УНИВЕРСИТЕТ ИННОПОЛИС

education

Материалы для подготовки ко второму отборочному этапу олимпиады Innopolis Open Robotics сезона 2023-24

17.01.2024

Оглавление

Для кого эта книга

В книге собраны материалы для подготовки ко второму отборочному этапу открытой олимпиады Innopolis Open Robotics 2023/24 учебного года. Книга полезна в первую очередь участникам олимпиады, а также школьникам и тренерам, желающим принять участие в олимпиаде в будущих сезонах.

Книга рассчитана на учащихся 7-11 классов общеобразовательных школ. Учителя и тренеры, занимающиеся робототехникой, могут пользоваться материалами и примерами из книги для улучшения своих учебных курсов, расширения базы практических примеров или составления оценочных и контрольных работ и мероприятий.

Как пользоваться этой книгой

Книга подготовлена в электронном виде с доступом по ссылке из «электронного облака». Это вызвано тем, что материалы в ней регулярно обновляются и дополняются. Часть глав и параграфов имеют пометку «*Материал в процессе подготовки*». Это означает, что по той же самой ссылке появится обновленная версия книги с этими материалами. Об обновлении книги, новых доступных или исправленных материалах, будет сообщаться в официальных каналах олимпиады.

Читатели могут распечатать книгу или отдельные ее главы, но должны быть готовы к тому, что в будущем материалы могут обновиться или дополниться. Обновление материалов может быть вызвано необходимостью уточнить те или иные понятия по просьбе участников олимпиады, попыткой устранить разночтения или недостаточно очевидные моменты.

При возникновении спорных ситуаций во время проведения этапов олимпиады рассматривается последняя версия книги, доступная по ссылке на момент возникновения спорной ситуации.

Авторы гарантируют, что необходимые для подготовки к определенному этапу олимпиады материалы будут добавляться в книгу не позднее, чем за неделю до окончания этого этапа.

Авторы стараются расположить главы в порядке усложнения материалов, но это не всегда возможно. В некоторых случаях могут упоминаться свойства и параметры, которые в дальнейших главах рассматриваются подробнее. Подобные случаи описываются в тексте.

В книге приняты следующие элементы форматирования:

Ключевые определения выделяются курсивом и горизонтальными линиями и отделяются от остального материала горизонтальными линиями.

Подобное выделение позволяет легко найти определения в общем тексте.

Формулы выносятся на отдельные строки и имеют выравнивание по центру. Пример формулы:

$$a^2 + b^2 = c^2 \quad (0)$$

Ключевые формулы имеют нумерацию, обозначенную справа в круглых скобках. Текст книги может ссылать на эти формулы в последующих главах и параграфах. Например, в (0) представлена теорема Пифагора и согласно (0) квадрат гипотенузы равен сумме квадратов катетов.

Изображения, схемы и иные графические элементы выносятся на отдельную строку и имеют как нумерацию, так и описание. Часть изображений может иметь несколько элементов, каждый из которых обозначен и раскрывается в описании или тексте книги.

а) б)

Рисунок 0.1. Смайлики

Обратите внимание на нумерацию: первая часть номера обозначает главу, вторая – сквозной номер рисунка внутри главы. В тексте изображения упоминаются следующим образом: выше представлены веселый смайлик (рисунок 0.1 а) и грустный смайлик (рисунок 0.1 б).

Встречающиеся в тексте номера в квадратных скобках обозначают номера источников, представленных в конце книги. Например, упоминание в тексте [1] означает ссылку на источник под номером 1, а упоминание в тексте [2, стр.3] означает ссылку на страницу 3 источника под номером 2.

В главе 1 встречаются физические величины, взятые в квадратные скобки, но в ней так обозначаются их единицы измерения.

В некоторых главах встречается псевдокод управляющих программ, который выделен `ОСОБЫМ ШРИФТОМ`. Этот псевдокод описывает `команды` и `действия` словами русского языка, а не какого-либо конкретного языка программирования. Авторы стараются выносить псевдокод на отдельные строки и визуально отделять его отступами. При необходимости псевдокод может иметь нумерацию строк.

1. Основные физические понятия

Второй этап олимпиады Innopolis Open Robotics проверяет знания участников в области робототехники. Большинство задач направлены не только на подготовку каких-либо абстрактных алгоритмов, но и вычисления скоростей и пройденного пути, реализацию управляющих алгоритмов, распознавание объектов на кадре с камеры. Так как робототехника является разделом техники, то она тесно связана с физикой, ее поведение описывается физическими законами и моделями. Для эффективной подготовки к этому этапу олимпиады придется вспомнить некоторые понятия из школьного курса математики и физики, а также познакомиться с некоторыми темами, выходящими за рамки школьного курса.

1.1. Единицы измерения

Базовым понятием, которое необходимо постоянно помнить при вычислениях любых физических величин, является «единица измерения».

Единица измерения – величины, которые обозначают меру расстояния, массы, времени, скорости, площади и пр.

Без введения понятия «единицы измерения» невозможно будет оценить, насколько одна величина больше другой. Например, сравнивая скорости вращения колес робота можно понять, каким образом он будет перемещаться.

