# Введение

В настоящее время практически во всех промышленно развитых странах интенсивно ведутся работы по созданию и исследованию шагающих роботов. Это вызвано тем, что шагающие машины по сравнению с традиционными колёсными и гусеничными машинами имеют ряд преимуществ перед традиционными транспортными средствами при движении по поверхности со сложным рельефом, такими как пресечённая местность, завалы, а также внутри зданий и сооружений, где необходимо перемещаться по лестницам и узким коридорам и шахтам.

В случаях, когда желательно или необходимо, чтобы след от опор целевой мобильной платформы имел дискретный характер, достойную замену шагающей машине найти невозможно [6]. Шагающий аппарат при движении использует для опоры лишь некоторые точки на поверхности в отличие от колесных и гусеничных машин, имеющих непрерывную колею. Кроме того, шагающий аппарат существенно меньше повреждает почвенный покров, что может оказаться важным для некоторых районов.

Хотя колесные транспортные средства в настоящее время явно преобладают, известно, что при ходьбе по неподготовленной поверхности существенные преимущества имеют шагающие системы передвижения. Однако указанные преимущества шагающего аппарата определяют его высокую сложность. Система управления должна обеспечить переработку информации о местности, принятие решений о характере движения, контроль за их реализацией. Именно создание системы управления аппаратом – центральная проблема шагающего робота, так как опыт создания даже самых сложных систем автоматического управления невозможно непосредственно использовать для построения системы управления шагающим роботом.

# Постановка задачи

Проанализируем преимущества и недостатки шагающих роботов и поставим задачи, которые необходимо решить в рамках данного дипломного проекта.

Как было описано во введении, шагающие роботы, по сравнению с колесными и гусеничными машинами, имеют ряд очевидных преимуществ и недостатков.

Из преимуществ выделим:

* система имеет дискретные точки опоры;
* маневренность в узких пространствах;
* проходимость;

Из недостатков:

* малые скорости перемещения;
* присутствие энергозатрат на удержание в горизонтальной плоскости;
* сложность реализации движения;

Движение шагающего робота можно организовать двумя путями, используя при этом следующие виды походок:

* Статически устойчивые походки – походки, при которых центр масс системы всегда находится внутри многоугольника, образованного точками опоры системы; т.е. если в любой момент времени остановить приводы системы, то она останется в устойчивом положении. Недостатком данного метода является неравномерное распределение нагрузки между опорными ногами (вытекает из замкнутости кинематической схемы), который можно устранить, используя обратную связь не только по позиции, но и по силе (использование силомоментных датчиков). Так же в статически устойчивых походках фаза опоры длится дольше фазы переноса, что сказывается на скоростях движения и энергозатратах;
* Статически неустойчивые походки (динамические походки) – походки, при которых не соблюдается правило устойчивости (см. выше). При таких походках возникает дефицит управления (пример: двуногий антропоморфный робот). Выход из данной ситуации – нахождение периодических устойчивых состояний системы (автоколебания);

Статически устойчивое движение на маленьких скоростях и ускорения легко реализуется на кинематическом уровне. Из условия устойчивости вытекает, что для организации такого вида движения одним типом походки шагающему роботу требуется как минимум 4 ноги с 3-мя степенями свободы. Для увеличения числа возможных вариантов походки требуется увеличить число ног.

Имея шесть ног, перемещение в пространстве можно осуществлять с помощью нескольких вариантов волновых (периодических) походок, основным отличием которых является число ног в фазе опоры/переноса. Самой простой для понимания и реализации является походка «трешками», при которой в фазе опоры всегда находятся три ноги, а три другие в фазе переноса.

Так как создание системы управления аппаратом – это центральная проблема шагающего робота, решим эту проблему, реализовав походку «трешками». А так же решим проблему перемещения – а именно поиск кратчайшего пути по заданной карте местности.

Выбранный робот сделан из легкого акрилового стекла, оснащён ультразвуковым дальномером, девятнадцатью сервоприводами (18 на конечностях и один на ультразвуковом дальномере), полностью готовой к работе плате Servator32, основанной на Arduino.

# Техническое задание

1. Произвести энергетический расчет трехзвенного манипулятора с 3-мя степенями свободы для шестиногого робота. Рассчитать и подобрать двигатели для каждого звена ноги.
2. Произвести частотный синтез и коррекцию следящего привода трехзвенного манипулятора с 3-мя степенями свободы для обеспечения следующих характеристик:

* Частоту среза системы: ;
* Показатель колебательности замкнутой системы: ;
* Время переходного процесса c;
* Нелинейные характеристики системы не учитывать.

1. Решить обратную кинематическую задачу (ОКЗ) для трехзвенного манипулятора с 3-мя степенями свободы.
2. Проанализировать алгоритмы поиска кратчайшего пути.
3. Реализовать прохождение кратчайшего пути по заданной карте местности.
4. Провести натурный эксперимент и определить ошибки в прохождении заданной траектории.
5. Разработать плату стабилизации напряжения.
6. Разработать технологический процесс сборки лапы шестиногого шагающего робота.
7. Рассчитать затраты на проектирование и изготовление шестиногого шагающего робота.
8. Проанализировать опасные и вредные факторы при разработке системы управления шестиногим шагающим роботом.
9. Устранить самый опасный фактор, влияющий на персонал, при разработке системы управления шестиногим шагающим роботом.
10. Проанализировать влияние на окружающую среду технологического процесса сборки печатной платы для системы управления шестиногим шагающим роботом.
11. Устранить самый опасный фактор, влияющий на окружающую среду, при сборке печатной платы для системы управления шестиногим шагающим роботом.

# Научно-исследовательская часть

# Энергетический расчет

Энергетический расчет будем проводить для одной ноги, приняв следующие ограничения и допущения:

* Лапу робота примем за 3х-звенный манипулятор с 3-мя степенями свободы;
* Нагрузкой является каркас (тело робота, к которому крепятся ноги), масса которого сосредоточена в точке и закреплена на конце манипулятора;
* Расчет будем вести от последнего звена, установленного на землю;
* Допускаем, что робот стоит абсолютно жестко, и не проскальзывает в точке контакта ноги с поверхностью опоры;
* Рассматриваем наихудший вариант, т.е. когда нога абсолютно выпрямлена. В этом случае на сочленения действуют самые большие силы и моменты;

Расчет моментов инерции звеньев, приведенных к оси вращения.

Твёрдый параллелепипед с высотой h, шириной a, глубиной b (рис. 2.1 и рис 2.2) и массой m имеет следующие осевые моменты инерции, приведенные к центру одной из граней:

Теорема Гюйгенса — Штейнера [4].

Момент инерции твёрдого тела относительно какой-либо оси зависит от массы, формы и размеров тела, а также и от положения тела по отношению к этой оси. Согласно теореме Штейнера (теореме Гюйгенса-Штейнера), момент инерции тела I относительно произвольной оси равен сумме момента инерции этого тела  относительно оси, проходящей через центр масс тела параллельно рассматриваемой оси, и произведения массы тела *m* на квадрат расстояния *d* между осями:

,

где *m* — полная масса тела.

Момент инерции плеча (рис. 2.1)

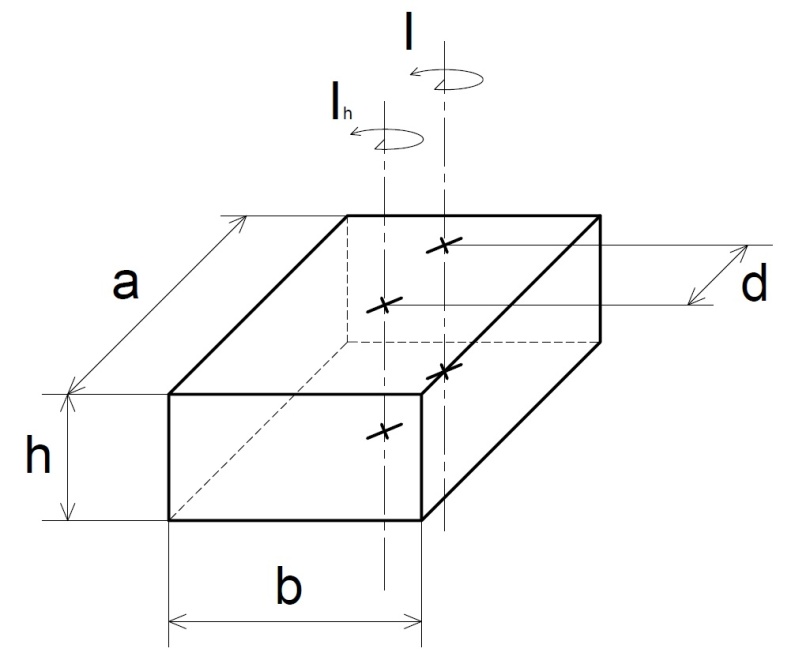
****

Рисунок 2.1. Момент инерции плеча.

Момент инерции предплечья и кисти (рис. 2.2)

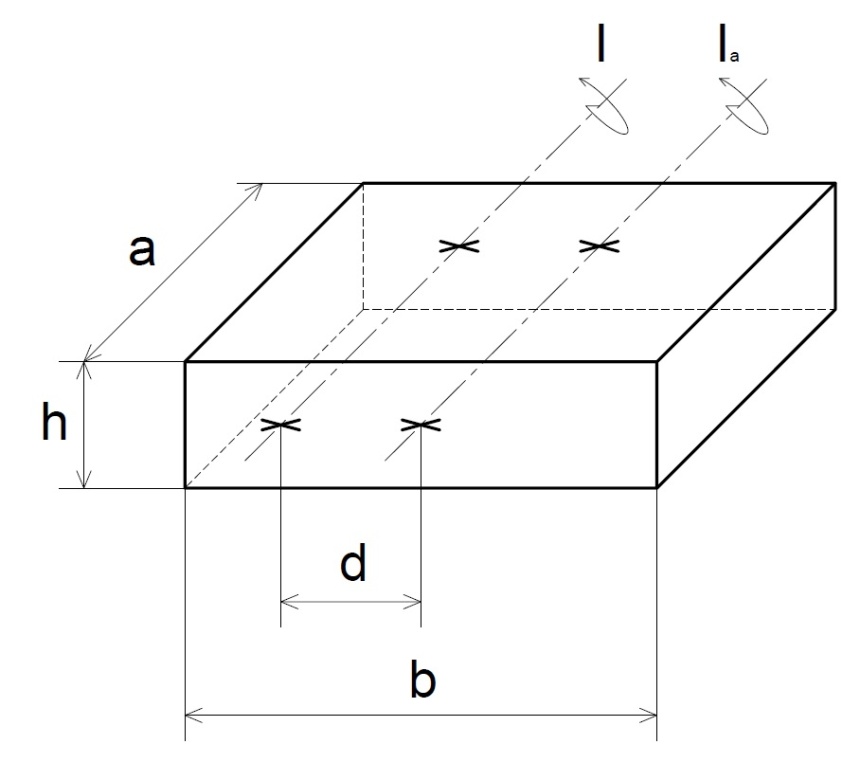
****

Рисунок 2.2. Моменты инерции предплечья и кисти.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № |  |  |  |  |  |
| 1 | 0.030 | 50 | 45 | 10 | 35 |
| 2 | 0.030 | 30 | 70 | 25 | 17 |
| 3 | 0.030 | 53 | 60 | 27 | 10 |

Таблица 2.1. Начальные условия для расчетов.

Начальные условия для расчетов приведены в таблице 2.1.

Расчет момента инерции для плеча:

Расчет момента инерции для предплечья:

Расчет момента инерции для кисти:

Полученные данные сведены в таблицу 2.2.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | Привод |  |  |  |  |  |  |  |  |
| 1 | Плеча | 0.030 | 0.026 | 0.013 |  | 2.5 | 5.0 | 0 | 0,125 |
| 2 | Локтя | 0.030 | 0.050 | 0.025 |  | 2.5 | 5.0 | 0 | 0,125 |
| 3 | Кисти | 0.030 | 0.054 | 0.027 |  | 2.5 | 5.0 | 0 | 0,125 |

Таблица 2.2. Начальные условия для расчетов.

Используемые обозначения в таблице 2.2 :

массы звеньев;

длины звеньев;

радиусы инерции звеньев ();

моменты инерции звеньев, приведенные к осям вращения;

рабочие угловые скорости и ускорения звеньев;

максимально допустимая ошибка копирования;

масса груза (объекта манипулирования; в роли объекта выступает платформа гексапода);

Энергетический расчет ЭМП плеча в I приближении.

Расчет Jп MAX Mп MAX

Расчет механической части ЭМП с объектом управления (ОУ)

Требуемые скорости и ускорения:

Расчет наибольшего значения требуемого момента на валу ОУ

Расчет эквивалентной скорости движения ОУ

Примем предварительно

Расчет граничного значения максимальной требуемой мощности

Для данного расчета зададимся значениями КПД редуктора прямого и обратного хода:

Расчет ориентировочного значения требуемой номинальной мощности двигателя кисти

Для расчета ориентировочного значения требуемой номинальной мощности двигателя кисти необходимо определить теоретическую кратность пускового момента. Для этого достаточно принять

Так номинальную мощность двигателя получим из соображений:

коэффициент кратности пускового момента двигателя.

