Введение

актуальность темы

Планирование движения - фундаментальная проблема робототехники и искуственного интеллекта. В последние 20 лет в научном сообществе возрос интерес к ней, особенно в задачах связанных с мультиагентными системами.

цель работы

Цель данной работы - исследование одного из способов решения данной проблемы - программирование роботов, основанное на поведениях, или на реакциях на окружающую обстановку. Этот подход часто используется в тех ситуациях, где сложное поведение нужно получить при сравнительно несложной реализации отдельных едениц системы. Системы основанные на этом решении получаются надежными и легко модернизируемыми.

основные задачи

Исследовать реактивный подход к решению задачи планирования движения.

Спроектировать модель системы которая сможет решить конкретную задачу плагнирования движения основываясь на данном подходе.

Разработать компьютерную симуляцию подобной системы, используя JavaFX в качестве инструмента для визуализации.

ВКР структура работы

области знаний

Вычислительная геометрия.

Искуственный интеллект.

Мультиагентные системы.

оценка современного состояния

все очень хорошо

новизна

Наша имплементация будет отличаться от существующих аналогов и позволит с легкостью на ее основе построить реально функционирующую систему роботов. В частности она предлагает решения таких фундаментальных проблем робототехники как:

Сложность архитектуры. Обычно системы состоящие как из одного, так и из нескольких роботов делают по пути который называют “сверху вниз”. Это значит, что заранее разрабатываются четкие спецификации для того, как система должна себя вести, и потом выдвигаются требования для каждого из отдельных компонентов. Такой подход можно назвать весьма непредсказуемым и запутанным, тяжело заранее предсказать как будет идти взаимодействие множества абсолютно разных компонентов. Наша система будет разработана наоборот, с использованием подхода “снизу вверх”. Это значит что мы

смоделируем систему из простых, идентичных друг-другу роботов, а необходимое сложное поведение появится как побочный эффект.

Надежность. Большинство систем так или иначе связанных с робототехникой и искуственным интеллектом, опираются на каждую маленькую часть (камеру, сенсор, привод), и если хотя бы одна из них дает сбой, то обычно это приводит в нерабочее состояние всю систему целиком. Система которую предложим мы будет гарантированно работать, даже если некоторые ее части выйдут из строя.

Гибкость. Системы, построенные по пути “сверху вниз” тяжело адаптируются к изменяющейся окружающей среде, и их тяжело быстро улучшить.

практическая значимость

Практическая значимость модели, разработанной нами очевидна: с ее помощью можно построить систему, которая будет иметь возможность работать там, где человеку это не под силу, например:

- сбор токсичных отходов с места аварии
- исследование других планет или любого другого пространства, недоступного человеку.

Глава 1. Обзор существующих методов планирования движения, и средств моделирования мультиагентных систем

особенности проектируемого ПО в сравнении с аналогичными, как отечественными, так и зарубежными

Планирование движения - фундаментальная проблема робототехники. Она может быть описана как получение непрерывного движения агента из одной точки пространства в другую, с избежанием столкновений как со стачичными так и с движущимися препятствиями.

характеристики аналогов и на основе их анализа обоснование необходимости разработки собственного решения- JADE

Существует множество готовых решений для симуляции работы мультиагентных систем. Среди наиболее изаестных - MRDS (Microsoft Robotics Development Studio), Webots, Khepera Simulator и JADE. Два из этих решений являются коммерческими, и их нет в открытом доступе.

Одно из самых популярных средств моделирования мультиагентных систем на сегодняшний день это некоммерческий проект JADE написанный на языке программирования Java. Он основан на стандартах, разработанных FIPA (The Foundation for Intelligent Physical Agents). Эти стандарты описывают то, как

должны взаимодействовать интеллектуальные агенты. Фреймворк включает в себя [WIKI]:

- среду исполнения агентов
- набор графических утилит
- библиотеку классов.

Несмотря на то, что подобные фреймворки предоставляют широкий набор средств для моделирования поведения в мультиагентных системах, было принято решение отказаться от их использования и разработать “с нуля” свою среду в которой можно было бы запускать автономных агентов и следить за их поведением. Это позволит глубже понять как происходит их взаимодействие.

