Карпов В.Э.

ПИД-управление в нестрогом изложении

Москва, 2012

Содержание

Введение

В этой работе на очень неформальном уровне излагаются основы построения ПИД-регуляторов – одной из основных компонент автоматики и робототехники. ПИД-регулятор – это то, что изучается в курсе Теории автоматического управления, и различного рода монографий, учебников, статей имеется великое множество. Однако специфика современной робототехники заключается в том, что в нее порой приходят люди, далекие от классического робототехнического образования. Если для «настоящего» робототехника Теория автоматического регулирования управления – это одна из основных дисциплин, то для тех, кто «приходит со стороны», изучение принципов работы автоматических устройств представляет определенные трудности. Настоящая работа представляет собой очередную попытку неформального, популярного описания основ построения ПИД-регулятора, причем с уклоном в робототехнику, если под этим уклоном понимать иллюстрацию теории на примере движения мобильного робота.

Постановка задачи

Начнем с постановки простой задачи. Пусть у нас имеется мобильная платформа (тележка, автомобиль), которая должна проехать из пункта **А** в пункт **Б**. Естественно, автоматически, без участия человека. Ехать она должна, скажем, по прямой. Для решения этой задачи можно "зафиксировать" рулевое управление. Или, если на тележке установлен дифференциальный привод (два мотор-редуктора), можно подавать одинаковое напряжение на оба двигателя. Однако в реальности такая система работать не будет. Рано или поздно тележка собьется с курса. Причин тому множество: и неровность покрытия дороги, и помехи, и недостаточно одинаковое напряжение питания, и вообще одинаковых моторов не бывает. Особенно задача усложнится тогда, когда надо будет ехать не по прямой, а по более замысловатой траектории.

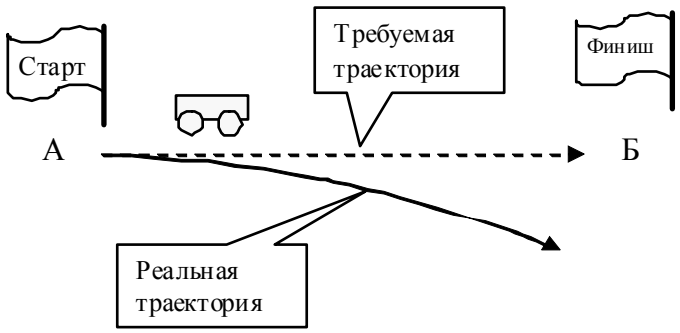


Рис. 1. Тележка должна проехать из пункта А в пункт Б

Как бы то ни было, для решения нашей задачи надо создавать специальную систему, управляющую движением тележки – *регулятор*. Построением регуляторов занимается замечательная наука – теория автоматического регулирования (ТАР), некоторые положения которой нам придется кратко изложить (да простят автора его коллеги и знакомые за упрощенчество).

Терминология. Как и положено, разберемся для начала с терминологией. Итак, пусть наша платформа представляет собой тележку с дифференциальным приводом – два электрических мотор-редуктора. Отметим, что конструктив пока никакой роли не играет, он понадобится лишь в дальнейшем. На тележке установлен некий блок – источник напряжения, питающий двигатели. Собственно, то, какое питающее напряжение будет выдавать на двигатели этот блок, и будет определять характер движения тележки. Это блок вкупе с двигателями и всей ходовой частью назовем *объектом управления*. Наша

задача как раз и будет заключаться в том, чтобы заставить блок выдавать нужное для требуемого движения тележки напряжение.

На вход объекта управления мы будем подавать т.н. *задающее воздействие*, т.е. некий сигнал, определяющий то, что должно быть на выходе объекта управления. Это задающее воздействие обозначим как $x_з(t)$, а выходной сигнал обозначим через $y(t)$. Например, если требуется, чтобы тележка ехала прямо, то $x_з(t)$ будет представлять собой постоянную величину. Если же надо, чтоб траектория движения тележки была иной, то $x_з(t)$ будет представлена соответствующей функцией. Здесь t, естественно, время.