Выбор двигателя по номинальной мощности с учетом жесткости механической характеристики из таблиц ДПТ

Выбираем по результатам расчетов I приближения ДПТ TowerPro SG92R (Табл. 2.3.)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | кг |  |
|  | 0.675 | 600 | 13 | 0.250 | 0.575 | 0.009 | 11 |

Таблица 2.3. Технические характеристики TowerPro SG92R

Выполним пересчет параметров для выбранного двигателя.

Номинальная угловая скорость:

Номинальный момент двигателя:

Коэффициент момента

Коэффициент противо ЭДС

Выбор усилителя мощности (УМ), расчет его внутреннего сопротивления, вычисление сопротивления якорной цепи УМ-двигатель

действующее сопротивление цепи УМ – ЭД, Ом;

выходное сопротивление усилителя мощности, Ом;

сопротивление обмотки якоря, Ом.

Выбираем транзисторный УМ ()*.* Необходимо назначить коэффициенты форсирования (КФ)

В силу естественного насыщения УМ по выходному напряжению, ЭД присуще ограничение по управляющему напряжению (скорости вращения):

максимальный КФ по напряжению (для транзисторных УМ )

Для получения запаса по управляющему напряжению в расчет принимается допустимое значение управляющего напряжения :

Для ЭД характерно естественное насыщение по электромагнитному моменту:

максимальный КФ по моменту (току) (назначаем )

Допустимый момент двигателя:

КФ двигателя в рабочем режиме (назначаем )

Расчет передаточного отношения редуктора в первом приближении

Требуемый момент на валу двигателя:

Требуемый момент, при котором обеспечивается заданное движение ОУ:

двигатель данному критерию удовлетворяет

Требуемая скорость вращения вала ЭД:

двигатель данному критерию удовлетворяет

Требуемое напряжение цепи якоря ЭД:

Масса звена (плеча)

Энергетический расчет ЭМП локтя в I приближении

Расчет JЛ MAX ; MЛ MAX

Расчет механической части ЭМП с ОУ

Требуемые скорости и ускорения:

Расчет наибольшего значения требуемого момента на валу ОУ

Расчет эквивалентной скорости движения ОУ

Примем предварительно

Расчет граничного значения максимальной требуемой мощности

Для данного расчета зададимся значениями КПД редуктора прямого и обратного хода:

Расчет ориентировочного значения требуемой номинальной мощности двигателя кисти

Для расчета ориентировочного значения требуемой номинальной мощности двигателя кисти необходимо определить теоретическую кратность пускового момента. Для этого достаточно принять

Так номинальную мощность двигателя получим из соображений:

коэффициент кратности пускового момента двигателя.

Выбор двигателя по номинальной мощности с учетом жесткости механической характеристики из таблиц ДПТ

Выбираем по результатам расчетов I приближения ДПТ TowerPro SG92R (Табл. 2.3.)

Выполним пересчет параметров для выбранного двигателя.

Номинальная угловая скорость:

Номинальный момент двигателя:

Коэффициент момента

Коэффициент противоЭДС

Выбор усилителя мощности (УМ), расчет его внутреннего сопротивления, вычисление сопротивления якорной цепи УМ-двигатель

действующее сопротивление цепи УМ – ЭД, Ом;

выходное сопротивление усилителя мощности, Ом;

сопротивление обмотки якоря, Ом.

Выбираем транзисторный УМ (). Необходимо назначить коэффициенты форсирования (КФ)

В силу естественного насыщения УМ по выходному напряжению, ЭД присуще ограничение по управляющему напряжению (скорости вращения):

максимальный КФ по напряжению (для транзисторных УМ )

Для получения запаса по управляющему напряжению в расчет принимается допустимое значение управляющего напряжения :

Для ЭД характерно естественное насыщение по электромагнитному моменту:

максимальный КФ по моменту (току) (назначаем )

Допустимый момент двигателя:

КФ двигателя в рабочем режиме (назначаем )

Расчет передаточного отношения редуктора в первом приближении

Требуемый момент на валу двигателя:

Требуемый момент, при котором обеспечивается заданное движение ОУ:

двигатель данному критерию удовлетворяет

Требуемая скорость вращения вала ЭД:

двигатель данному критерию удовлетворяет

Требуемое напряжение цепи якоря ЭД:

Масса звена (локтя)

Энергетический расчет ЭМП кисти в I приближении

Расчет Jк MAX ; Mк MAX

Расчет механической части ЭМП с ОУ

Требуемые скорости и ускорения:

Расчет наибольшего значения требуемого момента на валу ОУ

Расчет эквивалентной скорости движения ОУ

Примем предварительно

Расчет граничного значения максимальной требуемой мощности

Для данного расчета зададимся значениями КПД редуктора прямого и обратного хода:

Расчет ориентировочного значения требуемой номинальной мощности двигателя кисти

Для расчета ориентировочного значения требуемой номинальной мощности двигателя кисти необходимо определить теоретическую кратность пускового момента. Для этого достаточно принять

Так номинальную мощность двигателя получим из соображений:

коэффициент кратности пускового момента двигателя.

Выбор двигателя по номинальной мощности с учетом жесткости механической характеристики из таблиц ДПТ

Выбираем по результатам расчетов I приближения ДПТ TowerPro SG92R (Табл. 2.3.)

Выполним пересчет параметров для выбранного двигателя.

Номинальная угловая скорость:

Номинальный момент двигателя:

Коэффициент момента

Коэффициент противоЭДС

Выбор усилителя мощности (УМ), расчет его внутреннего сопротивления, вычисление сопротивления якорной цепи УМ-двигатель

действующее сопротивление цепи УМ – ЭД, Ом;

выходное сопротивление усилителя мощности, Ом;

сопротивление обмотки якоря, Ом.

Выбираем транзисторный УМ (). Необходимо назначить коэффициенты форсирования (КФ)

В силу естественного насыщения УМ по выходному напряжению, ЭД присуще ограничение по управляющему напряжению (скорости вращения):

максимальный КФ по напряжению (для транзисторных УМ )

Для получения запаса по управляющему напряжению в расчет принимается допустимое значение управляющего напряжения :

Для ЭД характерно естественное насыщение по электромагнитному моменту:

максимальный КФ по моменту (току) (назначаем )

Допустимый момент двигателя:

КФ двигателя в рабочем режиме (назначаем )

Расчет передаточного отношения редуктора в первом приближении

Требуемый момент на валу двигателя:

Требуемый момент, при котором обеспечивается заданное движение ОУ:

двигатель данному критерию удовлетворяет

Требуемая скорость вращения вала ЭД:

двигатель данному критерию удовлетворяет

Требуемое напряжение цепи якоря ЭД:

Масса звена (кисти)

# Синтез следящего привода трехзвенного манипулятора с 3-мя степенями свободы

Необходимо синтезировать следящий привод, желаемая характеристика которого должна пройти через рабочую точку.

Требуется обеспечить:

* Частоту среза системы: ;
* Показатель колебательности замкнутой системы: ;
* Время переходного процесса c;
* Нелинейные характеристики системы не учитывать.

Расчет будем вести для самого нагруженного привода, т.е. привода кисти.

Исходные данные

Силовая часть привода состоит из усилителя мощности, двигателя постоянного тока, редуктора и датчиков обратной связи.

САР должна обеспечить воспроизведение управляющего воздействия , для которого заданы требуемые скорости и ускорения:

Максимальная ошибка копирования:

В качестве усилителя мощности выбираем транзисторный усилитель мощности (ТУМ) с передаточной функцией *kTP*, где *kTP* - коэффициент усиления ТУМ.

Исходные данные двигателя TowerPro SG92R представлены в табл. 2.4.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | кг |  |
|  | 0.675 | 600 | 13 | 0.250 | 0.575 | 0.009 | 11 |

Таблица 2.6. Технические характеристики TowerPro SG92R.

Передаточное отношение редуктора:

Параметры нагрузки:

Синтез с использование обратных ЛАЧХ

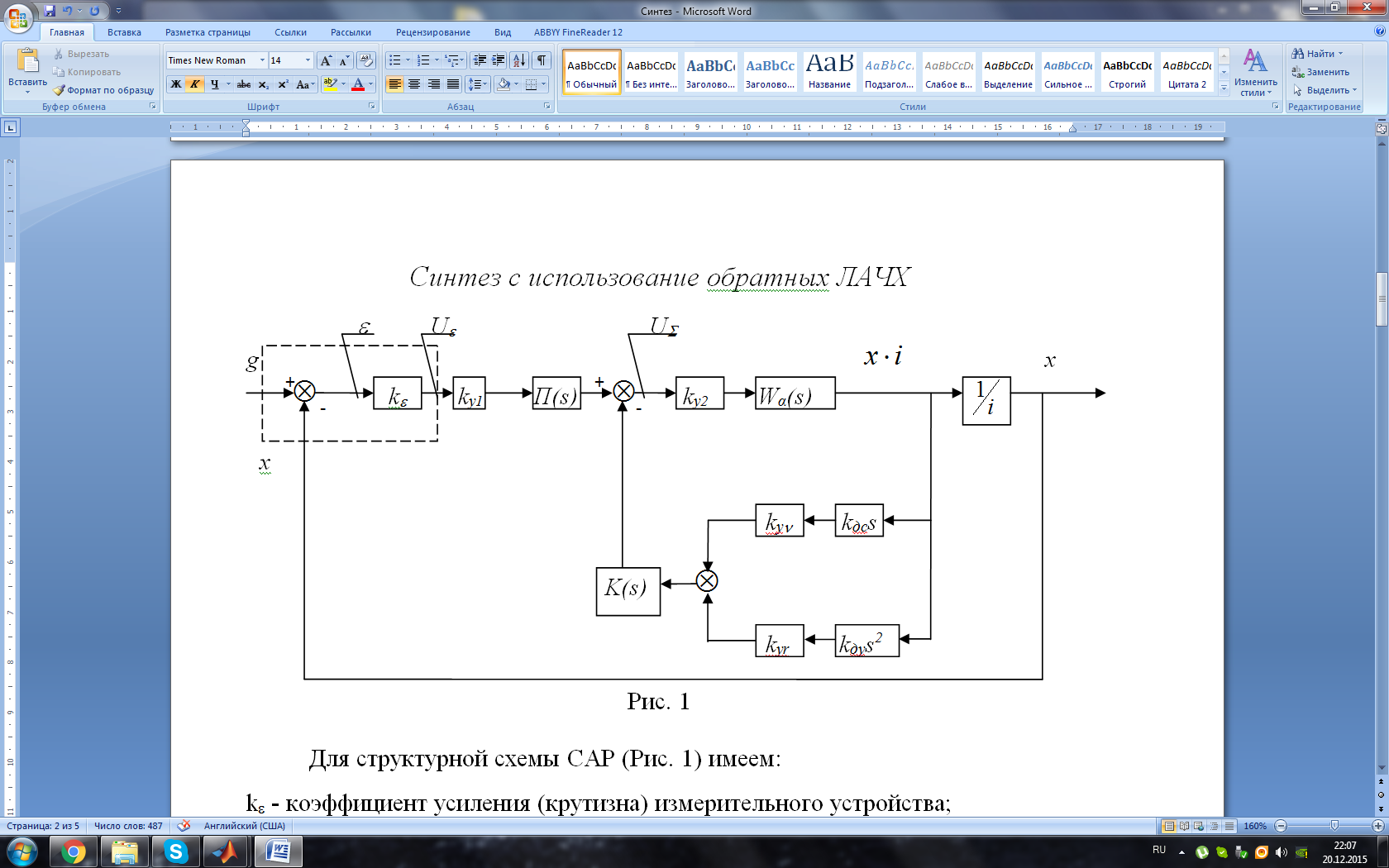
**

Рисунок 2.3. Структурная схема САР.

Для структурной схемы САР (Рис. 2.3) имеем:

kε - коэффициент усиления (крутизна) измерительного устройства;

ky1 - коэффициент усиления первого каскада электронного усилителя (это ФЧВ)

П(s) - передаточная функция последовательного корректирующего устройства:

ky2 - коэффициент усиления второго каскада электронного усилителя;

- передаточная функция исполнительного двигателя (ИД), пусть

где  – коэффициент усиления неизменяемой части ИД;

- электрическая постоянная времени;

- электромеханическая постоянная времени;

- индуктивность обмотки якоря двигателя;

- омическое сопротивление цепи якоря двигателя;

- коэффициент противоЭДС двигателя; - моментный коэффициент;

- момент инерции вала двигателя;

- момент инерции объекта управления;

- передаточное число редуктора;

*kдс* - крутизна датчика скорости вала ИД;

*kду* **-** коэффициент усиления (крутизна) датчика ускорения вала ИД;

*k**yν, kyr* - коэффициенты усиления дополнительных усилителей в цепях обратной связи по скорости и по ускорению соответственно;

*К(s)* - передаточная функция параллельного корректирующего устройства.

Найдем обратную передаточную функцию *W-1(s)* разомкнутой САР.

Имеем:

Введем обозначения:

*μ =* *kε·ky1**·kку·ky2·**kдв·1/i*

*ν = kyν·kдс·ky2·kдв*

*r = kyr·kду·ky2·**kдв*

*μ -* коэффициент усиления разомкнутой системы;

*ν -* коэффициент усиления контура обратной связи по скорости;

*r –* коэффициент усиления контура обратной связи по ускорения.