анализ исходных данных

выбор метрик для сравнения решений

Эталоном эффективности для разработанной системы должен быть уровень интеллекта который достигается в колониях муравьев и в стаях пчел. Это лучшие примеры мультиагентных систем, которые были разработаны самой природой, потому что по одиночке ни муравьи ни пчелы не могут выполнить сколь угодно простую задачу. Их кажущееся сложным поведение есть прямой результат того, что они, только собираясь в больших количествах могут при помощи локальных взаимодействий создать систему, которая в сумме будет являться нечто большим чем просто сумма ее частей.

Для сравнения с существующими решениями были выбраны следующие метрики:

- Стоимость
- Время
- Энергозатраты
- Надежность

выбор целевых функций

выбор оптимального варианта решения поставленной задачи

Существует два основных подхода к решению данной проблемы - логическо-пространственный и поведенческий.

В основе лоического подхода лежит знание агента об окружающей обстановке. Каждый агент, использующий данный подход должен быть оборудован неким механизмом логического вывода. Он так же должен быть способен хранить и обновлять модель окружающего его мира в терминах логики предикатов. Обновление модели происходит в результате получения информации от сенсоров и последующего рассуждения. В данном контексте под термином “рассуждение” понимается способность робота принимать решения, основываясь на знаниях об окружающем мире, и способность строить планы на будущее. Эти планы впоследствие становятся конкретными действиями, и могут изменятьсяв зависимости от ситуации. На первый взгляд такой подход кажется естественным, потому, что он основывается на том, как работает человеческое сознание. Но у него есть один

большой минус - связанная с ним вычислительная сложность, которая возникает из-за некоторых фундаментальных проблем математики решение которых еще не было найдено. Несмотря на тот факт, что подобные системы разрабатываются и технологии в наше время быстро развиваются, до сих пор предсталяется фактически невозможным полностью воссоздать механизм работы человеческого мозга.

Поведенческий подход (его так-же можно называть ективным) обычно ставится в противопоставление логическо-пространчивенному. Реактивное принятие решений основывается не на накопленном знании об окружающем мире и не на построенных заранее планах. Оно основывается только на информации о текущей обстановке, здесь и сейчас. Множество примеров подобного поведения можно встретить в мире простейших животных или насекомых. Основная задача, которую решают насекомые - передвижение в поисках пищи.

Без сомнения, логический подход необходим при решении сложных задач, в особенности тех где необходимо показать точный путь и причины выбора того или иного шага на этом пути, например доказательство математических теорем. В своей работе мы покажем, что для задачи планирования движения в контексте мультиагентной системы больше подходит реактивный подход.

выбор методов проектирования,обоснование выбора

Представим ситуацию, когда космический корабль (один из примеров автономного агента) отправляется на другую планету с исследовательской миссией. Основная задача миссии - собрать как

можно больше разлиных образцов породы (объектов интереса) с большой территории. Это хороший пример реальной задачи которую уже способны выполнять роботы. К решению задачи можно подойти с различных позиций.

Одним из таких решений может быть построение сложного робота, способного при помощи логики преобразовывать имеющуюся информацию в конкретные действия. К такой информации может относится например расстояние до ближайших объектов, текущая температура окружающей среды, наличие или отсутствие освещения, и т.д. Так-же обычно такие роботы должны обладать возможностью посылать и принимать радиосигналы высокой дальности для отправки фотографий или принятия указаний к действиям. Таким образом устроены роботы которые сейчас работают на планете Марс, всего их три: Spirit, Opportunity и Curisoity.

С нашей точки зрения такой подход к решению задачи нельзя назвать надежным и быстрым как минимум по двум причинам. Во-первых робот описанный выше должен быть очень хорошо собран, и оттестирован, так как если хотя бы одна его часть (сенсор, шасси, камера и т.д.) перестанет функционировать, то тогда всю миссию можно считать проваленной. То есть первая проблема это трудность сборки единого целого из множества сложных составных частей. Вторая проблема заключается в том, что даже если подобный робот будет способен бесперебойно осуществлять свою деятельность в экстремальных условиях, то для того чтобы охватить большую территорию, ему понадобится много времени.