В задачах по робототехнике для школьников чаще всего приходится сталкиваться с единицами измерения длин, углов, времени, скоростей, массы и веса, сил и крутящих моментов. Поэтому рекомендуется по мере повторения и изучения дальнейших тем останавливаться и выбирать единообразные единицы измерения для физических величин.

Чаще всего в физике используется международная система единиц физических величин (СИ) и при вычислениях рекомендуется переводить все исходные значения в ее единицы. Но на практике бывают случаи получения исходных данных в других единицах, которые проще и удобнее вычислять. Например, углы могут измеряться как в градусах, так и в радианах, при этом большинство библиотек для стандартных языков программирования работает в радианах, но школьникам удобнее воспринимать углы в градусах. Или расстояния согласно СИ необходимо измерять в метрах, но их измерение в миллиметрах может позволить производить вычисления с целыми числами (которые более эффективны на медленных вычислителях, таких как контроллеры роботов). В общем случае следует соблюдать единообразие единиц измерения для одинаковых физических величин и вводить поправочные коэффициенты. Если же использовать величины в СИ, то поправочные коэффициенты упрощаются до единицы.

В дальнейших вычисления крайне важно следить за единицами измерения и научиться переводить значения между ними. Участникам олимпиады будут предложены соответствующие задачи.

Во всех задачах второго отборочного этапа олимпиады Innopolis Open Robotics единицы измерения входных и выходных значений строго определены и указаны.

1.2.Материальная точка

Еще одно базовое понятие, которое нам предстоит вспомнить, это «материальная точка». Чтобы упростить описание движения робот или отдельные его части будут рассматриваться как материальные точки.

Материальная точка – это тело, размерами которого в данных условиях можно пренебречь.

Важно помнить, что материальная точка не имеет размеров (так как ими пренебрегают), но имеет массу. Это значит, что при описании движения робота или, например, его колеса, можно рассматривать их как простые точки заданной массы.

Согласно определению размерами материальной точки можно пренебречь. В задачах робототехники они используются скорее не потому, что размерами можно пренебречь, а для облегчения описания движения всего робота через описание его геометрического центра или центра масс.

В общем случае материальная точка может совершать не только перемещение, но и вращение. Так как с помощью материальных точек могут быть описаны центры масс или геометрические центры объектов, чаще всего принимаемых за начало локальных систем координат, то и полное описание движения объекта будет совпадать с движением материальной точки.

1.3. Расстояние, траектория, перемещение, путь

На этом этапе проще всего вспомнить определения из курса физики, так как сами понятия не так сложны.

Для описания последующих терминов применяется понятие расстояния и единицы измерения расстояния.

Расстояние – степень удалённости объектов друг от друга.

В технике под расстояние чаще всего понимают длину отрезка, соединяющего два объекта. То есть определение похоже на определение «перемещения» (см. далее). На этом этапе важно понимать, что единицей расстояния являются метры (в СИ), сантиметры, миллиметры или иные единицы измерения длины. В олимпиаде Innopolis Open Robotics не используются дюймовые или футовые единицы измерения длины и расстояния, только метры или кратные им единицы.

Траектория – линия, вдоль которой движется материальная точка.

Траектория может быть сложной, закольцованной и содержащей движения в разных направлениях.

Перемещение – вектор, проведенный из начальной в конечную точку движения.

В СИ единица измерения модуля вектора перемещения – метр: [S] = м. Для общего случая вектор может быть описан в трехмерном пространстве. Из этого следует, что надо понимать или повторить вычисление длины векторов в пространстве.

Путь – длина траектории.

В СИ единица измерения путь – метр: [L] = м.

1.4. Сила, крутящий момент

Материал в процессе подготовки.

1.5. Скорости линейные и угловые

Материал в процессе подготовки.

1.6. Ускорения линейные и угловые

Материал в процессе подготовки.

2. Привод мобильного робота

Перед тем как рассматривать различные платформы мобильных роботов следует подробнее изучить отдельный привод робота.

Привод робота – элемент, обеспечивающий движение робота в пространстве или его составных частей друг относительно друга, преобразуя различную энергию в механическое движение, под управлением программы.

В общем случае приводы робота могут быть электрическими, пневматическими и гидравлическими. В задачах олимпиады Innopolis Open Robotics сезона 2023/24 применяются и рассматриваются только электрические приводы.

2.1. Точные проезды колеса на заданное расстояние

Часто во время подготовки к соревнованиям или во время решения иных практических задач требуется проехать заранее известное расстояние – оно обусловлено конфигурацией пространства, в котором перемещается робот, это могут быть проезды по известной комнате и т.д. Подобные проезды лучше всего делать по оборотам или градусам поворота колеса, а не по времени. В таком случае проезды будут одинаковыми даже на разных моторах, при разном заряде батареи и прочих изменяющихся условиях. Единственное ограничение состоит в том, что расстояние измеряется в сантиметрах или дюймах, а проезд устанавливается в градусах или оборотах колеса. Далее описано, как связаны эти величины.

Рассмотрим колесо робота сбоку (рисунок 2.1). Его граница является окружностью с заданным радиусом / диаметром. Когда колесо остановлено только одна его точка (A) касается поверхности. При начале движения еще одна точка (B) касается поверхности, еще одна (C) и так далее. В результате, за один оборот колеса все точки окружности коснутся поверхности.

