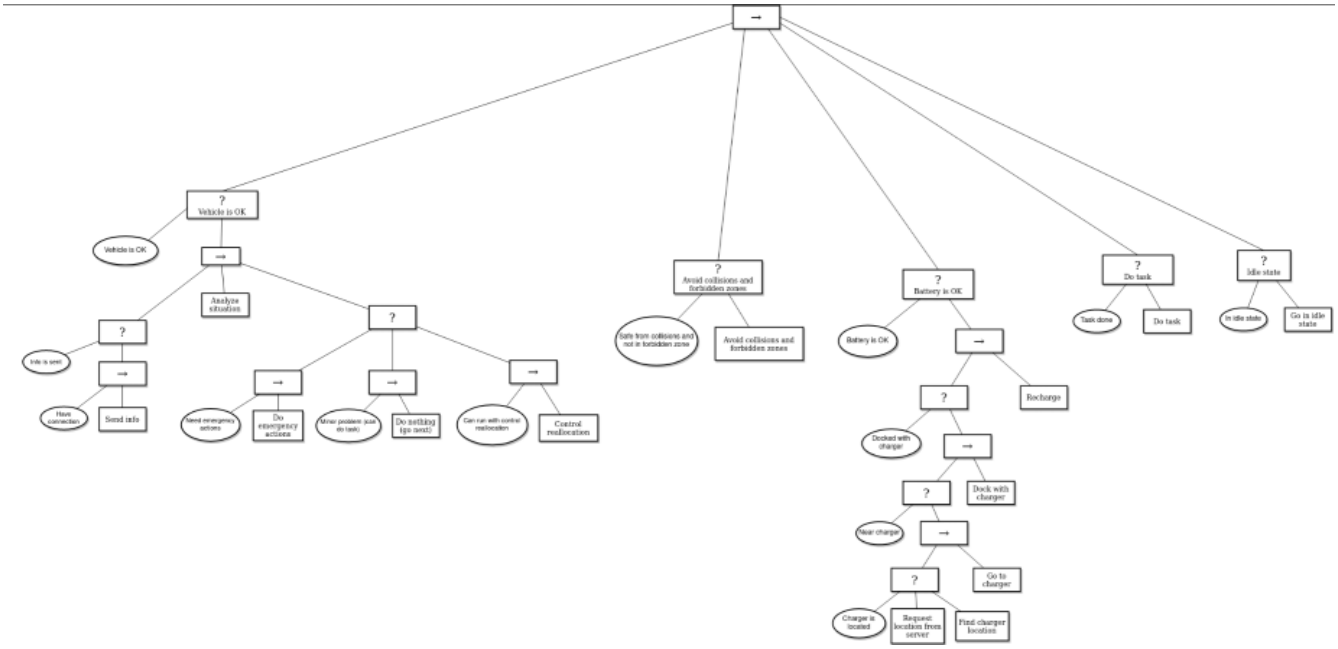

Рисунок 32 – Графическое представление поведенческого дерева для принятия решений в нашей задаче (построено с помощью draw.io)

Если захотим реализовать мультиагентную сеть, то можно добавить действие к корневому узлу (между Avoid collisions и Battery is OK): Revise Task Assignment, в котором будет обновлять текущее задание.

4 условия для сходимости дерева

1. Все действия должны выполнять соответствующие post-conditions за конечное время
2. Действия не нарушают "ключевые" цели, достигнутые в прошлом (смотреть таблицу ACC)

Таблица Active-constraint-conditions

Действия A_i	Post-conditions для A_i (цели)	Active constraint conditions (ACC)
Send info	Vehicle is OK	-
Control reallocation	Vehicle is OK	-
Emergency stop	Vehicle is OK	-
Avoid collisions and forbidden zones	Safe from collisions and not in forbidden zones	Vehicle is OK
Recharge	Battery is OK	Docked with charger AND Safe from collisions and not in forbidden zones AND Vehicle is OK
Dock with charger	Docked with charger	Near charger AND Safe from collisions and not in forbidden zones AND Vehicle is OK
Go to charger	Near charger	Charger is located AND Safe from collisions and not in forbidden zones AND Vehicle is OK
Request location from server	Charger is located	Safe from collisions and not in forbidden zones AND Vehicle is OK
Find charger location	Charger is located	Safe from collisions and not in forbidden zones AND Vehicle is OK
Do task	Task done	Battery is OK AND Safe from collisions and not in forbidden zones AND Vehicle is OK
Go in idle state	In idle state	Task is done AND Battery is OK AND Safe from collisions and not in forbidden zones AND Vehicle is OK

3. Действия не должны нарушать условия, необходимые для выполнения будущих целей. Синтезируем and-or-tree из нашего BT