Другой подход, который можно использовать для решения задачи исследования некоторой поверхности - создание децентрализованной системы роботов небольшого размера. Эти роботы должны быть полностью автономными, и им не понадобятся средства для связи на большом расстоянии. Такая система имеет ряд преимуществ:

• **Масса при транспортировке.** Небольших размеров роботов легче транспортровать, их можно так-же для удобства и надежности транспортировать раздельными группами.

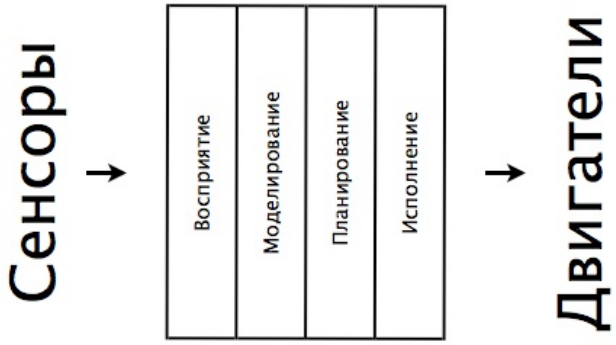
• **Взаимозаменяемость.** При отказе нескольких роботов, подобная система сможет гарантированно выполнить задачу, пусть и за большее время.

• **Децентрализованность.** Роботы в подобной системе могут быть абсолютно автономными и у них нет необходимости подчиняться командам центрального сервера.

• **Стоимость проектирования и производства.** Стоимость подобной системы может быть значительно ниже, чем стоимость одного очень сложного робота.

- перепрограммирование made easy
- масштабирование

Теперь опишем архитектуру отдельно взятого робота. Эта архитектура основывается на работе Brooks [1]. Она отличается от традиционной модели построения системы управления роботом. Так выглядит традиционная модель:



Эта модель состоит из шагов по которым идет любой робот, принимая решения к действию. Сначала происходит срабатывание некоторых сенсоров, затем информация, поступающая с этих сенсоров обрабатывается. После этого робот переводит обработанную информацию в понятную ему модель и проводит на основании этой модели планирование дальнейших действий. Затем эти действия из модели переводятся в конкретные действия движущихся частей (например напряжение и угол поворота моторов).

Шаги, представленные в традиционной архитектуре построения, неотделимы друг от друга, и должны выполняться строго отдин за другим. Основная проблема данной архитектуры в том, что если на одном из шагов произойдет ошибка, то она точно отобразится на всех последующих.

Наша модель построения архиектуры выглядит следующим образом:

Она основывается на конкретных поведениях, которые вместе организованы как слои. Их можно реализовывать отдельно друг от друга, и они могут выполнятся параллельно и независимо. Преимущества подобной архитектуры:

• **Несколько целей.** Слои (или поведения) могут работать одновременно над несколькими задачами, к примеру можно поставить одновременно две задачи: избегать препятствия и достигнуть определенной точки.

• **Надежность.** После того, как один слой разработан и оттестирован, к нему можно уже не возвращаться. Можно быть уверенным что при наслаивании новых поведений, уже существующие не изменятся.

• **Возможность добавления поведений.** Так как слои подобной системы абсолютно не зависят друг от друга, появляется возможность с легкостью добавлять новые поведения.

Взаимодействие отдельных роботов будет относительно простым, и не прямым. За основу модели взаимодействия было

решено взять взаимодействие простейших насекомых, таких как муравьи или пчелы.

выбор программных и аппаратных средств решения поставленной задачи

Было принято решение писать приложение на языке Java, так как это самый распространенный, кросплатформенный язык ООП, поддерживающий многопоточность. Так же вместе со стандартными средствами JavaSE было решено использовать фреймворк JavaFX для визуализации. Это некоммерческая платформа написанная и полностью совместимая с языком Java.

В конце исследовательского раздела приводится постановка задачи на разработку (на исследование).