Тогда вместо (3) имеем:

Формула (4) является очень удобной для проведения синтеза с использованием обратных ЛАЧХ.

Получение передаточной функции двигателя (неизменяемой части)

Для подавляющего большинства ДПТ справедливо:

Таким образом, мы получим:

# Моделирование синтезированной системы

После того как мы провели синтез нашей системы, нам необходимо провести моделирование. Моделирование будем проводить с помощью блоков в графической среде имитационного моделирования Simulink, встроенной в среду MATLAB. Для этого мы собираем структурную схему замкнутой системы с помощью блоков Gain, Transfer Function и Integrator (рис. 2.4.).

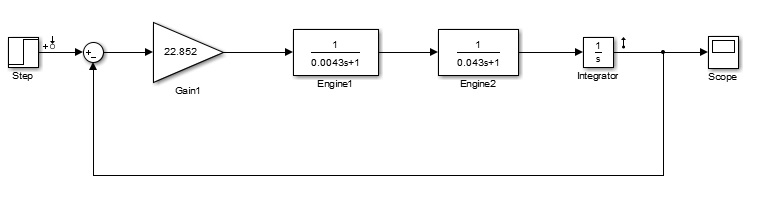


Рисунок 2.4. Структурная схема замкнутой системы.

Смоделировав нашу систему, мы построили ЛАЧХ и ЛФЧХ разомкнутой системы (Приложение 1.1, рис. 10). Так же мы построили ЛАЧХ и ЛФЧХ замкнутой системы (Приложение 1.1, рис. 11). Так же мы построили график переходной функции для замкнутой системы (Приложение 1.1, рис. 12).

Определение запасов устойчивости системы в целом (Приложение 1.1)

Запас устойчивости по амплитуде и фазе достаточный.

Проверка начальных условий:

Коррекция не требуется.

# Решение обратной задачи кинематики

Прямая задача кинематики (ПКЗ) состоит в том, чтобы по заданному вектору координат шарниров МР и известным параметрам его звеньев и сочленений найти положение и ориентацию рабочего органа в декартовом пространстве.

Обратная задача кинематики(ОКЗ) состоит в том, чтобы для заданного положения и ориентации рабочего органа МР в декартовом пространстве и известным параметрам его звеньев и сочленений определить координаты шарниров, обеспечивающие заданное положение.

Прямая задача кинематики всегда имеет единственное решение, обратная задача может не иметь решения, иметь одно или более решений.

Существуют различные методы решения ОКЗ: метод обратных преобразований, геометрический подход, итерационный метод на основе обращения матрицы-Якобиана МР и т.д. Будем использовать геометрический подход [1], который имеет ряд достоинств:

* решение в явном виде и за один проход;
* высокая вычислительная скорость и стабильность;
* однозначность (с учетом геометрической конфигурации);
* сохранение управляемости КМР в «кинематических» внутренних сингулярных точках;
* наглядность.

Недостатком же такого подхода является то, что полученный алгоритм применим только для одной кинематической схемы.

Относительно простое решение ОКЗ при данном подходе обеспечивается за счет того, что кинематические схемы большинства современных МР удовлетворяют одному из двух условий:

* Оси трех смежных шарниров пересекаются в одной точке;
* Оси трех смежных шарниров параллельны между собой.
* Исходными данными для алгоритма решения ОКЗ являются:
* Параметры звеньев и сочленений: длины звеньев, ориентация шарниров в начальном положении МР, минимальные и максимальные допустимые значения углов в шарнирах (постоянные параметры).
* Матрица поворота нулевой СК (СК0) в базовой СК (BASE).
* Знание геометрической конфигурации робота (для разрешения положений с несколькими решениями).
* Значения углов шарниров на предыдущем шаге.

Решение обратной задачи кинематики

За начальное положение (обобщенные координаты равны нулю) ноги примем вытянутое положение в горизонтальной плоскости, направленное из центра шестиугольника.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Таблица 2.5. Начальные условия для расчетов.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Номер ноги |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Таблица 2.6. Начальные условия для расчетов.

Введенные обозначения

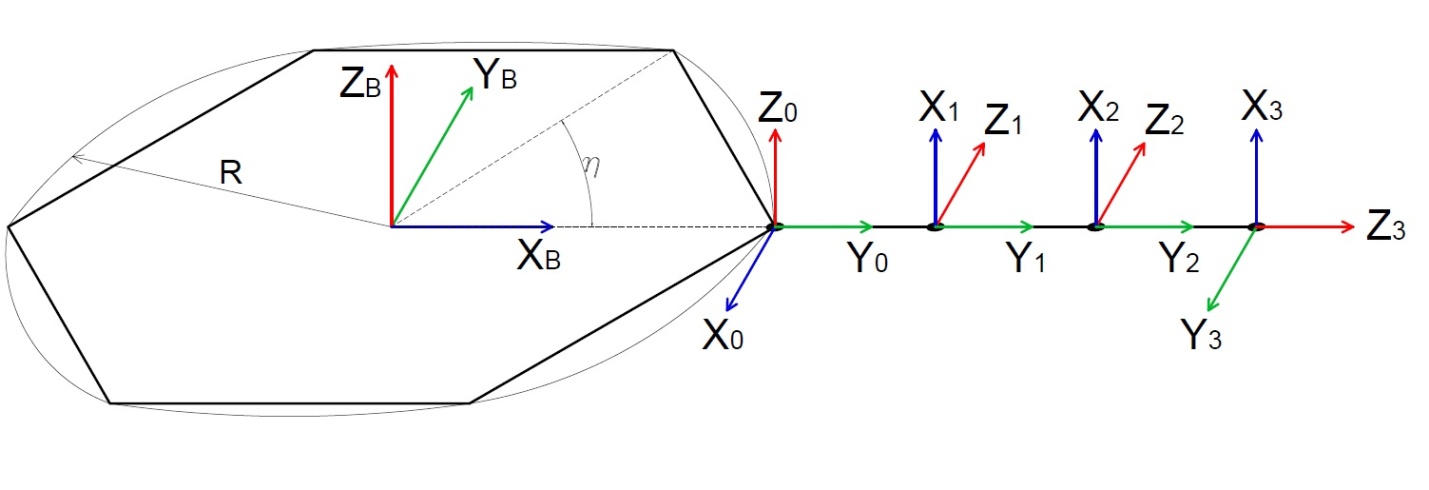
углы поворота относительно вокруг осей , , соответственно;

Длина звена – вектор , проведенный из начала к началу

(в исходном положении -го сочленения); вектор задается в проекциях на оси : = ;

углы поворота относительно вокруг оси ;

линейные длины звеньев (расстояния между осями вращений )

Рисунок 2.5. Кинематическая расчетная схема.

Расчеты обратной задачи кинематики

Для решения ОКЗ требуется определить положение конца манипулятора в декартовом пространстве .

Из курса «ОСНОВЫ РОБОТОТЕХНИКИ» имеем:

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

Определив положение конца манипулятора можем найти угол поворота плеча:

Два других угла найдем, решив следующую геометрическую задачу:

Даны две окружности, каждая определена координатами своего центра и радиусом. Требуется найти все их точки пересечения (либо одна, либо две, либо ни одной точки, либо окружности совпадают).

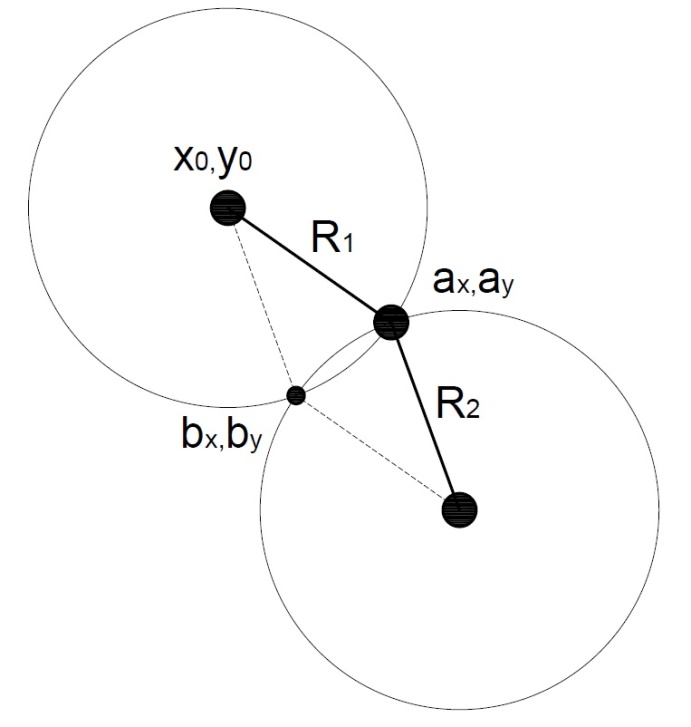


Рисунок 2.6. Задача на пересечение двух окружностей.

Решение:  
Сведём нашу задачу к задаче о Пересечении окружности и прямой.

Предположим, не теряя общности, что центр первой окружности - в начале координат (если это не так, то перенесём центр в начало координат, а при выводе ответа будем обратно прибавлять координаты центра). Тогда мы имеем систему двух уравнений:

Вычтем из второго уравнения первое, чтобы избавиться от квадратов переменных:

Таким образом, мы свели задачу о пересечении двух окружностей к задаче о пересечении первой окружности и следующей прямой:

;

;

;

;

Вместо формального решения системы двух уравнений подойдём к задаче с геометрической стороны (причём, за счёт этого мы получим более точное решение с точки зрения численной устойчивости).

Предположим, не теряя общности, что центр окружности находится в начале координат (если это не так, то перенесём его туда, исправив соответствующе константу C в уравнении прямой). Т.е. имеем окружность с центром в радиуса и прямую с уравнением .

Сначала найдём ближайшую к центру точку прямой - точку с некоторыми координатами . Во-первых, эта точка должна находиться на таком расстоянии от начала координат:

Во-вторых, поскольку вектор перпендикулярен прямой, то координаты этой точки должны быть пропорциональны координатам этого вектора. Учитывая, что расстояние от начала координат до искомой точки нам известно, нам нужно просто нормировать вектор к этой длине, и мы получаем:

(здесь неочевидны только знаки 'минус', но эти формулы легко проверить подстановкой в уравнение прямой - должен получиться ноль)

Зная ближайшую к центру окружности точку, мы уже можем определить, сколько точек будет содержать ответ, и даже дать ответ, если этих точек 0 или 1.

Действительно, если расстояние от до начала координат (а его мы уже выразили формулой - см. выше) больше радиуса, то ответ - ноль точек. Если это расстояние равно радиусу, то ответом будет одна точка - . А вот в оставшемся случае точек будет две, и их координаты нам предстоит найти.

Итак, мы знаем, что точка лежит внутри круга. Искомые точки и , помимо того что должны принадлежать прямой, должны лежать на одном и том же расстоянии от точки , причём это расстояние легко найти:

Заметим, что вектор коллинеарен прямой, а потому искомые точки и можно получить, прибавив к точке вектор , нормированный к длине (мы получим одну искомую точку), и вычтя этот же вектор (получим вторую искомую точку).

Окончательное решение такое:

;

;

;

;

Зная координаты центра окружности и координаты точек пересечения, однозначно определяем углы .

# Моделирование решения ОКЗ

Моделирование решения обратной кинематической задачи[2] производится в два этапа. Сначала с помощью блоков в графической среде имитационного моделирования Simulink, встроенной в среду MATLAB, собирается модель конечности робота. Затем в m-файле прописывается алгоритм вычисления углов поворота сервоприводов исходя из решенной обратной кинематической задачи для всей ноги. Таким образом, на каждый двигатель подается угол, рассчитанный ранее в m-файле (приложение 1).