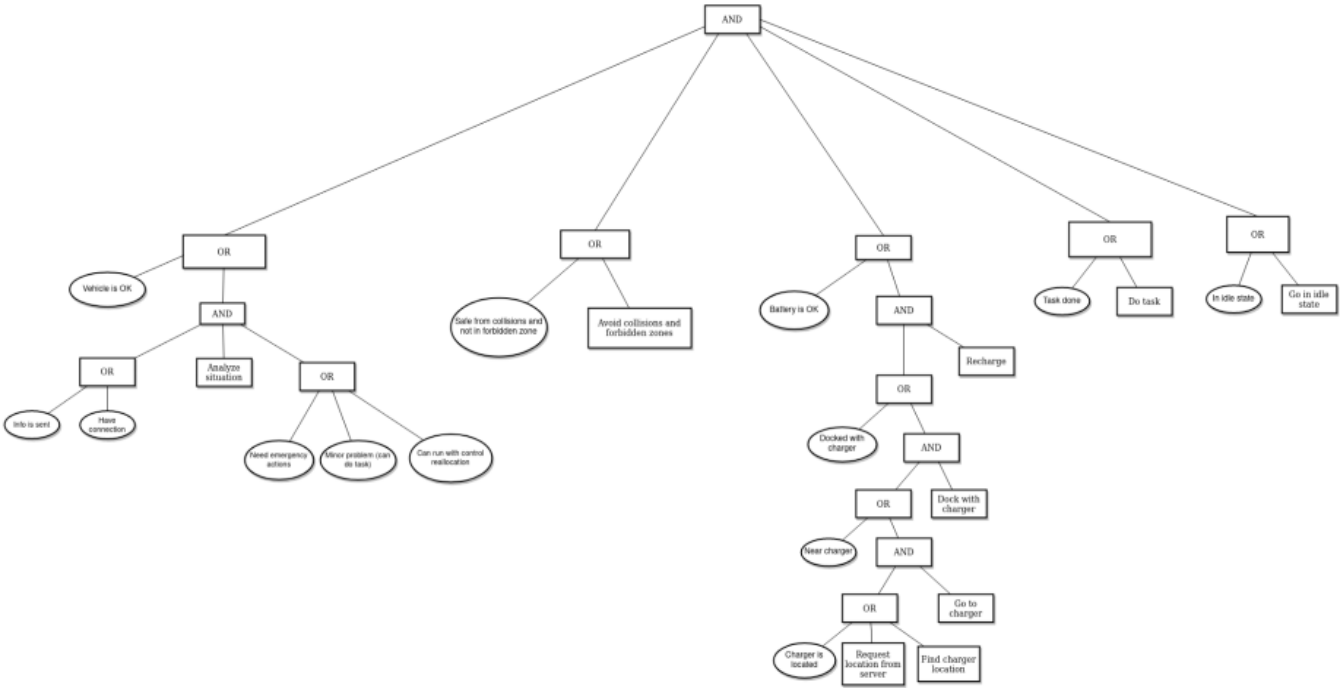

Рисунок 33 – And-or дерево

Если and-or-tree возвращает true, то мы можем достичь всех целей нашего робота. В данном случае всё достигается.

4. Действия не должны вызывать поддеревья (листы), которые уже вернули Failure в прошлом. Избежать вызов таких поддеревьев можно добавлением дополнительных pre-conditions к поддереву, содержащих и проверяющих данные, которые мы уже имеем. Также нужно учитывать, что pre-condition соответствующего действия могут быть нарушены в момент достижения post-condition. В конкретной данном дереве это соблюдается.

Выводы

По итогам проведенного обзора можно сделать следующие выводы:

1. Для реализации алгоритма принятия решения в нашей задаче будем использовать структуру поведенческого дерева, так как это позволит нам получить необходимые в поставленной задаче гарантии безопасности функционирования агента, а также гарантии сходимости (при учете правильного построения дерева (условия были представлены в соответствующем пункте))
2. Использование обучения с подкреплением (или генетических алгоритмов) возможно только в отдельных листах дерева (возможно изучения поведения, которое сложно запрограммировать вручную), так как при замене всего дерева (end2end RL) не будет гарантий безопасности и сходимости
3. Планировщик траектории должен быть разбит на две части: глобальный и локальный. Это позволит уменьшить вычислительные мощности, а также добавит модульность в

решение (глобальный планировщик ставит цели, в то время как локальный их решает с учетом локальной ситуации)

4. Так как роботу заранее не дана карта местности, необходимо использовать алгоритмы SLAM

Ресурсы

1. Using Finite State Automata in Robotics: Richard Balogh and David Obdrˇz´alek
2. Introduction to Autonomous Mobile Robots: Ronald C. Arkin
3. Behavior Trees in Robotics and AI: Michele Colledanchise and Petter Ogren
4. Safe and Efficient Navigation in Dynamic Environments Anirudh Vemula CMU-RI-TR-17-40 July 2017
5. Control Barrier Functions: Theory and Applications: Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada
6. Autonomous Navigation in Dynamic Environments: Springer, Christian Laugier and Raja Chatila
7. Planning and Decision Making for Aerial Robots: Yasmina Bestaoui Sebbane, Springer
8. Past, Present and Future of Path-Planning Algorithms for Mobile Robot Navigation in Dynamic Environments H. S. HEWAWASAM (Student Member, IEEE), M. YOUSEF IBRAHIM ,AND GAYAN KAHANDAWA APPUHAMILLAGE (Member, IEEE)
9. Курс "Машинное обучение с подкреплением" Центра когнитивного моделирования МФТИ http://rairi.ru/wiki/index.php/Машинное_обучение_с_подкреплением