Глава 2. Реактивное планирование движения, и проблема поиска

---- behavior based motion planning approach

Реактивное планирование движения (в англоязычной литературе часто встречается термин behavior-based) это подход который основан на немедленных реакциях на происходящее вокруг, без использования накопленных знаний об окружающей среде.

Если проводить параллель с миром животных, то к таким немедленным реакциям можно отнести инстинкты. Использование инстинктов характерно для таких навыков как восприятие, передвижение, речь и других. Например, человек не может сказать какую конкретно мышцу он напрягает для того, чтобы воспринять звуковой сигнал, или например, просто улыбнуться. Или профессиональный наборщик текстов на клавиатуре не задумывается над тем, каким пальцем нужно нажимать клавишу, он просто инстинктивно печатает. В основе такого поведения несомненно лежит некоторая реакция, а не просто знание.

Реактивный подход позволяет значительно упростить постановку задачи планирования движения по следующтм причинам:

- **Описание всего пространства в терминах логики предикатов отсутствует.** Не существует центрального элемента, который должен хранить в себе набор фактов. Так же этот набор фактов не должны хранить и сами агенты.

Основная задача, которую решают насекомые - передвижение в поисках пищи.

---------- ЗАДАЧА ПОИСКА

Для разработки системы необходимо было выбрать конкретную задачу. Так как эта система основана на поведении простейших живых организмов, мы решили выбрать из тех задач которые преподносит природа этим организмам. Одна из таких задач - поиск пищи. Из нее мы извлекли следующие условия:

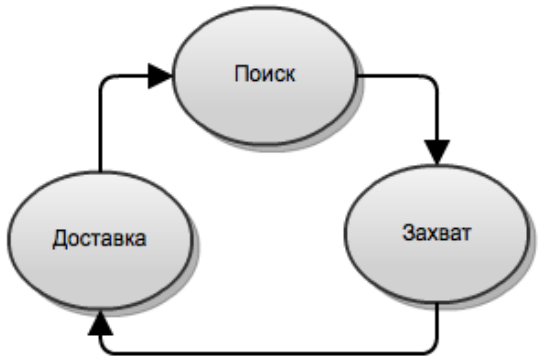
- некоторое ограниченое 2D пространство
- некоторая исходная точка
- ...
- препятствия расставленные некоторым образом
- расположенные в случайных мастах предметы которые нужно с

В реальности эта задача может быть сформулирована в следующих примерах:

- сбор токсичных отходов с места аварии
- исследование других планет или любого другого пространства, недоступного человеку.

Теперь рассмотрим подробнее механику этой задачи. Во время ее выполнения робот может находиться в одном из трех состояний: поиск, захват цели и доставка цели назад к дому. Все роботы начинают свое существование в состоянии поиска. Если

цели нет в пределах видимости сенсоров робта, то он продолжает поиск, как только цель не появится в пределах видимости. В этот момент он переключается в состояние захвата цели. Сразу после захвата цели происходит переход робота в состояние доставки цели до места из которого был произведен его запуск. После доставки цели до необходимой локации робот переходит снова в состояние поиска. Из всех этих состояний можно составить конечный автомат:



Глава 3. Реализация

В этом разделе описывается разработка программного средства – от формальных моделей (проектирования структурных, функциональных схем, UML-моделей, структур баз данных и т.п.) до разработки алгоритмов, программ и пользовательских интерфейсов.

АЛГОРИТМ:

Для решения задачи был предложен алгоритм, основанный на [ref]. Этот алгоритм основан на так называемом “муравьином алгоритме” [ref], который заключается в следующем:

- Муравьи начинают свой поиск в состоянии случайного перемещения.
- После обнаружения еды, они возвращаются обратно в муравейник, оставляя за собой след из специального феромона, который могут почувствовать другие муравьи.
- В тот момент, когда муравей начинает чувствовать тропу из такого феромона, он с гораздо большей вероятностью станет перемещаться по этой тропе, чем просто случайно перемещаться.

Особенность этого механизма в том, что феромонные тропы испаряются со временем. Это значит, что муравьи могут во первых, почувствовать в какую сторону им идти по этой тропе, и во-вторых - отличить тропу по которой ходят чаще, от той, по которой проходят реже. Это так же позволяет муравьям строить самые короткие тропы от источника с едой до муравейника.