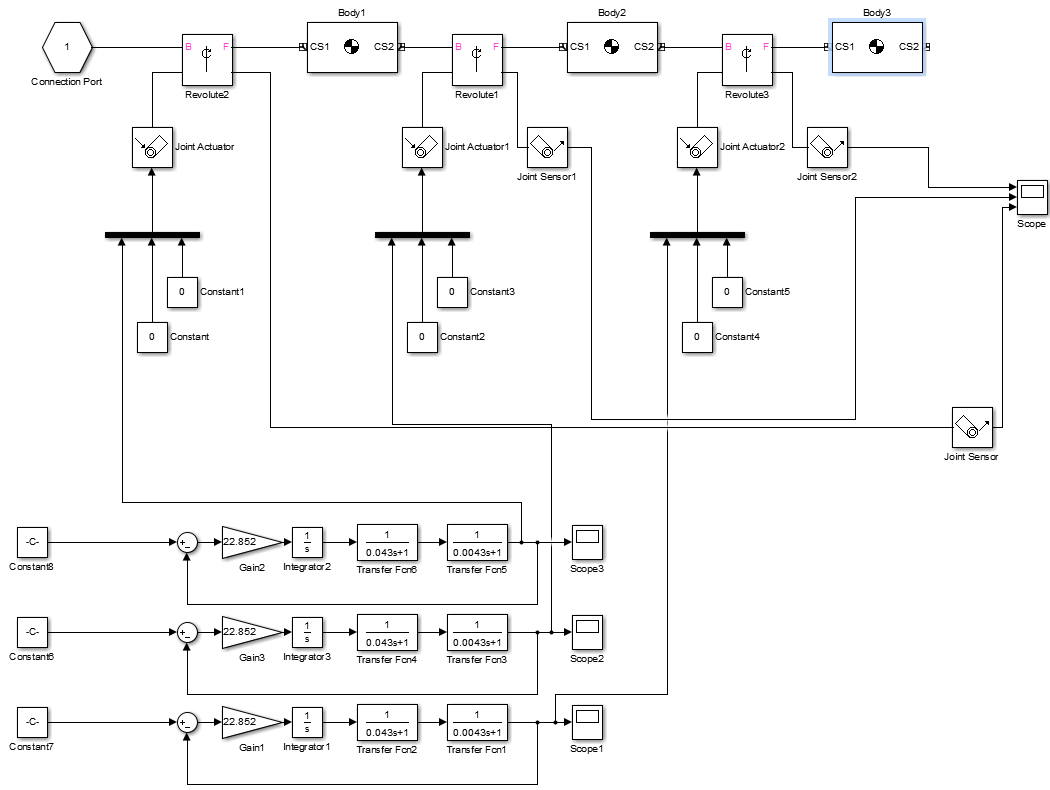


Рисунок 2.7. Блок-схема ноги робота.

На данной схеме представлены следующие блоки:

* Body1, Body2, Body3 ­– плечо, предплечье, кисть;
* Connection port – соединение кисти с платформой;
* Revolute2, Revolute1, Revolute3 – шарниры соединения;
* Constant8, Constant6, Constant7 – углы, подаваемые на двигатели;
* Через JointActuator’ы подается угол, скорость и ускорение на шарнир;
* C JointSensor снимаются данные с шарниров;
* Scope отображает графики изменения положения шарниров (рис. 2.8.).

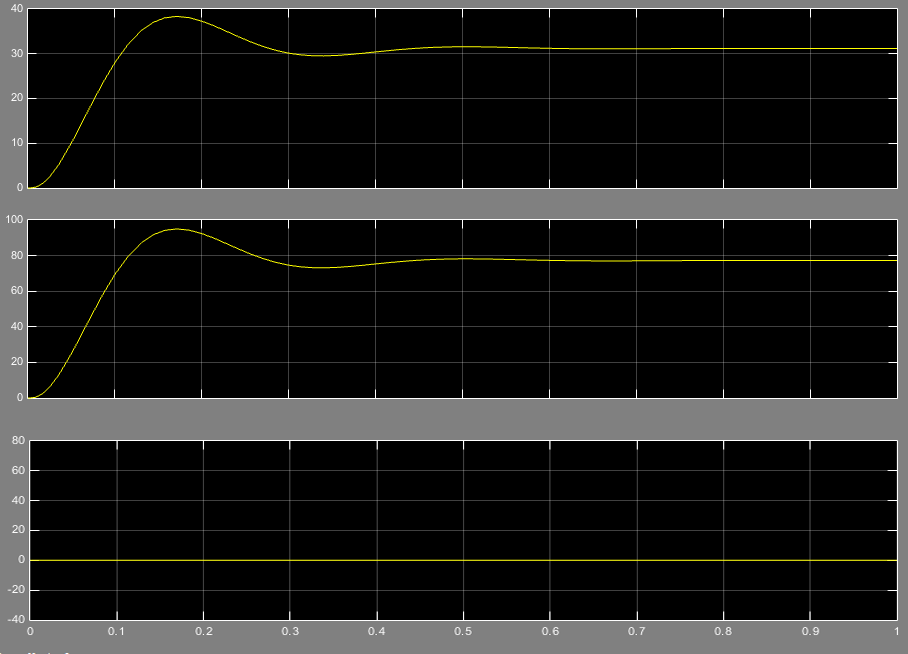


Рисунок 2.8. Графики изменения положения звеньев ноги #5.

M-файл, с помощью которого были произведены расчеты, находится в приложении 1.

Зная длины сочленений и координаты конца ноги, произведем расчет поворота углов звеньев ноги #5 вручную, чтобы проверить правильность расчета программы.

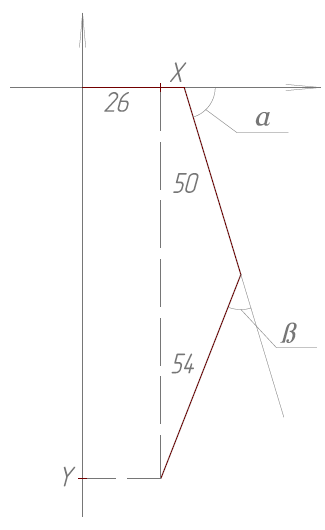


Рисунок 2.9. Схема поворота ноги #5.

В данном случае плечо неподвижно. Исходя из расчётов программы, угол α и β соответственно равны 77.2° и 31.2°. При таких значениях α и β искомые X и Y считаются по формулам:

Полученные значения X и Y есть наши изначально установленные координаты, это значит, что программа работает верно.

# Программный код для управления движения шагающим роботом

Описание алгоритма управления роботом [5]

В первую очередь требуется установить связь с аппаратной частью робота. Связь обеспечивается через последовательное (Serial) проводное соединение, с помощью стандартных библиотек OS Windows.

Далее нужно ввести координаты точки назначения в декартовых координатах (в базовой системе координат ).

На стратегическом уровне программа определяет угол курса () и расстояние прямолинейного движения () по посчитанному курсу. После вычисления и вызываются функции тактического уровня, в которых выполняются расчеты и перемещение на и .

В силу ограничения подвижности габаритами элементов, робот не может повернуться сразу на угол и пройти на расстояние . Исходя из этого движения и разбиваются на части меньшие фиксированной длинны и угла, т.е. элементарные угловые и линейные перемещения.

Для каждого элементарного перемещения задаются вектора управления концами ног. Учитывая текущее положение и вектор управления, решаем ОКЗ для каждой ноги.

После решения ОКЗ и получения углов шарниров требуется сформировать сообщение для последовательного соединения по заданному протоколу. Нулевое положение принимается за 1500мс. .

Отправка сообщений происходит с задержками, принятыми для завершения переходных процессов (см. Синтез). Для исключения ударов ноги, при опускании ее на землю, введена следующая последовательность отработки углов: ;

Достигнув заданную точку, программа завершается.

# Анализ алгоритмов поиска кратчайшего пути

Задача о кратчайшем пути — задача поиска самого короткого пути (цепи) между двумя точками (вершинами) на графе, в которой минимизируется сумма весов ребер, составляющих путь.

Кратчайшая (простая) цепь часто называется геодезической.

Задача о кратчайшем пути является одной из важнейших классических задач теории графов. Сегодня известно множество алгоритмов для её решения.

У данной задачи существуют и другие названия: задача о минимальном пути или, в устаревшем варианте, задача о дилижансе.

Значимость данной задачи определяется её различными практическими применениями. Например в GPS-навигаторах, где осуществляется поиск кратчайшего пути между двумя перекрестками. В качестве вершин выступают перекрестки, а дороги являются ребрами, которые лежат между ними. Сумма расстояний всех дорог между перекрестками должна быть минимальной, тогда найден самый короткий путь. В нашем случае мы находим кратчайший путь от точки А до точки Б исходя из заданной карты местности и расположения начальной и конченых точек. В качестве вершин выступают неровности местности, кратчайшие возможные расстояния между этими вершинами являются ребрами.

Существуют различные постановки задачи о кратчайшем пути:

* Задача о кратчайшем пути в заданный пункт назначения.

Требуется найти кратчайший путь в заданную вершину назначения t, который начинается в каждой из вершин графа (кроме t). Поменяв направление каждого принадлежащего графу ребра, эту задачу можно свести к задаче о единой исходной вершине (в которой осуществляется поиск кратчайшего пути из заданной вершины во все остальные).

* Задача о кратчайшем пути между заданной парой вершин.

Требуется найти кратчайший путь из заданной вершины u в заданную вершину v.

* Задача о кратчайшем пути между всеми парами вершин.

Требуется найти кратчайший путь из каждой вершины u в каждую вершину v. Эту задачу тоже можно решить с помощью алгоритма, предназначенного для решения задачи об одной исходной вершине, однако обычно она решается быстрее.

В различных постановках задачи, роль длины ребра могут играть не только сами длины, но и время, стоимость, расходы, объем затрачиваемых ресурсов (материальных, финансовых, топливно-энергетических и т. п.) или другие характеристики, связанные с прохождением каждого ребра. Таким образом, задача находит практическое применение в большом количестве областей (информатика, экономика, география и др.).

В связи с тем, что существует множество различных постановок данной задачи, есть наиболее популярные алгоритмы для решения задачи поиска кратчайшего пути на графе:

Алгоритм Дейкстры находит кратчайший путь от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса.

Алгоритм Беллмана — Форда находит кратчайшие пути от одной вершины графа до всех остальных во взвешенном графе. Вес ребер может быть отрицательным.

Алгоритм поиска A\* находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной), используя алгоритм поиска по первому наилучшему совпадению на графе.

Алгоритм Флойда — Уоршелла находит кратчайшие пути между всеми вершинами взвешенного ориентированного графа.

Алгоритм Джонсона находит кратчайшие пути между всеми парами вершин взвешенного ориентированного графа.

Алгоритм Ли (волновой алгоритм) основан на методе поиска в ширину. Находит путь между вершинами s и t графа (s не совпадает с t), содержащий минимальное количество промежуточных вершин (ребер).

Рассмотрим три самых популярных алгоритма и выберем оптимальный для нашей задачи:

* Алгоритм Дейкстры;
* Алгоритм Беллмана — Форда;
* Алгоритм Ли (волновой алгоритм).

Алгоритм Дейкстры

Принцип работы алгоритма Дейкстры - нахождение оптимальных маршрутов и их длины между одной конкретной вершиной (источником) и всеми остальными вершинами графа. Недостаток данного алгоритма в том, что он работает некорректно, если граф имеет дуги отрицательного веса.

Присвоим 1-й вершине метку равную 0, потому как эта вершина — источник. Остальным вершинам присвоим метки равные бесконечности.

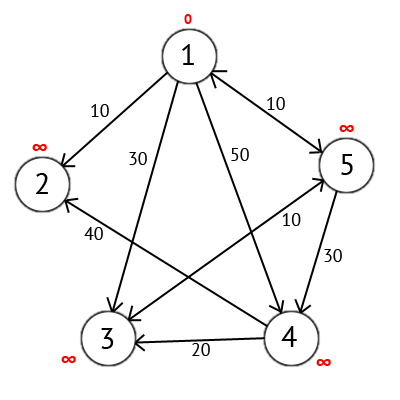
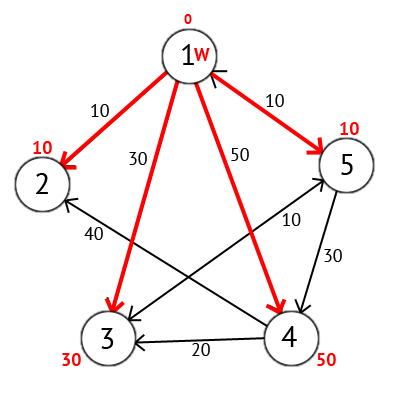


Рис. 2.10. Начальный этап алгоритма Дейкстры.

Далее выберем такую вершину W, которая имеет минимальную метку (сейчас это вершина 1) и рассмотрим все вершины в которые из вершины W есть путь, не содержащий вершин посредников. Каждой из рассмотренных вершин назначим метку равную сумме метки W и длинны пути из W в рассматриваемую вершину, но только в том случае, если полученная сумма будет меньше предыдущего значения метки. Если же сумма не будет меньше, то оставляем предыдущую метку без изменений.

  
Рис. 2.11. Второй этап алгоритма Дейкстры.

После того как мы рассмотрели все вершины, в которые есть прямой путь из W, вершину W мы отмечаем как посещённую, и выбираем из ещё не посещенных такую, которая имеет минимальное значение метки, она и будет следующей вершиной W. В данном случае это вершина 2 или 5. Если есть несколько вершин с одинаковыми метками, то не имеет значения, какую из них мы выберем как W.

Мы выберем вершину 2. Но из нее нет ни одного исходящего пути, поэтому мы сразу отмечаем эту вершину как посещенную и переходим к следующей вершине с минимальной меткой. На этот раз только вершина 5 имеет минимальную метку. Рассмотрим все вершины в которые есть прямые пути из 5, но которые ещё не помечены как посещенные. Снова находим сумму метки вершины W и веса ребра из W в текущую вершину, и если эта сумма будет меньше предыдущей метки, то заменяем значение метки на полученную сумму.

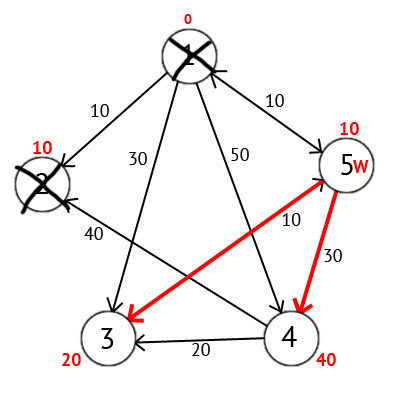


Рис. 2.12. Третий этап алгоритма Дейкстры.