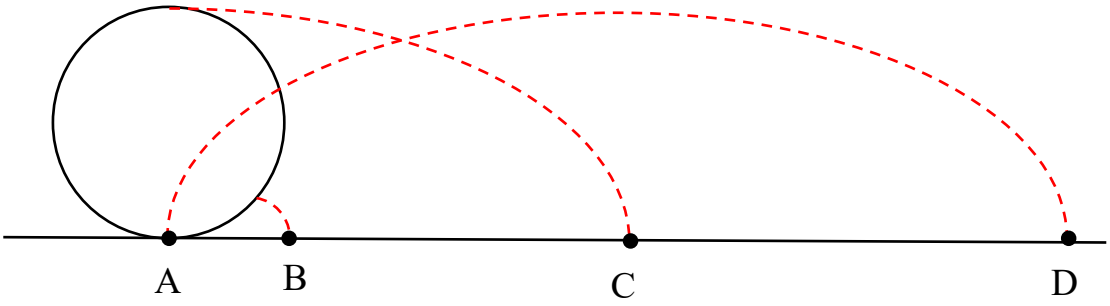


Рисунок 2.1. Перемещение точек колеса при его вращении

Точка A колеса, с которой начиналось движение, проделает полный оборот и «приземлится» на поверхность в точку D. То есть, длина отрезка AD на поверхности равна сумме всех точек колеса. То есть, длина отрезка AD равна длине окружности колеса. Для вычисления длины окружности используется следующая формула:

$$l = \pi d = 2\pi r \quad (1)$$

где:

d – диаметр окружности (например, 56 мм у колес Lego EV3 и Lego SPIKE Prime). Это расстояние между противоположными точками окружности.

r – радиус окружности. Это расстояние от ее центра до любой ее точки. Равен половине диаметра (28 мм для примера выше).

π – число Пи, которое равно примерно 3.141592. Это число всегда такое, для всех колес и всех окружностей.

Подставим значения из указанного примера для колес Lego EV3 и Lego SPIKE Prime в формулу:

$$l = \pi d = 3.141592 * 56\text{мм} = 175{,}929152 \text{ мм}.$$

Таким образом вычисления показали, что за один оборот колеса робот проезжает 175.9291 мм. Теперь осталось только узнать, сколько оборотов колеса в нужном для преодоления роботом расстоянии. Для простоты предположим, что все колеса робота вращаются с одинаковой скоростью и совершают поворот на одинаковый угол, то есть каждое из них и робот в целом преодолевают одинаковой расстояние.

Важно отметить, что диаметр колеса был указан в миллиметрах, следовательно и длина одного его оборота тоже вычислена в миллиметрах. Если бы диаметр / радиус были заданы в метрах, то и расстояние получилось бы в метрах.

Например, роботу требуется проехать 1 метр, то есть 1000 миллиметров. Требуется вычислить, сколько оборотов колеса помещается в один метр, то есть разделить один метр на длину проезда за один оборот:

$$n = \frac{S}{l} \quad (2)$$

$$n = \frac{1000 \text{ мм}}{175{,}9291 \text{ мм}} = 5{,}684106 \text{ оборота}$$

За 5,6841 оборота колеса робот проезжает примерно 1 метр. Чем точнее входные значения в формулах (чем больше знаков после запятой), тем точнее получатся вычисления. Многие языки программирования позволяют подставлять в формулы числа с большим количеством знаков после запятой. Если подставить 5.68 оборота, то ошибка перемещения робота будет примерно 1 мм. Если подставим 5.684106, то ошибка перемещения будет меньше 0.001 мм (меньше 1 микрометра). В большинстве случаев, даже подставляя значения в СИ, достаточно указывать не более четырех знаков после запятой.

Перевести обороты в градусы поворота колеса достаточно просто, достаточно умножить их на 360:

$$n_{\text{град}} = 360 * n_{\text{обор}} \quad (3)$$

$$n_{\text{град}} = 360 * 5{,}6841 \text{ оборота} = 2046{,}27816 \text{ градусов.}$$

Необходимое для проезда расстояние можно задавать как в оборотах, так и в градусах.

На практике добиться полностью одинакового (синхронного) вращения всех колес робота достаточно сложно, в результате чего перемещение робота может отличаться от пути или траектории движения каждого колеса. Часто на практике необходимо организовать движение робота не по прямой линии, а по дуге, задавая разные скорости вращения колес. Более того, некоторые платформы роботов предполагают вращение колес с разной скоростью даже для движения по прямой линии (например, платформа с омни-колесами). Способы синхронизации колес и зависимости перемещения робота от перемещения колес будут рассмотрены в следующих главах. На этом этапе достаточно понимать, как рассчитывать путь, преодоленный одним колесом в зависимости от угла его поворота.