Данный алгоритм довольно легко смоделировать на компьютере, и даже сделать визуализацию такой модели. Проблема здесь заключается в том, что попытка перенести подобрую модель в условия рельного мира потерпит неудачу, т.к. очень тяжело, практически невозможно, смоделировать то, как муравьи оставляют на земле некоторое вещество, которое затем может чувствоваться другими муравьями. Подобные попытки были, ученые пытались заменить феромоны на:

- физические пометки пути, такие как спирт, или метки RFID.
- использование беспроводной сети высокой дальности

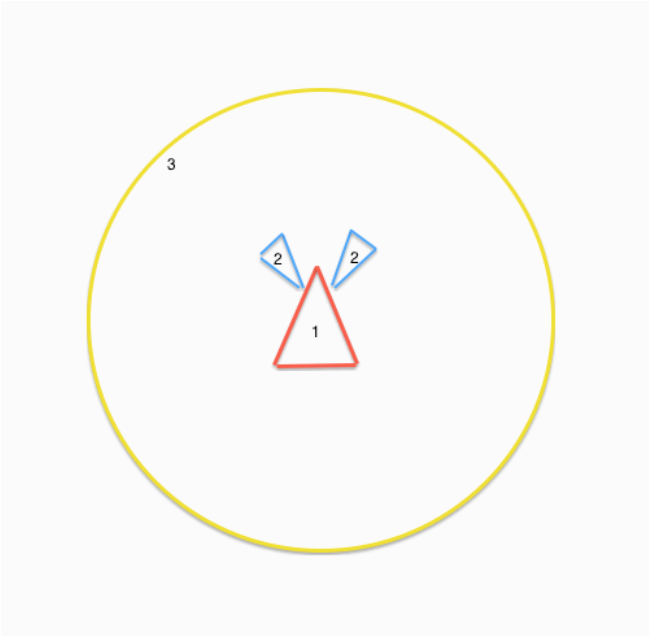
- сенсорные сети, выставленные на исследуемой местности до запуска системы
- маячки которые роботы могут выпускать, или собирать.

У всех этих попыток, которые работали с переменным успехом есть свои недостатки, которые в нашей работе удалось избежать, а именно:

- физические метки поверхности могут быть недопустимы, и временные или разлагающиеся со временем метки тяжело разработать.
- выставление сенсорных сетей заранее очень ограничивает возможную исследуемую область.
- выпускаемые маячки это хороший подход, но для его использования у роботов должен быть механизм, который позволит им выпускать и затем обратно собирать большое количество этих маячков.

В нашем подходе было решено использовать самих роботов как маячков. Подобный подод значительно упрощает оборудование, потому что в нем нет необходимости измерять расстояния или знать собственное плоложение на карте.

Схема:



1. Робот
2. Сенсоры препятствий
3. Радиус радиокоммуникации

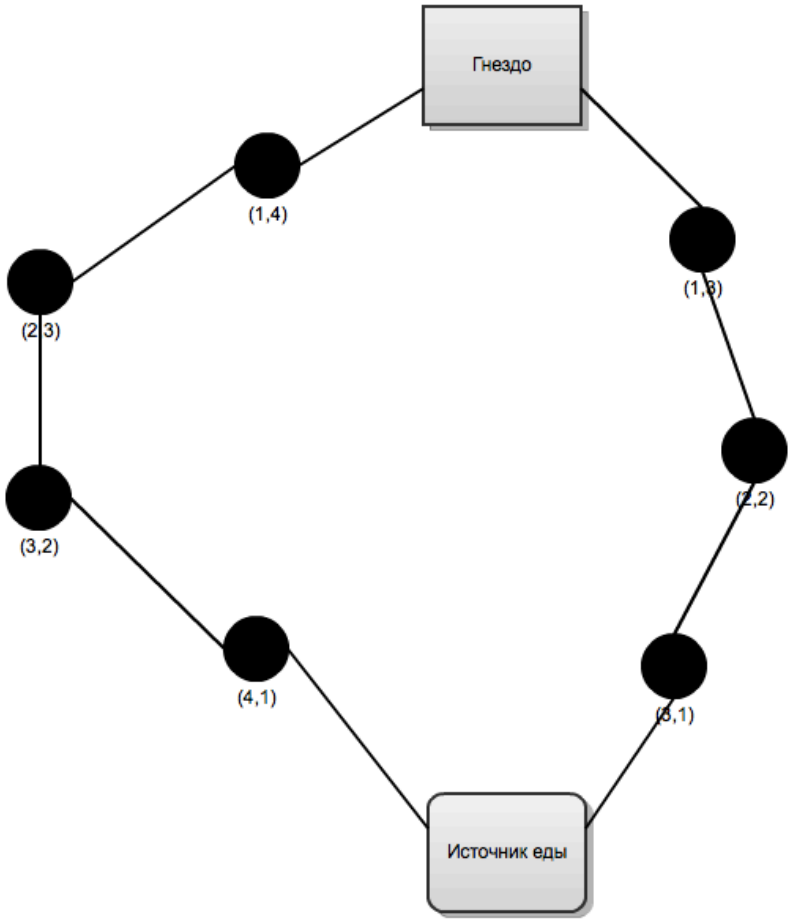
У каждого робота есть сенсоры(2), способные к распознаванию препятствий, источника пищи и, наконец гнезда из которого они были выпущены. Так-же у роботов есть инфракрасная связь небольшого радиуса, позволяющая им на малых расстояниях связываться друг с другом и определять с какой именно стороны идет сигнал.

Описание алгоритма:

Во время выполнения алгоритма роботы могут принимать одно из двух состояний - маяка, или состояние свободного перемещения. Маяки хранят и передают некоторые значения, а роботы находящиеся в состоянии перемещения могут эти значения считывать, и на основе их решать свое текущее поведение. Обычно

в подобных реализациях маяки хранят значение “виртуального феромона”. Это некоторое число, которое можно сопоставить с количеством феромона на земле у настоящих муравьев.

В нашей реализации было решено заменить эти значения на другие. Эти новые значения были названы кардинальностями, и они представляют собой количество звеньев цепи из роботов-маяков до источника с едой, или до гнезда. Схема:



Число в скобках слева показывает количество шагов, которое необходимо сделать, чтобы попасть к гнезду, число справа показывает количество шагов до источника с едой. Первый робот

в состоянии маяка в правой цепочке, который находится непосредственно вблизи гнезда имеет кардинальность (1,3). Робот стоящий рядом с едой имеет кардинальность (3,1). Чтобы добиться правильной расстановки этих кардинальностей, нужно чтобы каждый робот хранил минимальное значение которое он может получить от своих соседей, плюс один.

Роботы, находящиеся в состоянии свободного перемещения могу позже использовать эти значения для выбора правильного маршрута передвижения, двигаясь всегда по направлению к меньшей кардинальности. Две кардинальности необходимо хранить для того, чтобы роботы могли находить кратчайшую дорогу как до источника с едой, так и до гнезда.

Приведем псевдокод для каждого из возможных состояний в которых может находиться робот:

```
первый временной шаг:
перейи в состояние walker
```

```
каждый следующий шаг времени:
   если beacon, делай beacon()
   если walker, делай walker()
```

```
walker()
    если (в радиусе слышимости 2 или более робота в состоянии beacon)
       если (переносишь еду)
          делай Поиск(гнездо)
       иначе
          делай Поиск(еда)
   иначе
          перейти в состояние beacon
```

```
Поиск(что_искать)
   если (что_искать == еда)
      if (в радиусе слышимости маяк)
          иди к самой слабой
          кардинальности еды
      иначе
         исследуй
   иначе
     если (в радиусе слышимости маяк)
          иди к самой слабой
          кардинальности гнезда
      иначе
         исследуй
```

beacon() выставить собственную кардинальность как минимальную из слышимых плюс один	

Параметры, которые можно задавать алгоритму:

- размер пространства, которое необходимо исследовать.
- плотность популяции роботов
- расстояние между гнездом и источником с едой.
- радиус радиокоммуникации между роботами.