Исходя из картинки, мы можем увидеть, что метки 3-ей и 4-ой вершин стали меньше, то есть был найден более короткий маршрут в эти вершины из вершины источника. Далее отмечаем 5-ю вершину как посещенную и выбираем следующую вершину, которая имеет минимальную метку. Повторяем все перечисленные выше действия до тех пор, пока есть непосещенные вершины.

Выполнив все действия, получим такой результат:

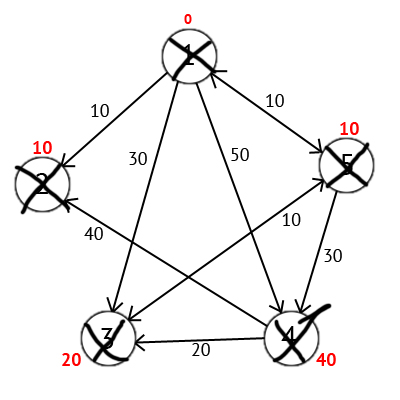


Рис. 2.13. Заключительный этап алгоритма Дейкстры.

Также есть вектор Р, исходя из которого можно построить кратчайшие маршруты. По количеству элементов этот вектор равен количеству вершин в графе, Каждый элемент содержит последнюю промежуточную вершину на кратчайшем пути между вершиной-источником и конечной вершиной. В начале алгоритма все элементы вектора Р равны вершине источнику (в нашем случае Р = {1, 1, 1, 1, 1}). Далее на этапе пересчета значения метки для рассматриваемой вершины, в случае если метка рассматриваемой вершины меняется на меньшую, в массив Р мы записываем значение текущей вершины W. Например: у 3-ей вершины была метка со значением «30», при W=1. Далее при W=5, метка 3-ей вершины изменилась на «20», следовательно мы запишем значение в вектор Р — Р[3]=5. Также при W=5 изменилось значение метки у 4-й вершины (было «50», стало «40»), значит нужно присвоить 4-му элементу вектора Р значение W — P[4]=5. В результате получим вектор Р = {1, 1, 5, 5, 1}.

Зная что в каждом элементе вектора Р записана последняя промежуточная вершина на пути между источником и конечной вершиной, мы можем получить и сам кратчайший маршрут.

Алгоритм Беллмана — Форда

Пусть дан ориентированный взвешенный граф G с n вершинами и m рёбрами, и указана некоторая вершина v. Требуется найти длины кратчайших путей от вершины v до всех остальных вершин.

В отличие от алгоритма Дейкстры, этот алгоритм применим также и к графам, содержащим рёбра отрицательного веса. Впрочем, если граф содержит отрицательный цикл, то, понятно, кратчайшего пути до некоторых вершин может не существовать (по причине того, что вес кратчайшего пути должен быть равен минус бесконечности); впрочем, этот алгоритм можно модифицировать, чтобы он сигнализировал о наличии цикла отрицательного веса, или даже выводил сам этот цикл.

Алгоритм носит имя двух американских учёных: Ричарда Беллмана (Richard Bellman) и Лестера Форда (Lester Ford). Форд фактически изобрёл этот алгоритм в 1956 г. при изучении другой математической задачи, подзадача которой свелась к поиску кратчайшего пути в графе, и Форд дал набросок решающего эту задачу алгоритма. Беллман в 1958 г. опубликовал статью, посвящённую конкретно задаче нахождения кратчайшего пути, и в этой статье он чётко сформулировал алгоритм в том виде, в котором он известен нам сейчас.

Формулировка задачи

Дан ориентированный или неориентированный граф G со взвешенными рёбрами. Длиной пути назовём сумму весов рёбер, входящих в этот путь. Требуется найти кратчайшие пути от выделенной вершины s до всех вершин графа.

Заметим, что кратчайших путей может не существовать. Так, в графе, содержащем цикл с отрицательным суммарным весом, существует сколь угодно короткий путь от одной вершины этого цикла до другой (каждый обход цикла уменьшает длину пути). Цикл, сумма весов рёбер которого отрицательна, называется отрицательным циклом.

Решение задачи на графе без отрицательных циклов

Решим поставленную задачу на графе, в котором заведомо нет отрицательных циклов.

Для нахождения кратчайших путей от одной вершины до всех остальных, воспользуемся методом [динамического программирования](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Построим [матрицу](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) Aij, элементы которой будут обозначать следующее: Aij — это длина кратчайшего пути из s в i, содержащего не более *j* рёбер.

Путь, содержащий 0 рёбер, существует только до вершины *s*. Таким образом, Ai0 равно 0 при *i* = *s*, и +∞ в противном случае.

Теперь рассмотрим все пути из *s* в *i*, содержащие ровно *j* рёбер. Каждый такой путь есть путь из j-1 ребра, к которому добавлено последнее ребро. Если про пути длины j-1 все данные уже подсчитаны, то определить *j*-й столбец матрицы не составляет труда.

Так выглядит алгоритм поиска длин кратчайших путей в графе без отрицательных циклов:

**for** v \in V

**do** d[v] \gets +\infty

d[s] \gets 0

**for** i \gets 1 **to** |V| - 1

**do for** (u, v) \in E

**if** d[v] > d[u] + w(u, v)

**then** d[v] \gets d[u] + w(u, v)

**return** d

Здесь *V* — множество вершин графа *G*, *E* — множество его рёбер, а *w* — весовая функция, заданная на ребрах графа (возвращает длину [дуги](https://ru.wikipedia.org/wiki/%D0%94%D1%83%D0%B3%D0%B0_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)#.D0.B4.D1.83.D0.B3.D0.B0), ведущей из вершины *u* в *v*), d - массив, содержащий расстояния от вершины s до любой другой вершины.

Внешний цикл выполняется |V| - 1 раз, поскольку кратчайший путь не может содержать большее число ребер, иначе он будет содержать цикл, который точно можно выкинуть.

Вместо массива *d* можно хранить всю матрицу A, но это требует *O(V²)* памяти. Зато при этом можно вычислить и сами кратчайшие пути, а не только их длины. Для этого заведем матрицу *Pij*.

Если элемент *Aij* содержит длину кратчайшего пути из *s* в *i*, содержащего *j* рёбер, то *Pij* содержит предыдущую вершину до *i* в одном из таких кратчайших путей (ведь их может быть несколько).

Теперь алгоритм Беллмана–Форда выглядит так:

**for** v \in V

**for** i \gets 0 **to** |V| - 1

**do** A_{vi} \gets +\infty

A_{s0} \gets 0

**for** i \gets 1 **to** |V| - 1

**do for** (u, v) \in E

**if** A_{vi} > A_{u, i-1} + w(u, v)

**then** A_{vi} \gets A_{u, i-1} + w(u, v)

P_{vi} \gets u

После выполнения этого алгоритма элементы A_{i, j} содержат длины кратчайших путей от *s* до *i* с количеством ребер *j*, и из всех таких путей следует выбрать самый короткий. А сам кратчайший путь до вершины *i* с *j* ребрами восстанавливается так:

**while** j > 0

p[j] \gets i

i \gets P_{ij}

j \gets j - 1

**return** *p*

Граф с отрицательными циклами

Алгоритм Беллмана–Форда позволяет очень просто определить, существует ли в графе *G* отрицательный цикл, достижимый из вершины *s*. Достаточно произвести внешнюю итерацию цикла не |V| - 1, a ровно |*V*| раз. Если при исполнении последней итерации длина кратчайшего пути до какой-либо вершины строго уменьшилась, то в графе есть отрицательный цикл, достижимый из *s*. На основе этого можно предложить следующую оптимизацию: отслеживать изменения в графе и, как только они закончатся, сделать выход из цикла (дальнейшие итерации будут бессмысленны).

Алгоритм Ли

Алгоритм Ли так же известен как Волновой алгоритм или Алгоритм волновой трассировки. Алгоритм работает на дискретном рабочем поле (ДРП), представляющем собой ограниченную замкнутой линией фигуру, не обязательно прямоугольную, разбитую на прямоугольные ячейки, в частном случае — квадратные. Множество всех ячеек ДРП разбивается на подмножества: «проходимые» (свободные), т. е при поиске пути их можно проходить, «непроходимые» (препятствия), путь через эту ячейку запрещён, стартовая ячейка (источник) и финишная (приемник). Назначение стартовой и финишной ячеек условно, достаточно — указание пары ячеек, между которыми нужно найти кратчайший путь.

Алгоритм предназначен для поиска кратчайшего пути от стартовой ячейки к конечной ячейке, если это возможно, либо, при отсутствии пути, выдать сообщение о непроходимости.

Работа алгоритма включает в себя три этапа: инициализацию, распространение волны и восстановление пути.

Во время инициализации строится образ множества ячеек обрабатываемого поля, каждой ячейке приписываются атрибуты проходимости/непроходимости, запоминаются стартовая и финишная ячейки.

Далее, от стартовой ячейки порождается шаг в соседнюю ячейку, при этом проверяется, проходима ли она, и не принадлежит ли ранее меченной в пути ячейке.

Соседние ячейки принято классифицировать двояко: в смысле [окрестности Мура](https://ru.wikipedia.org/wiki/%D0%9E%D0%BA%D1%80%D0%B5%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%9C%D1%83%D1%80%D0%B0) и [окрестности фон Неймана](https://ru.wikipedia.org/wiki/%D0%9E%D0%BA%D1%80%D0%B5%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D1%84%D0%BE%D0%BD_%D0%9D%D0%B5%D0%B9%D0%BC%D0%B0%D0%BD%D0%B0), отличающийся тем, что в окрестности фон Неймана соседними ячейками считаются только 4 ячейки по вертикали и горизонтали, в окрестности Мура — все 8 ячеек, включая диагональные.

При выполнении условий проходимости и непринадлежности её к ранее помеченным в пути ячейкам, в атрибут ячейки записывается число, равное количеству шагов от стартовой ячейки, от стартовой ячейки на первом шаге это будет 1. Каждая ячейка, меченая числом шагов от стартовой ячейки становится стартовой и из неё порождаются очередные шаги в соседние ячейки. Очевидно, что при таком переборе будет найден путь от начальной ячейки к конечной, либо очередной шаг из любой порождённой в пути ячейки будет невозможен.

Восстановление кратчайшего пути происходит в обратном направлении: при выборе ячейки от финишной ячейки к стартовой на каждом шаге выбирается ячейка, имеющая атрибут расстояния от стартовой на единицу меньше текущей ячейки. Очевидно, что таким образом находится кратчайший путь между парой заданных ячеек. Трасс с минимальной числовой длиной пути, как при поиске пути в окрестностях Мура, так и фон Неймана может существовать несколько. Выбор окончательного пути в приложениях диктуется другими соображениями, находящимися вне этого алгоритма.

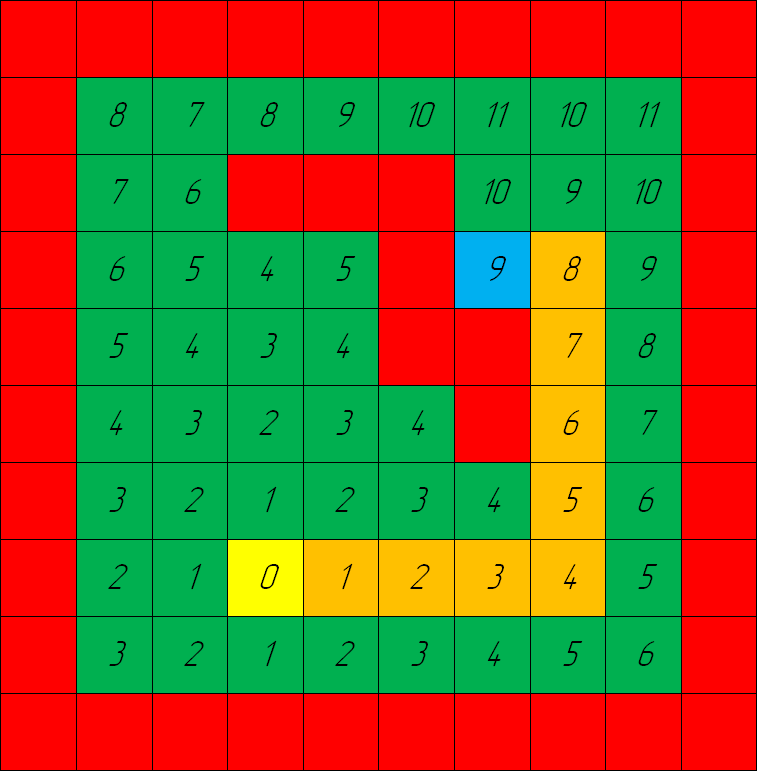
Результат работы волнового алгоритма представлен на рисунке 2.14.

Рис. 2.14. Результат работы волнового алгоритма.

Основное преимущество данного алгоритма заключается в легкости его реализации. Так же, реализовывая данный алгоритм, мы получаем массив чисел, пронумерованных по номеру движения и имеющий определенные, ранее заложенные действия для нашего робота.