2.2. Типы приводов

Обратившись к определению *привода* из предыдущей главы, можно отметить, что в нем не указан конкретный тип механического движения. Приводы могут совершать вращательное движение (крутить какой-либо выходной вал с колесом) или линейное движение (перемещать какую-либо каретку или выдвигать шток). Линейное движение линейных приводов ограничено механическими параметрами – переместить каретку за пределы ее направляющих невозможно. Движение вращательного привода может быть как ограниченным, так и повторяющимся без ограничений. Например, существуют сервоприводы вращательного движения, которые могут поворачивать выходной вал в диапазоне 90 градусов, 180 градусов или даже на несколько оборотов (больше 360 градусов).

Линейные приводы часто реализуются с помощью преобразования вращательного движения в поступательное различными механическими передачами, чаще всего зубчато-реечной передачей или шарико-винтовой парой. То есть изначально энергия преобразуется во вращательное механическое движение, а затем уже оно в механическое поступательное движение. Это объясняется массовым производством моторов, совершающих вращательное движение – экономически выгоднее использовать имеющиеся в наличии моторы и дорабатывать механику, чем изобретать и налаживать производство каких-то моторов, преобразующих энергию сразу в поступательное движение.

На примере линейных приводов видно, что под *приводом* понимается не только сам моторов, но и дополнительные элементы, обеспечивающие его работу в нужном режиме. Чаще всего это мотор для преобразования энергии в механическое движение, механические передачи для преобразования и получения движения заданных параметров, схема управления мотором и датчики, обеспечивающие ее работу. В

мобильной робототехнике для синхронизации двух и более моторов может применяться одна схема управления на несколько приводов.

Кажущаяся сложность привода опять же объясняется экономической целесообразностью: проще наладить производство нескольких электромоторов со строго заданными параметрами и дополнить их различными передачами и схемами управления для получения очень широкого спектра скоростей и крутящих моментов на выходном валу. Для наглядности рассмотрим схему привода с двигателем постоянного тока (ДПТ), представленную на рисунке 3.1. Многие производители используют при производстве моторы всего двух или трех серий (условно «легкой», «средней» и «тяжелой»), но комбинируя их с разными редукторами и энкодерами получают очень широкий номенклатурный ряд: приводы с разными скоростями вращения и крутящими моментами.

Основными характеристиками линейных приводов являются максимальные длина, линейная скорость и сила перемещения каретки или штока. При выборе также необходимо учитывать габариты привода и посадочные размеры (крепления самого привода к роботу и перемещаемого объекта к каретке).

Основными характеристиками вращательных приводов являются максимальная угловая скорость и крутящий момент. При выборе, как и у линейных приводов, следует учитывать габариты и посадочные размеры (в том числе диаметр выходного вала).

2.3. Редуктор

Материал в процессе подготовки.

2.4. Энкодер, тахометр, датчик тока

Материал в процессе подготовки.

3. Алгоритмы управления приводами робота

Материал в процессе подготовки.

3.1. Регуляторы

Одним из самых широко распространенных и применимых в робототехнике алгоритмов является регулятор.

В робототехнике и автоматизации под регулятором понимают алгоритм, удерживающий систему (или робота) в нужном состоянии, чтобы датчики видели нужные показания.

Чуть реже под регулятором понимают устройство, реализующее подобный алгоритм. Гораздо чаще регулятор прописывают как программу или часть программы управляющего контроллера.

Остановимся в этом месте и еще раз вчитаемся в предыдущий абзац, о чем в нем говорится? Что регулятор — это алгоритм, программа. Если робот собирается из наборов ТРИК или Lego, то у него не будет какой-то особенной "детали-регулятора", его придется описывать в программе.

Второй важный момент - регулятор держит и направляет системы так, чтобы датчики видели заданное значение. То есть, регулятор всегда настраивается на показания одного или нескольких датчиков. Они позволяют определить, находится ли система в заданном состоянии. Без сенсоров регулятор не имеет смысла - непонятно будет, что сейчас происходит с роботом и как корректировать его поведение. Например, есть коптер (робот-вертолет) и требуется, чтобы он летел на высоте 100 метров. Датчик робота видит высоту 95 метров, следовательно ему нужно немного взлететь. Если датчик видит 110 метров, то коптеру нужно немного снизиться. Программа, которая будет автоматически управлять взлетом и снижением робота и будет регулятором. Если бы у робота не было датчика, то он не понимал бы, на какой он высоте и надо ли ему взлетать или снижаться.

Подобных примеров можно придумать очень много, почти в каждом соревновании или производственной сфере:

- ехать строго на север по датчику-компасу и подруливать, если робот отклонился;
- удерживать руку робота на нужной высоте, даже если нагрузка на нее увеличивается;
- ехать по границе черной линии на белом фоне и подруливать, если робот отклонился сильно к черному или белому;
- удерживать одну и ту же скорость при движении робота на подъеме и спуске с горки (это пример автомобиля в режиме круиз-контроля);
- держаться подводным роботом на заданной глубине;
- и т.д.

Алгоритм "удерживания" нужного состояния можно реализовывать разными способами. Принято выделять четыре. Самый простой - релейный регулятор. Он подобен устройству реле. Это такой переключатель, который может иметь всего два состояния (выдавать два вида сигналов). Чаще всего это "вкл/выкл".

Релейный регулятор с точки зрения программы описывается буквально одним оператором:

```
ЕСЛИ (температура < пороговое_значение) ТО
        тэн.вкл();
ИНАЧЕ
        тэн.выкл();
КОНЕЦ ЕСЛИ
```

Если температура в печи ниже заданной, то нагревательный тэн включается, иначе - выключается.