Механика:

Nature of code

Для того, чтобы получить работающий код программы для псевдокода, представленного выше необходимо было разработать некий механизм моделирования роботов, того как они будут передвигаться, отрисовываться и т.д.

Подобный механизм был разработан и основан он на книге [ref] . В ней описан несложный способ смоделировать на компьютере физический мир и его явления. Он основывается на векторной математике, и все в нем представляется в виде системы векторов (ускорения, скорости, положения и т.д.)

Подробнее этот подход можно описать на конкретном примере - модели робота. У каждого робота в любой момент

времени имеется набор параметров, которые характеризуют его текущее состояние:

- положение (2D вектор)
- скорость (2D вектор)
- ускорение (2D вектор)

Зная эти параметры мы можем в любой момент отрисовать на экране робота,и высчитать его положение на следующем кадре для последующей отрисовки. Соответственно для того, чтобы появилась динамика, необходимо чтобы параметры менялись. Чтобы они менялись, нужно изменять вектор ускорения, и каждый новый цикл пересчета модели добавлять этот вектор к вектору скорости, и затем добавлять вектор скорости к вектору положения.

Вектор ускорения может изменяться у робота, когда у него, например, появляется цель. Выглядит это следующим образом:

Когда в поле сенсоров попадает некоторый объект интереса (цель), робот может скорректировать свой путь, чтобы попасть к этой цели. Для этого он должен приложить некоторое усилие, чтобы изменить текущую скорость на ту, которая необходима,

чтобы достичь этой цели. Вектор этой силы можно вычислить как разность векторов текущей скорости и желаемой. Каждый новый кадр эта сила будет меняться, в зависимости от того, насколько быстро робот может скорректировать свою текущую скорость.

Конкретно про поведения:

Теперь рассмотрим поведения, которыми были запрограммированы роботы, и наложение и совмещение которых позволило в итоге решить поставленную задачу. Эти поведения распределяются по так называемым уровням компетенции робота. Они могут выполняться параллельно, и, не мешая друг другу, накладываться, в итоге определяя конечное поведение всей модели.

Уровень 0.

На этом уровне робот должен быть способен избегать контакта с другими объектами окружающего мира. Для того, чтобы достигнуть этого уровня, роботу достаточно реагировать на сообщения сенсоров препятствий, расположенных спереди. Для того чтобы избежать столкновения с точкой, нужно вычислить вектор желаемой скорости, который должен быть направлен в обратную сторону от препятствия. В итоге получается поведение Flee() которое может быть использовано впоследствие для решения других задач.

```
public void Flee(MVector target){
    MVector desired = MVector.Subtract(target, location);
```

```
    desired.Limit(maxSpeed);
    steer = MVector.Subtract(velocity, desired);
    steer.Limit(maxForce);
    ApplyForce(steer);
  }
```

Уровень 1.

Этот уровень контроля совместно с первым позволяет роботу случайно перемещаться в свободном пространстве, и тем самым исследовать свое окружение. Для того, чтобы создать эффект случайного перемещения который был бы похож на ередвижение живого организма, было принято решение вместо выбора случайного вектора желаемой скорости каждый новый кадр, немного корректировать направление текущей скорости. Это происходит путем выбора случайного сдвига точки по воображаемой окружности стоящей перед роботом.

```
public void Wander(){
    MVector normilizedSpeed =
            new MVector(getDirection().getX(), getDirection().getY());
    normilizedSpeed.Normalize();
    normilizedSpeed.Multiply(100);

    pointOnCircleAngle += r.nextDouble() - 0.5;
    MVector pointOnCircle = new MVector(pointOnCircleAngle);
    pointOnCircle.Multiply(60);
    normilizedSpeed.Add(pointOnCircle);
    normilizedSpeed.Add(location);

    MVector desired = MVector.Subtract(normilizedSpeed, location);
    desired.Limit(maxSpeed);

    steer = MVector.Subtract(desired, velocity);
    steer.Limit(maxWanderForce);
    ApplyForce(steer);
  }
```

Уровень 2.

На этом уровне

ЗАКЛЮЧЕНИЕ

cooperation between distributed agents through self organisation - conclusion