Проанализировав основные алгоритмы, выясняем, что самый оптимальный для решения нашей задачи – это Волновой алгоритм или Алгоритм Ли. Реализация в рамках данной дипломной работы будет осуществленная на языке программирования C++.

# Реализация Алгоритма Ли для поиска кратчайшего пути по заданной карте местности

Как отмечено выше, структура кода разделена на три части: инициализация, распространение волны и восстановление пути.

В первой части инициализации мы дискретизируем карту местности, задаем ее габариты, отмечаем непроходимые и проходимые участки, а так же задаем начало и конец пути.

Для того, чтобы реализовать алгоритм Ли, нам необходимо задать карту местности, по которой будет перемещаться робот. Для этого нужно проанализировать имеющийся участок на наличие неровностей рельефа, в том числе проходимых и непроходимых участков. После оценки площади обрабатываемой местности, необходимо задать точность дискретизации, то есть размер одной ячейки карты местности. Для удобства расчетов, зададим длину и ширину ячейки карты местности равными расстоянию между передними и задними конечностями робота. Это нужно для того, чтобы за одно перемещение вперед робот перемещался ровно в соседнюю клетку на карте местности. Расстояние между передними и задними конечностями составляет 220мм. Соответственно, габариты одной ячейки зададим равными 220мм на 220мм.

Имея габариты одной клетки, разобьем всю карту местности на ячейки и обозначим непроходимые участки цифрой 0, а проходимые цифрой 1, как представлено на рисунке 2.15. В следующем шаге мы изменим значения ячеек, но на первом этапе, для наглядности, обозначим их именно так.

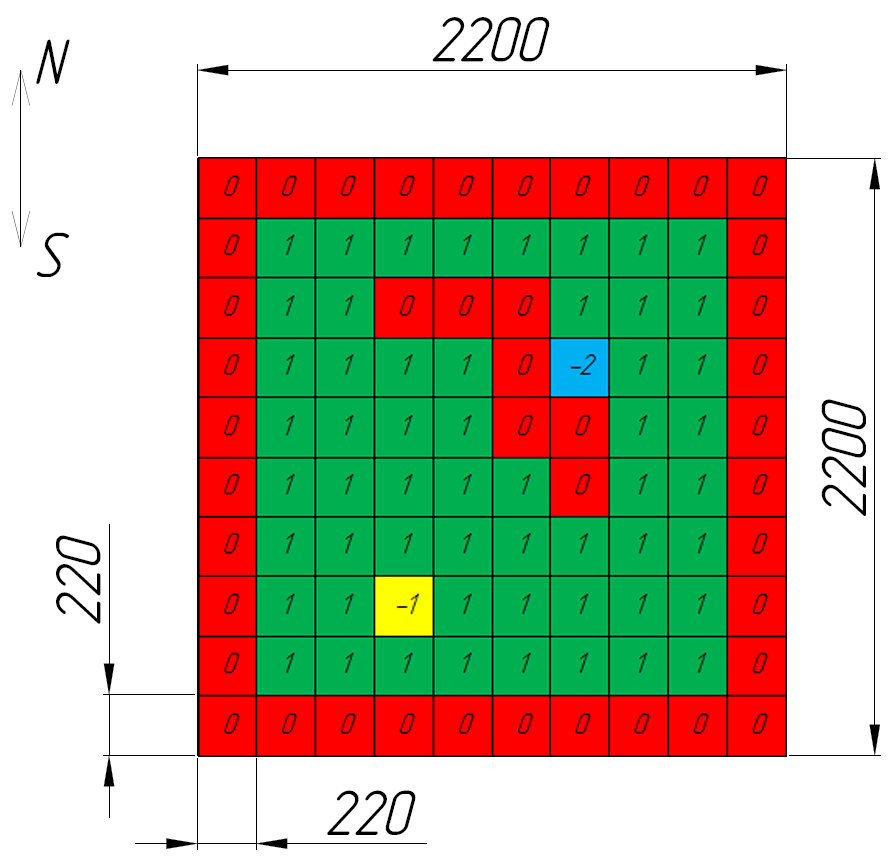


Рисунок 2.15. Дискретезированная карта местности.

Таким образом, изначально заданная карта местности с габаритами 2200 мм на 2200 мм превратилась в дискретизованную карту, с габаритами 10 на 10 клеток. Если в какой либо из частей клетки возникает непреодолимое препятствие, то вся клетка считается непроходимой для нашего робота.

Зададим координаты начала и конца пути и обозначим их на карте числами -1 и -2 соответственно (Рисунок 2.15). Так же необходимо задать изначальную ориентацию робота, то есть сторону света, в которую будет направлена передняя часть нашего робота, при установке его в начало пути. Зададим первоначальное положение обращенным на север.

Во второй части алгоритма разрабатывается непосредственно распространение волны. Так как волна при распространении будет присваивать каждой проходимой ячейке свой порядковый номер, начиная с ноля, нам необходимо переназначить значения ячеек для последующей их обработки. Для этого нам необходимо выбрать числа, значение которых будет превышать максимально возможное количество «волн» нашего алгоритма. При любом возможном рельефе карты с размером 10 на 10 клеток, максимальное число волн не превысит количества 100 штук. Поэтому переназначим все непроходимые участки на значение 100, проходимые и конец пути на значение 150, а начало пути на значение 0, так как последующий алгоритм будет строить волну из начальной точки нашего маршрута. Далее, начиная с начальной точки, проверяем соседние ячейки на проходимость. Если соседняя ячейка проходима, ты мы присваиваем ей значение, равное порядковому номеру волны. И так повторяем, пока общая «волна» не проверит всю карту (рисунок 2.16.).

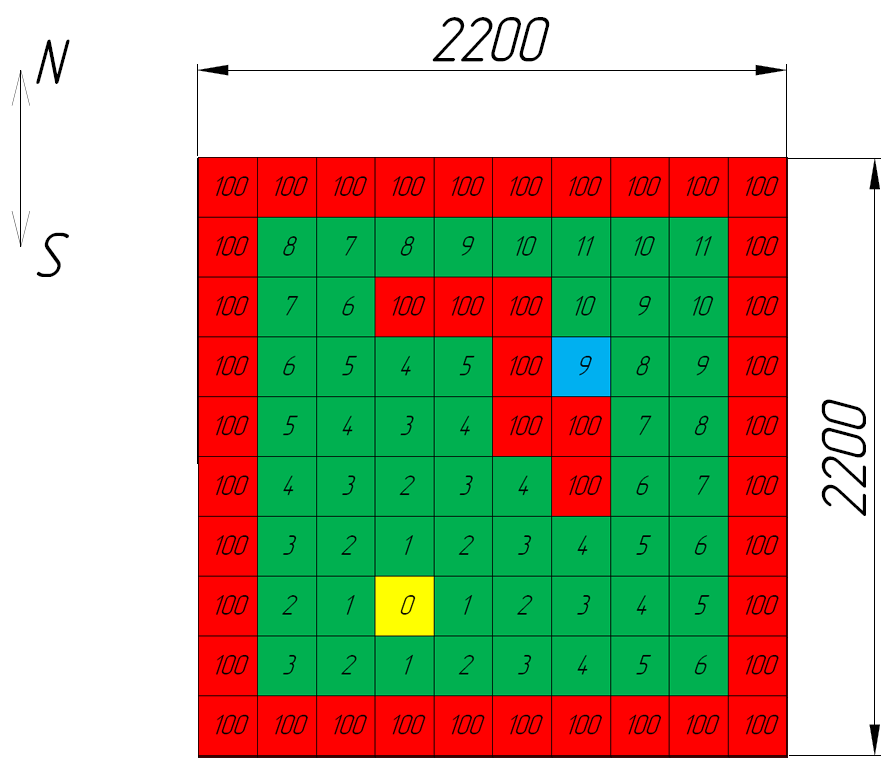


Рисунок 2.16. Распространение волны по карте от начальной точки маршрута.

В третьей, заключительной, части восстановления пути, по имеющимся сохраненным данным, мы восстанавливаем путь от начальной точки до конечной, по самому короткому маршруту. Для этого определяем значение ячейки конца пути, координаты которой были запомнены в первом шаге алгоритма. Значение этой ячейки будет равно количеству шагов по клеткам от начала маршрута до его конца. В нашем случае это значение равное девяти. Далее ищем ближайшую точку со значением, равным девять минус один. Запоминаем координаты этой точки и ищем следующую точку. Повторяем, пока не достигнем значения ноль, то есть конец пути (рисунок 2.17.).

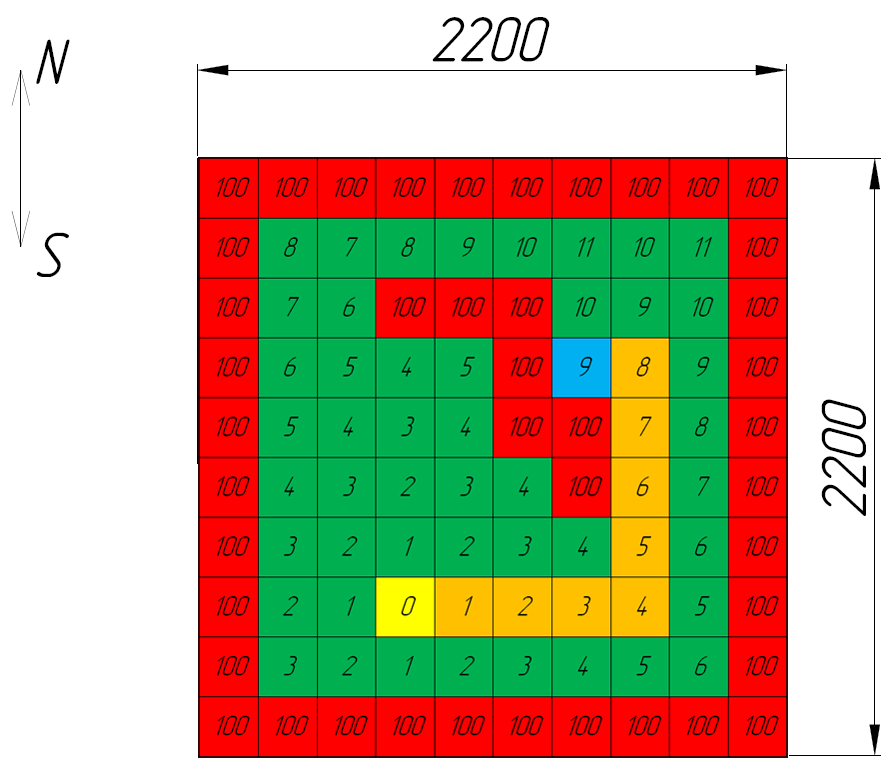


Рисунок 2.17. Построенный маршрут.

В итоге получаем массив координат ячеек, по которым должен двигаться робот, чтобы преодолеть путь из начальной точки в конечную по кратчайшему маршруту.

# Реализация движения робота по заданным координатам

В предыдущем пункте было подробно разобрано нахождение кратчайшего пути от начальной точки к конечной по заданной карте местности. Используя алгоритм Ли, получили массив координат ячеек, используя который, шестиногий робот должен передвигаться по заданному маршруту.

Так как робот движется только вперед, для поворота направо или налево ему необходимо совершить два последовательных действия: сначала повернуться на месте на заданный угол, а затем пройти вперед на заданное расстояние.

Зададим и опишем роботу три возможных действия:

* Движение вперед – робот перемещается вперед на расстояние 220мм.
* Движение налево – робот совершает поворот влево на 90° на месте. После завершения поворота робот перемещается вперед на расстояние 220мм.
* Движение направо – робот совершает поворот вправо на 90° на месте. После завершения поворота робот перемещается вперед на расстояние 220мм.

Таким образом, чтобы робот правильно отработал траекторию перемещения, ему необходимо задавать последовательные команды поворотов и движения вперед. Назначение каждой последующей команды будет зависеть от ориентации робота в пространстве (рисунок 2.18.) и от координат последующей точки перемещения.

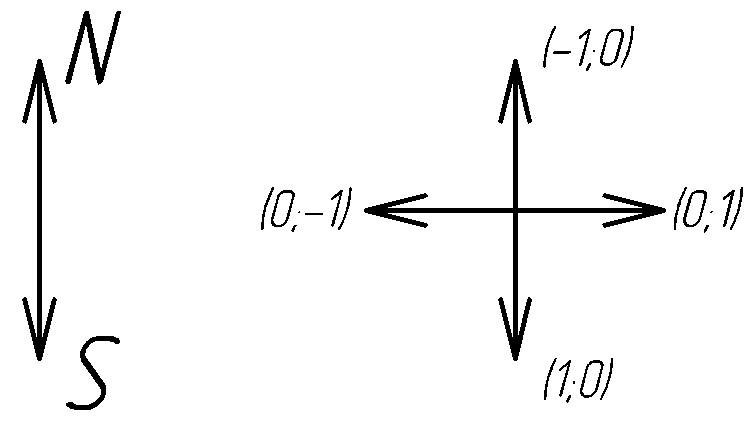


Рисунок 2.18. Схема ориентации робота

Если ориентация робота направлена в направлении следующей ячейки, то робот осуществляет движение вперед.