Подобный алгоритм подходит для систем с медленной реакцией на управляющие воздействия. Та же печь медленно нагревается и долго остывает, для нее подобный алгоритм подойдет нормально. Но представим, что требуется проехать роботом заданное расстояние. Опишем программу, которая крутит моторы вперед, если «датчик пройденного расстояния» видит меньше порогового значения и назад, если робот уехал слишком далеко (сенсор видит больше порогового значения):

```
ЦИКЛ (условие)
  ЕСЛИ (датчик < пороговое_значение) ТО
        моторы.вперёд();
  ИНАЧЕ
        моторы.назад();
  КОНЕЦ ЕСЛИ
КОНЕЦ ЦИКЛА
```

С такой программой робот будет дергаться возле нужного значения проезда. Он едет вперед, останавливает моторы и проезжает по инерции немного дальше. Слишком далеко с точки зрения алгоритма. И включается движение назад, после остановки которого робот немного проезжает по инерции. И все повторяется до бесконечности. Еще хуже будет выглядеть поведение коптера, который с подобным алгоритмом то включает винты на максимум и начинает взлетать, то выключает их полностью и начинает падать. Для систем с быстрой реакцией на управляющие воздействия нужен более адекватный алгоритм, плавно регулирующий поведение.

Для этого как раз подойдут три остальных способа регулирования:

- пропорциональная составляющая (П);
- интегральная составляющая (И);
- дифференциальная составляющая (Д).

Комбинируя их, можно добиться разного поведения регулятора. Регуляторы принято называть по имеющимся в них частям: ПИД-регулятор имеет все три компоненты, П-регулятор содержит только пропорциональную, ПД-регулятор содержит пропорциональную и дифференциальную части и т.д.

Обычно П-часть делает основную часть работы регулятора, а И- и Д-части улучшают его функционирование. Поэтому обычно бывают П-, ПИ-, ПД-регуляторы, и редко встречаются И- или ИД-регуляторы.

Перед тем, как двигаться дальше и рассматривать принципы функционирования каждой компоненты, необходимо ввести несколько терминов, которые используются при работе с регуляторами.

Уставка - желаемое состояние системы (или робота), то есть, желаемые показания датчиков.

Чаще всего ее задает сам разработчик-программист, ее можно запоминать в какой-то момент времени (когда система в нужном нам состоянии) и далее удерживать. Например, робот-футболист запоминает азимут на ворота противника и движется в их сторону по датчику-компасу.

Ошибка - разница между показаниями датчика и уставкой.

Это не ошибка в программе, а ошибка в поведении системы. Она должна быть в состоянии уставки, а датчик видит иное. Коптер должен лететь на высоте 100 метров, а он летит на высоте 110 метров, следовательно, ошибка равна 10. Обратите внимание, что если коптер летит на высоте 85 метров, то ошибка будет равна -15! Знак ошибки явно указывает, в какую сторону отклонение.

Управляющее воздействие - сигнал от регулятора, который должен влиять на систему, возвращая ее в состояние уставки.

Оно складывается из всех частей регулятора: пропорциональной, интегральной и дифференциальной. Степень влияния той или иной составляющей задается с помощью коэффициентов kp, ki и kd, о которых подробнее будет рассказано далее.

Учитывая эти термины, давайте опишем и рассмотрим принципы работы всех трех составляющих регулятора.

Принцип работы пропорциональной части регулятора: чем дальше система находится от состояния уставки, тем сильнее она к ней стремится.

Вернемся к примеру с коптером – при ошибке равной -15 (робот летит на высоте 85 метров) он сильно взлетает. Чем ближе он к требуемой высоте, тем слабее/медленнее он взлетает. Если рассмотреть модель подводной лодки или самолета, то на нужной глубине/высоте ее рули устанавливаются вертикально, и чем дальше модель ушла от требуемой глубины/высоты, тем сильнее ее рули отклоняются (рисунок 3.1).

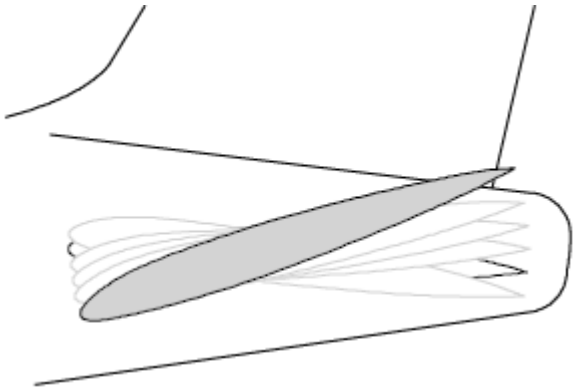


Рисунок 3.1. Схема поворота рулей высоты самолета.