Если следующая ячейка находится справа (или слева) от робота, то робот осуществляет поворот вправо (или влево) на 90° на месте. Ориентация робота переназначается и робот осуществляет движение вперед.

Просчитывая все необходимые движения, программа сохраняет их в одномерный массив. В итоге, преобразовывая массив координат ячеек, по которым должен двигаться робот, получаем массив движений робота, каждое движение которого может иметь один из трех вариантов: движение вперед, поворот налево или поворот направо.

Подробный код представлен в приложении 2.

# Натурный эксперимент

При постановке задач эксперимента было выбрано несколько целей. Для проверки правильности решения ОКЗ и оценки точности перемещения:

* Прохождение робота по прямой на 1000мм;
* Поворот робота на 90° по часовой стрелке;
* Поворот робота на 90° против часовой стрелке;

Для проверки правильности прохождения робота по заданным координатам было выбрано четыре пути, отличающихся по сложности, т.е. количеству поворотов (рис. 2.19).

|  |  |
| --- | --- |
| D:\Бауманка\12 семестр\!!Диплом\Научно-исследовательская часть\map110.png  а) Ноль поворотов. | D:\Бауманка\12 семестр\!!Диплом\Научно-исследовательская часть\map130.png  б) Один поворот. |
| D:\Бауманка\12 семестр\!!Диплом\Научно-исследовательская часть\map100.png  б) Два поворота. | D:\Бауманка\12 семестр\!!Диплом\Научно-исследовательская часть\map120.png  г) Три поворота. |

Рисунок 2.19. Пути прохождения маршрута.

Так же было рассмотрено несколько вариантов питания с использованием четырех аккумуляторных батарей:

* Полностью заряженные с суммарным напряжением 5,5 В;
* Заряженные с суммарным напряжением 5,0 В;
* Разряжены и суммарное напряжение ниже номинального 4,8 В.

Для первой части натурного эксперимента был отмечен участок 1000 мм на ровной поверхности. Так же были отмечены углы +90° и -90°. После чего было проведено по 10 экспериментов, результаты которых отражены в таблице 2.7.

|  |  |  |  |
| --- | --- | --- | --- |
| №  Эксперимента | 1000 мм  по прямой | 90°  по часовой | 90°  против часовой |
| Результаты эксперимента | | |
| 1 | 1010 | 86 | 91 |
| 2 | 950 | 88 | 90 |
| 3 | 973 | 86 | 95 |
| 4 | 1018 | 87 | 94 |
| 5 | 964 | 86 | 89 |
| 6 | 1068 | 85 | 96 |
| 7 | 1050 | 89 | 92 |
| 8 | 920 | 86 | 95 |
| 9 | 988 | 88 | 91 |
| 10 | 990 | 85 | 93 |
| Среднее значение отклонения | 36,1 | 3,4 | 2,8 |
| Среднее значение отклонения в % | 3,6 | 3,8 | 3,1 |

Таблица 2.7. Результаты экспериментов.

Для второй части натурного эксперимента было размечено поле 2200 мм на 2200мм с размером ячейки 220 мм на 220 мм. Так же было проведено по 10 натурных экспериментов, результаты которых отражены в таблице 2.8.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  Эксперимента | 0 поворотов | 1 поворот | 2 поворота | 3 поворота |
| Результаты эксперимента | | | |
| 1 | ✔ | ✔ | ✔ | ✔ |
| 2 | ✔ | ✔ | ✔ | × |
| 3 | ✔ | ✔ | ✔ | ✔ |
| 4 | ✔ | ✔ | ✔ | ✔ |
| 5 | ✔ | ✔ | ✔ | × |
| 6 | ✔ | ✔ | ✔ | ✔ |
| 7 | ✔ | ✔ | × | ✔ |
| 8 | ✔ | ✔ | ✔ | × |
| 9 | ✔ | ✔ | ✔ | ✔ |
| 10 | ✔ | ✔ | ✔ |  |
| Сумма правильных отработок | 10 | 10 | 9 | 7 |
| Сумма неправильных отработок | 0 | 0 | 1 | 3 |

Таблица 2.8. Результаты экспериментов.

Условные обозначения:

✔ – робот правильно отработал траекторию

× – робот неправильно отработал траекторию

Для третьей части эксперимента было рассмотрено три случая:

* В случае полностью заряженных аккумуляторов робот полностью отрабатывает траекторию на высокой скорости.
* В случае напряжения 5,0 В траектория так же выполняется без ошибок, но время исполнения команд увеличивается и робот шагает медленнее.
* В случае разряженных аккумуляторов с напряжением ниже 4,8 В робот продолжает принимать команды, но выполняет их неправильно и не решает поставленной задачи.

Вывод

В результате проведенных экспериментов мы получили следующие данные.

В первой части натурного эксперимента получили следующие ошибки:

В случае движения по прямой – 3,6%, в случае движения по часовой стрелке – 3,8%, в случае движения против часовой стрелки – 3,1%.

Во второй части натурного эксперимента при прохождении простых траекторий с нулем и одним поворотами проблем с прохождением траектории не возникло. При прохождении двух поворотов один раз из десяти робот не отработал траекторию. При прохождении трех поворотов робот трижды неправильно отработал траекторию. Увеличение погрешности с увеличением количества поворотов связано с тем, что изначально робот имеет ошибки при прохождении прямых и при поворотах, из-за чего не может попасть в изначально заданную ячейку карты.

# Заключение

* Произведен энергетический расчет, выбран двигатель ДПТ TowerPro SG92R для каждого звена.
* Моделирование показало, что синтез не требуется.
* Решена обратная кинематическая задача трехзвенного манипулятора с 3-мя степенями свободы.
* После проверки с помощью графической среды, а так же с помощью шестиногого робота, управляя им с помощью программного кода, написанного на языке C++ было установлено, что ОКЗ решена верно.
* Реализовано нахождение кратчайшего маршрута по заданной карте местности.
* Реализовано движение робота в пространстве по заданным координатам.

# 7. Список использованной литературы

1. *Лесков А.Г.*, «МАНИПУЛЯЦИОННЫЕ РОБОТЫ»,Конспект лекций по курсу «ОСНОВЫ РОБОТОТЕХНИКИ» – М.: Издательство МГТУ, 2011;
2. *Калеватых И.А.*, методическое пособие «Создание виртуальной модели манипуляционного робота, решение и программирование обратной кинематической задачи» – М.: Издательство МГТУ, 2014;
3. *Павловский Е.В.*, *Панченко А.В.*, «Модели и алгоритм управления движением малого шестиногого робота», журнал «Мехатроника, автоматизация, управление» № 11, – М.: Изд-во «Новые Технологии», 2012.
4. *Стрелков С. П.*, «ОБЩИЙ КУРС ФИЗИКИ» – М.: Издательство «НАУКА», 1975. - 61 с.
5. *Улли Соммер*, «Программирование микроконтроллерных плат Arduino/Freeduino» – СПб: Изд-во «БХВ-Петербург», 2016.
6. *Фокин В. Г., Шаныгин С. В.* Обзор и перспективы развития мобильных шагающих робототехнических систем // Молодой ученый. — 2015. — №18. — С. 207-215.
7. *Прата С.* «Язык программирования С++. Лекции и упражнения» (5-е издание) - М.: Издательство Вильямс, 2013.
8. *Бесекерский В.А.*, *Попов Е.П.,* «Теория автоматического управления» - М.: Издательство Профессия, 2007.
9. *Иванов В.А., Медведев В.С., Чемоданов Б.К., Ющенко А.С.* «Математические основы теории автоматического управления» - М.: Издательство МГТУ, 1971.
10. *Кузьмина А.О.,* методическое пособие «Расчет электрической исполнительной системы манипулятора» – М.: Издательство МГТУ, 2006.

# 8. Приложения

Приложение 1. Алгоритм вычисления углов поворота сервоприводов исходя из решенной обратной кинематической задачи для всей ноги.

%перевод из радианов в градусы

rad2deg = pi/180;

%параметры матриц и векторов

AXIS = 3; %количество осей декартового пространства

LEGS = 6; %количество ног гексапода

JOINTS = 3; %количество шарниров на 1й ноге

%индексы

X = 1;

Y = 2;

Z = 3;

Q1 = 1;

Q2 = 2;

Q3 = 3;

LEG\_1 = 1;

LEG\_2 = 2;

LEG\_3 = 3;

LEG\_4 = 4;

LEG\_5 = 5;

LEG\_6 = 6;

x\_axis = [1 0 0];

y\_axis = [0 1 0];

z\_axis = [0 0 1];

%допустимая погрешность вычислений(+определение нуля)

EPS = 1E-06;

%геометричческие параметры

RADIUS = 100; %расстояние отн. BASE до СК0 (правильный шестиугольник)

COXA\_LENGHT = 26.0; %длина таза

FEMUR\_LENGHT = 50.0; %длина бедра

TIBIA\_LENGHT = 54.0; %длина голени

panel\_w = [0 0 5]; %толщина основной пластины

head\_w = [0 -5 0]; %толщина головы

L1 = COXA\_LENGHT \* x\_axis;

L2 = FEMUR\_LENGHT \* x\_axis;

L3 = TIBIA\_LENGHT \* x\_axis;

%расположение точек ног на основной панели

CK0\_1 = [-RADIUS\*cos(50\*rad2deg) RADIUS\*sin(50\*rad2deg) 0];

CK0\_2 = [-RADIUS 0 0];

CK0\_3 = [-RADIUS\*cos(50\*rad2deg) -RADIUS\*sin(50\*rad2deg) 0];

CK0\_4 = [RADIUS\*cos(50\*rad2deg) RADIUS\*sin(50\*rad2deg) 0];

CK0\_5 = [RADIUS 0 0];

CK0\_6 = [RADIUS\*cos(50\*rad2deg) -RADIUS\*sin(50\*rad2deg) 0];

CK0\_Head = [0 RADIUS\*sin(50\*rad2deg) 0];

%расположение точек косых ног

CK1\_1 = -1\*[-COXA\_LENGHT\*cos(50\*rad2deg)/2 COXA\_LENGHT\*sin(50\*rad2deg)/2 0];

CK2\_1 = -1\*[-FEMUR\_LENGHT\*cos(50\*rad2deg)/2 FEMUR\_LENGHT\*sin(50\*rad2deg)/2 0];

CK3\_1 = -1\*[-TIBIA\_LENGHT\*cos(50\*rad2deg)/2 TIBIA\_LENGHT\*sin(50\*rad2deg)/2 0];

CK1\_3 = -1\*[-COXA\_LENGHT\*cos(50\*rad2deg)/2 -COXA\_LENGHT\*sin(50\*rad2deg)/2 0];

CK2\_3 = -1\*[-FEMUR\_LENGHT\*cos(50\*rad2deg)/2 -FEMUR\_LENGHT\*sin(50\*rad2deg)/2 0];

CK3\_3 = -1\*[-TIBIA\_LENGHT\*cos(50\*rad2deg)/2 -TIBIA\_LENGHT\*sin(50\*rad2deg)/2 0];

CK1\_4 = [COXA\_LENGHT\*cos(50\*rad2deg)/2 COXA\_LENGHT\*sin(50\*rad2deg)/2 0];

CK2\_4 = [FEMUR\_LENGHT\*cos(50\*rad2deg)/2 FEMUR\_LENGHT\*sin(50\*rad2deg)/2 0];

CK3\_4 = [TIBIA\_LENGHT\*cos(50\*rad2deg)/2 TIBIA\_LENGHT\*sin(50\*rad2deg)/2 0];

CK1\_6 = [COXA\_LENGHT\*cos(50\*rad2deg)/2 -COXA\_LENGHT\*sin(50\*rad2deg)/2 0];

CK2\_6 = [FEMUR\_LENGHT\*cos(50\*rad2deg)/2 -FEMUR\_LENGHT\*sin(50\*rad2deg)/2 0];

CK3\_6 = [TIBIA\_LENGHT\*cos(50\*rad2deg)/2 -TIBIA\_LENGHT\*sin(50\*rad2deg)/2 0];

%ограничения углов в градусах

Q1\_MAX = 30;

Q1\_MIN = -30;

Q2\_MAX = 80;

Q2\_MIN = -80;

Q3\_MAX = 90;

Q3\_MIN = -90;

%параметры ходьбы

INIT\_HEIGHT = -100; %начальная высота платформы

%координаты векторов управления

C\_L\_X(1) = 70;

C\_L\_Y(1) = -85;

C\_L\_X(2) = 110;

C\_L\_Y(2) = 0;

C\_L\_X(3) = 70;

C\_L\_Y(3) = 85;

C\_L\_X(4) = -70;

C\_L\_Y(4) = -85;

C\_L\_X(5) = -110;

C\_L\_Y(5) = 0;

C\_L\_X(6) = -70;

C\_L\_Y(6) = 85;

%длины поремещений в декартовых и угловых координатах

DELTA\_Y = 15; %mm

DELTA\_Z = 15; %mm

DELTA\_PHI = 9; %degrees

% задержка на исполнение механической частью системы

DELAY\_MS = 250;