Теперь запишем программный алгоритм, реализующий этот принцип. Для простоты и наглядности на примере движения робота по одному датчику вдоль линии:

```
ЦИКЛ (условие)
        err = S – ust;
        P = err * kp;
        U = P;
        Vleft = Vbase + U;
        Vright = Vbase – U;
КОНЕЦ ЦИКЛА
```

где:

- err - *ошибка*;
- S - показания датчика;
- ust - *уставка* (требуемые показания датчика);
- P - пропорциональная составляющая;
- kp - коэффициент пропорциональности;
- U - *управляющее воздействие*;
- Vleft, Vright - скорости левого и правого моторов робота;
- Vbase - базовая скорость, которая должна быть у моторов, если робот в состоянии уставки.

Данные команды (математические операции) необходимо делать в цикле, вычисляя ошибку и изменяя управляющее воздействие на каждом его шаге.

Рассмотрим, как это работает. Уставку задает сам программист. Для робота, движущегося вдоль линии, за уставку принимаются показания датчика освещенности на ее границе. Например, на белом датчик видит 10, на черном видит 80, на границе примерно среднее арифметическое между этими числами:

$$ust = \frac{10 + 80}{2} = 45$$

То есть, уставку следует сделать равной 45. Коэффициент пропорциональности — это просто число. Он может быть вычислен исходя из математической модели робота, его физических параметров, электромеханических свойств моторов и т.д. В

проектах школьников его можно просто подобрать. Для наглядного примера установим этот коэффициент равным 2, базовую скорость равной 50.

Предположим, что в момент запуска датчик видит как раз 45, получим следующие вычисления на первом шаге цикла:

```
 0  = 45 -  45
err=S - ust;
0 =  0  *  2
P=err*kp;
0  =  0
U=P;
  50  =   50   + 0
Vleft=Vbase+U;
  50   =   50   - 0
Vright=Vbase-U;
```

Робот в состоянии уставки, ошибка получается равной нулю, умножаем этот нуль на коэффициент пропорциональности и получаем тот же нуль в пропорциональной части и управляющем воздействии. Оба мотора крутятся с базовыми скоростями, робот едет прямо.

В какой-то момент линия начнет поворачивать, а робот поедет прямо. Предположим, что линия ушла вправо и датчик робота стал видеть 50:

```
 5  = 50 -  45
err=S - ust;
10 =  5  *  2
P=err*kp;
10 = 10
U=P;
  60  =   50   + 10
Vleft=Vbase+U;
  40   =   50   - 10
Vright=Vbase-U;
```

Получаем положительную ошибку, равную 5. Она умножается на коэффициент пропорциональности и в пропорциональной части (равно как и в управляющем воздействии) получаем 10. Скорость левого мотора увеличивается на 10, правого - замедляется на 10. В результате, так как левый мотор крутит быстрее, робот начинает поворачивать вправо, к линии.

Предположим, что робот проскочил границу линии и датчик теперь видит 40:

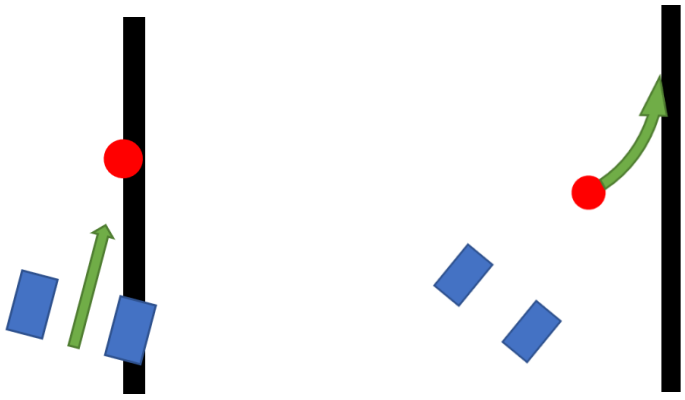
```
 -5   = 40 -  45
err=S - ust;
-10 = -5 *  2
P=err*kp;
-10 = -10
U=P;
   40   =     50    + (-10)
Vleft=Vbase+U;
   60     =    50     - (-10)
Vright=Vbase-U;
```

Получим ошибку той же величины, но с другим знаком. В результате скорости колес поменяются, и робот начнёт подруливать влево.

И чем больше будет значение (модуль) ошибки, тем больше будет получаться пропорциональная часть, тем сильнее она будет тянуть робота к линии.

Рассмотрим коэффициент пропорциональности. Как уже было сказано, это просто число и его можно просто подобрать. Число не должно быть отрицательным (иначе робот станет не возвращаться к уставке, а убегать от нее), оно может быть не целым. Подбор kp можно сделать буквально "на глаз" – если система слишком слабо реагирует на изменение состояния, то kp слишком мал и его надо поднять. Если система начинает резко реагировать на изменения и дергается, то kp слишком большой. Начинать можно с числа 1 или 2 и подбирать коэффициент несколькими итерациями.

Даже при самом идеально подобранном коэффициенте kp система с П-регулятором неизбежно будет "раскачиваться" вокруг состояния уставки. На примере робота, движущегося вдоль линии, наглядно видна причина: если робот немного отклонился, то, возвращаясь к границе, находит состояние уставки. В этот момент он едет прямо, но куда это относительно линии?



а. б.

Рисунок 3.2. а. – движение робота вдоль линии на П-регуляторе.
б. – движение робота вдоль линии на ПД-регуляторе

На рисунке датчик находится ровно на границе черной линии, колеса крутятся вперед с одинаковой скоростью, робот неизбежно проскочит состояние уставки.

Успокоить эти колебания, выводя робота плавно на линию, позволяет дифференциальная составляющая.

Принцип работы дифференциальной части регулятора: чем быстрее система уходит от состояния уставки, тем сильнее она к ней стремится.

Эта компонента следит уже не за величиной ошибки, а за тем, как быстро ошибка изменяется. Дифференциальная составляющая позволяет роботу сильнее подруливать на резких поворотах. Для математического или программного описания Д-части надо помнить значение ошибки с предыдущего шага:

```
ЦИКЛ (условие)
        err = S – ust;
        P = err * kp;
        D = kd*(err – errold);
        U = P + D;
        Vleft = Vbase + U;
        Vright = Vbase – U;
        errold = err;
        ПАУЗА(0.005 с);
КОНЕЦ ЦИКЛА
```

Здесь добавились:

- D - дифференциальная составляющая;
- kd - коэффициент дифференциальности;
- errold - ошибка на предыдущем шаге.

Подобрать коэффициент дифференциальности "на глаз" уже сложнее, для этого придется делать специальные вычисления. Оставим их за рамками этой книги, так как они требуют знания математических сущностей, изучаемых в курсе высшей математики. В дальнейших наглядных примерах будем считать kd равным единице для простоты вычислений.

Рассмотрим вычисления на трех шагах: начало движения и ошибка равна 0, далее резкий поворот и ошибка равна 15, затем робот начинает приближаться к линии, и ошибка уменьшается до 10:

```
 0  =  45  -  45
err = S - ust;
 0  =  0  *  2
P = err * kp;
 0  =  1  *  ( 0  –   0  )
D = kd * (err – errold);
 0  = 0  +  0
U = P + D;
  50   =   50    +  0
Vleft = Vbase + U;
  50    =   50   -  0
Vright = Vbase - U;
  0    =  0
errold = err;
```

```
 15  =  60 -  45
err = S - ust;
30 =  15  *  2
P = err * kp;
15  =  1  *  ( 15 –   0  )
D = kd * (err – errold);
45  = 30  +  15
U = P + D;
  95   =   50    +  45
Vleft = Vbase + U;
  5    =   50   -  45
Vright = Vbase - U;
  15    =  15
errold = err;
```

```
 10  =  55 -  45
err = S - ust;
20 =  10  *  2
P = err * kp;
-5  =  1  *  ( 10 –   15  )
D = kd * (err – errold);
15  = 20  +  (-5)
U = P + D;
  65   =   50    +  15
Vleft = Vbase + U;
  35   =   50   -  15
Vright = Vbase - U;
  10    =  10
errold = err;
```

Обратите внимание, как на третьем шаге Д-часть поменяла знак. В первый момент, когда ошибка текущая стала больше ошибки предыдущего шага, Д-часть была того же знака, что и П-часть и они работали вместе, возвращая робота к линии. В следующий момент ошибка начала уменьшаться и Д-часть стала сопротивляться. П-компонента тянет робота к линии, Д-часть мешает этому. Эта помеха и выводит робота плавно на линию.

Еще один очень важный момент состоит в том, что дифференциальная часть зависит от времени. Алгоритм оценивает скорость изменения ошибки, а скорость — это изменение чего-либо за единицу времени. Если итерации цикла будут выполняться с разной скоростью, то и разница (err-errold) будет принимать разные значения. А итерации будут разными по самым разным причинам: на некоторых шагах будут происходить сложные вычисления дальнейшего маршрута, на некоторых шагах будет происходить долгое считывание медленных датчиков и многим другим. И после каждой долгой итерации робот будет немного (или сильно, смотря какой порядок задержки) дергаться.

Избежать этого можно разными способами. Идеально вычисление Д-части делать по следующей формуле:

$$D = \frac{kd * (err - errold)}{dt}$$

В ней *dt* обозначает период времени от предыдущего шага до текущего. Если это время во много раз длиннее остальных шагов, то Д-составляющая станет пропорционально меньше. Вопрос лишь в том, как измерять это время на конкретной платформе: с помощью таймера, настроить прерывания в строго определенные моменты времени и жестко зафиксировать *dt* или как-то иначе.

В первом математическом описании дифференциальной части сделана задержка на 5 миллисекунд командой ПАУЗА(0.005 с). Это один из самых простых способов - намеренно делает большая пауза на каждой итерации. Предполагается, что все вычисления выполняются за более короткое время, примерно за 50 микросекунд. То есть пауза в 100 раз больше и общая длительность одной итерации составляет 50+5000=5050 микросекунд. Даже если какой-то шаг будет вычисляться значительно дольше обычного, 500 микросекунд, общая длина шага получится 500+5000=5500 микросекунд, то есть всего на 9% дольше. Пренебрегая изменением этой длительности, можно принимать *dt* за константу (постоянную величину) и подбирать не kd, а kd/dt.

Еще один способ, применимый на Arduino (с использованием регистров микроконтроллера ATMega) или STM - выполнение каждого нового вычисления регулятора по прерываниям таймера. В этом случае промежуток времени *dt* тоже строго определен и заранее задан в таймере, ситуация аналогична предыдущему способу.