%далее сам расчет

%устанавливаем ограничения углов поворота шарниров

maxStateAngle(Q1) = Q1\_MAX;

maxStateAngle(Q2) = Q2\_MAX;

maxStateAngle(Q3) = Q3\_MAX;

minStateAngle(Q1) = Q1\_MIN;

minStateAngle(Q2) = Q2\_MIN;

minStateAngle(Q3) = Q3\_MIN;

%устанавливаем углы поворота СК0 относительно BASE

legBaseAngle(LEG\_1) = 130.0; %угол ню

legBaseAngle(LEG\_2) = 180.0;

legBaseAngle(LEG\_3) = 230.0;

legBaseAngle(LEG\_4) = 50.0;

legBaseAngle(LEG\_5) = 0.0;

legBaseAngle(LEG\_6) = 310.0;

%массив номеров шарниров

jointNumbers(LEG\_1,Q1) = 7;

jointNumbers(LEG\_1,Q2) = 6;

jointNumbers(LEG\_1,Q3) = 5;

jointNumbers(LEG\_2,Q1) = 11;

jointNumbers(LEG\_2,Q2) = 10;

jointNumbers(LEG\_2,Q3) = 9;

jointNumbers(LEG\_3,Q1) = 15;

jointNumbers(LEG\_3,Q2) = 14;

jointNumbers(LEG\_3,Q3) = 13;

jointNumbers(LEG\_4,Q1) = 24;

jointNumbers(LEG\_4,Q2) = 25;

jointNumbers(LEG\_4,Q3) = 26;

jointNumbers(LEG\_5,Q1) = 20;

jointNumbers(LEG\_5,Q2) = 21;

jointNumbers(LEG\_5,Q3) = 22;

jointNumbers(LEG\_6,Q1) = 16;

jointNumbers(LEG\_6,Q2) = 17;

jointNumbers(LEG\_6,Q3) = 18;

%обнуление матриц

controlVector = [0 0 0];

rotateBaseToZero = [0 0 0;0 0 0;0 0 0];

controlVectorSC0 = [0 0 0];

currentStateVector = [0 0 0];

currentState = [0 0 0];

calculatedStateVector = [0 0 0];

calculatedState = [0 0 0];

angleRotation = [0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0; 0 0 0];

%решение обратной кинематической задачи для кадой ноги и нахождение углов

%поворота для каждого двигателя

for i = 1:LEGS,

%заданный нами контрольный вектор

controlVector(1) = C\_L\_X(i);

controlVector(2) = C\_L\_Y(i);

controlVector(3) = INIT\_HEIGHT;

%матрица поворота BASE -> CK0

rotateBaseToZero(1,1) = cos((legBaseAngle(i) - 90.0)\*rad2deg); %вторая картинка внизу

rotateBaseToZero(1,2) = sin((legBaseAngle(i) - 90.0)\*rad2deg);

rotateBaseToZero(2,1) = -sin((legBaseAngle(i) - 90.0)\*rad2deg);

rotateBaseToZero(2,2) = cos((legBaseAngle(i) - 90.0)\*rad2deg);

rotateBaseToZero(3,3) = 1.0;

%умножаем матрицу поворота на контрольынй вектор

controlVectorSC0 = rotateBaseToZero \* controlVector';

controlVectorSC0 = controlVectorSC0';

%считаем текущий вектор состояния currentstatevector для нулевых углов

currentStateVector(X) = sin(currentState(Q1)\*rad2deg) \* (COXA\_LENGHT - TIBIA\_LENGHT \* sin(currentState(Q2)\*rad2deg) \* sin(currentState(Q3)\*rad2deg) + cos(currentState(Q2)\*rad2deg) \* (FEMUR\_LENGHT + TIBIA\_LENGHT \* cos(currentState(Q3)\*rad2deg)));

currentStateVector(Y) = cos(currentState(Q1)\*rad2deg) \* (COXA\_LENGHT - TIBIA\_LENGHT \* sin(currentState(Q2)\*rad2deg) \* sin(currentState(Q3)\*rad2deg) + cos(currentState(Q2)\*rad2deg) \* (FEMUR\_LENGHT + TIBIA\_LENGHT \* cos(currentState(Q3)\*rad2deg)));

currentStateVector(Z) = - TIBIA\_LENGHT \* cos(currentState(Q2)\*rad2deg) \* sin(currentState(Q3)\*rad2deg) - sin(currentState(Q2)\*rad2deg) \* (FEMUR\_LENGHT + TIBIA\_LENGHT \* cos(currentState(Q3)\*rad2deg));

calculatedStateVector(X) = currentStateVector(X);

calculatedStateVector(Y) = currentStateVector(Y);

calculatedStateVector(Z) = currentStateVector(Z);

%скалдываем две матрицы и получаем итоговые вектор положения конца всей конечности

calculatedStateVector = controlVectorSC0 + currentStateVector;

%расчет первого угла (таза)

calculatedState(Q1) = (atan2(calculatedStateVector(X),calculatedStateVector(Y)))/rad2deg;

%расчет второго и третьего угла (бедро и голень)

%расчет локальных координат конца ноги(центра второй окружности)

localX = sqrt(calculatedStateVector(X) \* calculatedStateVector(X) + calculatedStateVector(Y) \* calculatedStateVector(Y)) - COXA\_LENGHT;

localY = calculatedStateVector(Z);

%объявляем координаты точки пересечения

intersectionX = 0.0;

intersectionY = 0.0;

%находим коэффициенты уравнения прямой

A = -2.0 \* localX;

B = -2.0 \* localY;

C = localX \* localX + localY \* localY + FEMUR\_LENGHT \* FEMUR\_LENGHT - TIBIA\_LENGHT \* TIBIA\_LENGHT;

%расчет координат точки пересечения окружностей

x0 = -A\*C/(A \* A + B \* B);

y0 = -B\*C/(A \* A + B \* B);

aaa\_bigger = FEMUR\_LENGHT \* FEMUR\_LENGHT \* (A \* A + B \* B) + EPS;

aaa\_smaller = C\* C;

if (C \* C > FEMUR\_LENGHT \* FEMUR\_LENGHT \* (A \* A + B \* B) + EPS)

%нет пересечений

abcd = 0;

funcRes = false;

elseif (abs(C \* C - FEMUR\_LENGHT \* FEMUR\_LENGHT \* (A \* A + B \* B)) < EPS)

%1 пересечение(касательная)

intersectionX = x0;

intersectionY = y0;

else

%2 пересечения

D = FEMUR\_LENGHT \* FEMUR\_LENGHT - C \* C / (A \* A + B \* B);

mult = sqrt (D / (A \* A + B \* B));

Ax = x0 + B \* mult;

Bx = x0 - B \* mult;

Ay = y0 - A \* mult;

By = y0 + A \* mult;

%выбор точки пересечения

if (Ay < By)

intersectionX = Ax;

intersectionY = Ay;

else

intersectionX = Bx;

intersectionY = By;

end

end

%расчет углов Q2, Q3 исходя из координат точки в локальной СК

calculatedState(Q2) = (atan2(-1\*intersectionY, intersectionX))/rad2deg;

calculatedState(Q3) = (atan2(intersectionY - localY, localX - intersectionX))/rad2deg - calculatedState(Q2);

%задаем расчитанные углы

angleRotation(i,1) = calculatedState(Q1);

angleRotation(i,2) = calculatedState(Q2);

angleRotation(i,3) = calculatedState(Q3);

%корректировка левой стороны HEXY

if i < 4

angleRotation(i,1) = -angleRotation(i,1);

angleRotation(i,2) = -angleRotation(i,2);

angleRotation(i,3) = -angleRotation(i,3);

end

end

angleRotation = angleRotation + [-50 0 0; 0 0 0; 50 0 0; 50 0 0; 0 0 0; -50 0 0];

Приложение 2. Код программы поиска кратчайшего маршрута.

int N[10][10]; //размер нашей карты

struct trajectory

{

int X;

int Y;

};

typedef struct trajectory Path;

Path root[50]; //назначаем массив с координамтами точек от первой до последней по которым надо идти, чтобы пройти маршрут по траектории

int map[][10] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 1, 1, 1, 1, 1, 1, 1, 1, 0 },

{ 0, 1, 1, 0, 0, 0, 1, 1, 1, 0 },

{ 0, 1, 1, 1, 1, 0, 1, 1, 1, 0 },

{ 0, 1, 1, 1, 1, 0, 0, 1, 1, 0 },

{ 0, 1, 1, 1, 1, 1, 0, 1, 1, 0 },

{ 0, 1, 1, 1, 1, 1, 1, 1, 1, 0 },

{ 0, 1, 1, 1, 1, 1, 1, 1, 1, 0 },

{ 0, 1, 1, 1, 1, 1, 1, 1, 1, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }

};//карта местности, где "0" - пройти нельзя, "1" - пройти можно

map[7][3] = -1; //начало маршрута

map[3][6] = -2; //конец маршрута

for (int i = 0; i < 10; i++)

{

for (int j = 0; j < 10; j++)

{

N[i][j] = map[i][j];

if (map[i][j] == 0)

N[i][j] = 100; //назначаем непроходимым клеткам значение 100

if (map[i][j] == 1)

N[i][j] = 150; //назначем проходимым клеткам значение 150

if (map[i][j] == -2)

N[i][j] = 150; //назначаем концу пути 150

if (map[i][j] == -1)

N[i][j] = 0; //назначаем началу пути 0

}

}

for (int iter = 0; iter < 100; iter++) //строим волну, начиная из начала пути

{

for (int i = 0; i < 10; i++)

{

for (int j = 0; j < 10; j++)

{

if (N[i][j] == iter)

{

if ((i + 1)<10 && N[i + 1][j] != 100 && N[i + 1][j] > iter + 1)

N[i + 1][j] = iter + 1;

if ((i - 1) >= 0 && N[i - 1][j] != 100 && N[i - 1][j] > iter + 1)

N[i - 1][j] = iter + 1;

if ((j + 1)<10 && N[i][j + 1] != 100 && N[i][j + 1] > iter + 1)

N[i][j + 1] = iter + 1;

if ((j - 1) >= 0 && N[i][j - 1] != 100 && N[i][j - 1] > iter + 1)

N[i][j - 1] = iter + 1;

}

}

}

}

int znch0 = N[3][6]; //определяем колиество точек в маршруте

root[N[3][6]].X = 3; //координаты конца маршрута по Х

root[N[3][6]].Y = 6; //координаты конца маршрута по У

N[3][6] = -1; //задаем концу маршрута значение -1

for (int znch = (znch0 - 1); znch >= 0;) //ищем путь из конца в начало

{

for (int i = 0; i < 10; i++)

{

for (int j = 0; j < 10; j++)

{

if (N[i][j] == znch) //как только находим значение, на 1 меньшее чем предыдущее, тогда проверяем..

{

if (N[i + 1][j] == -1 || N[i - 1][j] == -1 || N[i][j + 1] == -1 | N[i][j - 1] == -1)//..чтобы клетка с этим значением была рядом по расположению с предыдущей клеткой..

{

root[N[i][j]].X = i; //..если это совпадает, и следующая клетка рядом с предыдщей и ее значение меньше на единицу, то записываем координаты этой новой точки

root[N[i][j]].Y = j;

N[i][j] = -1; //задаем клетке значение -1 и переходим к следующей точке и так, пока не придем в начало маршрута

znch = znch - 1;

}

}

}

}

}

int orient[2]; //массив с ориентацией робота в начальной точке

int coord[2]; //массив с координатами точки

int movements[100]; //массив со всеми движениями

orient[0] = -1; //ориентация робота..

orient[1] = 0; //..на свевер

int g,i;

for (i = 0, g = 0; i < znch0;) //определяем, какие команды подаем роботу (налево(1), прямо(2) или направо(3))

{

coord[0] = root[i + 1].X - root[i].X;

coord[1] = root[i + 1].Y - root[i].Y;

if (coord[0] == orient[0] && coord[1] == orient[1])

{

movements[g] = 2; //2 - идем прямо

i++;

g++;

}

else if (coord[1] == orient[0] && orient[1] == 0)

{

movements[g] = 1; //1 - поворачиваем налево при этих условиях и возвращаемся в начало алгоритма, чтобы пойти прямо

orient[0] = 0;

orient[1] = coord[1];

g++;

}

else if (coord[0] == orient[1] && orient[0] == 0)

{

movements[g] = 3; //3 - поворачиваем направо при этих условиях и возвращаемся в начало алгоритма, чтобы пойти прямо

orient[0] = coord[0];

orient[1] = 0;

g++;

}

else if (coord[1] == -orient[0] && orient[1] == 0)

{

movements[g] = 3; //3 - поворачиваем направо при этих условиях и возвращаемся в начало алгоритма, чтобы пойти прямо

orient[0] = 0;

orient[1] = coord[1];

g++;

}

else if (coord[0] == -orient[1] && orient[0] == 0)

{

movements[g] = 1; //1 - поворачиваем налево при этих условиях и возвращаемся в начало алгоритма, чтобы пойти прямо

orient[0] = coord[0];

orient[1] = 0;

g++;

}

}

for (int i = 0; i < g; i++)

{

printf("%2d ", movements[i]);

std::cout << std::endl;

} //выводим попорядку команды для робота, где 1 - поворот налево, 2 - идем прямо, 3 - поворот направо

std::cin.get();