Принцип работы интегральной части регулятора: чем дольше система находится не в состоянии уставки, тем сильнее она к ней стремится.

Эта компонента постепенно накапливает ошибку. Чем дольше она накапливается, тем сильнее становится И-часть. Эта компонента бывает полезна в тех случаях, если робот идет в затяжном повороте и никак не может приблизиться к линии. Или в балансирующих роботах, которым надо стать ровно в вертикальное положение.

Математически полный регулятор, содержащий все три компоненты, в том числе накапливающую ошибку И-часть, выглядит следующим образом:

```
ЦИКЛ (условие)
        err = S – ust;
        P = err * kp;
        I = I + ki * err;
        D = kd*(err – errold);
        U = P + I + D;
        Vleft = Vbase + U;
        Vright = Vbase – U;
        errold = err;
        ПАУЗА(0.005 с);
КОНЕЦ ЦИКЛА
```

Проводить пошаговый разбор значений я уже не буду, интегральная составляющая I просто плавно увеличивается на каждой итерации цикла.

Коэффициент интегральности ki должен быть на несколько порядков меньше kp. Например, если ki меньше kp в 100 раз, то И-часть станет такой же по силе как и П-часть только через 100 шагов (с заданной выше паузой - через 0,5 секунды).

Интегральная составляющая также зависит от времени. В идеале ее надо вычислять по следующей формуле:

$$I = I + ki * err * dt$$

Тут используется та же самая *dt*. И теми же способами можно от нее избавиться (сделать паузу или запускать вычисления по таймеру). Полный ПИД-регулятор с учетом dt описывается следующим псевдокодом:

```
T = текущее_время
ЦИКЛ (условие)
        dt = текущее_время – T
        T = текущее_время
        err = S – ust;
        P = err * kp;
        I = I + ki * err * dt;
        D = kd*(err – errold) / dt;
        U = P + I + D;
        Vleft = Vbase + U;
        Vright = Vbase – U;
        errold = err;
КОНЕЦ ЦИКЛА
```

Дополнительно можно проверять, равна ли ошибка нулю. В тот момент, когда ошибка становится нулевой (система приходит в состояние уставки) И-часть можно

принудительно обнулять. Если этого не делать, то И-часть останется с каким-то накопленным значением и продолжит тянуть систему в противоположное последней ошибки направление.

Иногда регуляторы строят на основе только И-части. Это делается в тех случаях, когда нужно плавное увеличение управляющего воздействия. Например, робот поворачивается по компасу. Азимут (уставка) резко изменяется, появляется ошибка. П- и Д-части резко дернут робота и его колеса начнут проскальзывать, буксовать. Если в регуляторе только И-часть, то она будет плавно накапливаться и постепенно увеличивать мощность на моторах, в результате робот плавно начнет движение без проскальзывания колес.

В большинстве задач достаточно П-, ПД- или даже релейных регуляторов. Не забывайте, что чем больше составляющих, тем больше коэффициентов требуется подбирать.

3.2. Плавный разгон и торможение

Для выполнения точных проездов очень важно, чтобы колеса робота не буксовали. Чаще всего пробуксовки могут случиться в начале движения и при остановке, когда скорость меняется слишком резко. Другими словами, когда робот испытывает большое ускорение, не зависимо от его знака. Поэтому скорость надо увеличивать и уменьшать плавно, без резких изменений, не превышая допустимого ускорения. Например, разгон с ускорением до 0.4 м/с^2 обычно не вызывает пробуксовок. То же самое касается торможения.

Для вычисления и соблюдения допустимого ускорения робота важно понимать взаимосвязь угловых и линейных скоростей, «тиков» энкодера с градусами поворота колес. Перед изучением дальнейшего материала рекомендуется убедиться, что разделы «Скорости линейные и угловые», «Ускорения линейные и угловые» и «Энкодер, тахометр, датчики тока» полностью изучены.

Одним из самых простых способов не допускать пробуксовок – сделать скорости всех движений заведомо малыми. Робот будет двигаться без пробуксовок, но выполнять задание слишком долго. Для соревнований, учитывающих время выполнения заезда, такой способ не подходит. Как не подходит и для многих производственных задач, в которых от скорости движения исполнительных органов зависит выработка продукции и конечная прибыль за единицу времени.

Наиболее эффективным и применимым на производстве способом управления приводом с ДПТ является подчиненное регулирование по положению, скорости и току.

Поэтому можно использовать такой способ:

```
00:    сброс таймера
01:    сброс энкодера левого мотора
02:    цикл (пока таймер меньше 1 секунды)
03:          измерение=считать значение таймера
```

```
04:         скорость=измерение*100
05:         запустить моторы в синхро-режиме со скоростью "скорость"
06:    конец цикла
```

3.3. Синхронизация приводов

Материал в процессе подготовки.

4. Дифференциальная платформа

Материал в процессе подготовки.

5. Платформа на омни-колесах

Материал в процессе подготовки.

6. Платформа на mecanum-колесах

Материал в процессе подготовки.

7. Одометрия

Материал в процессе подготовки.

8. Дополнительные материалы для изучения

Материал в процессе подготовки.