Выходной сигнал $y(t)$ будет определять то, как тележка едет на самом деле. Вообще же задача регулятора состоит в том, чтобы у нас была такая система, в которой выход $y(t)$ соответствовал задающему воздействию $x_з(t)$ (сказали, что надо ехать прямо – значит тележка должна ехать прямо; сказали, что надо ехать по синусоиде, - значит так и должно быть).

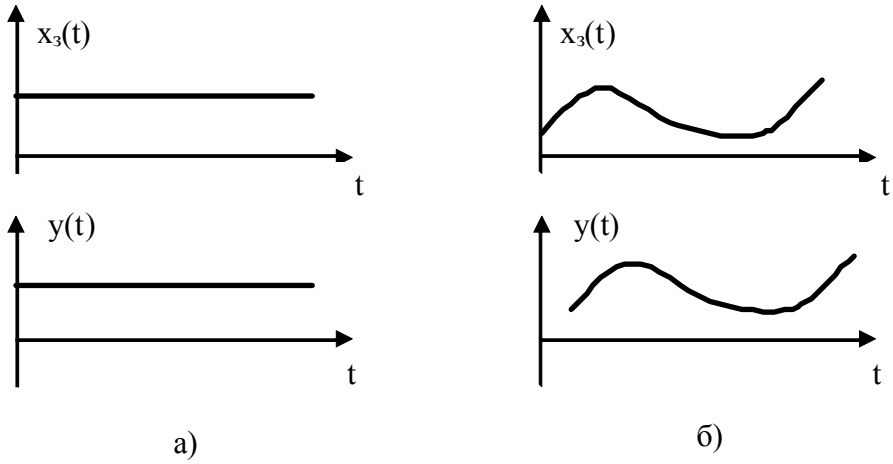


Рис. 2. а) Движение по прямой. б) Движение по сложной траектории

Теория автоматического регулирования (ТАР) – это такая дисциплина, которую конкретика задачи и устройство объекта управления не интересует. Задача ТАР – изучать информационные процессы, поэтому вместо красивых и наглядных натурных схем и рисунков (см. Рис. 3, а) в ТАР используются схематичные изображения – структурные схемы. Мы также будем ими иногда пользоваться, тем более, что это действительно удобно (не надо отвлекаться на детали).

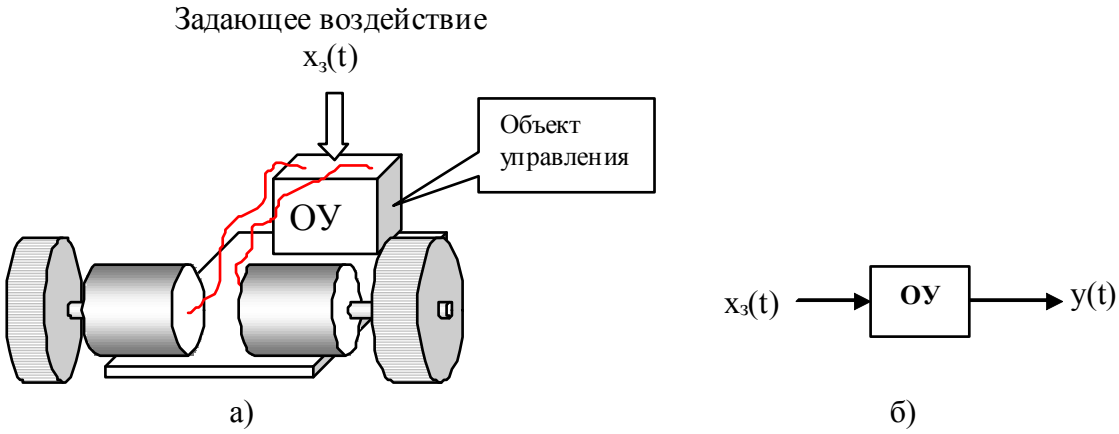


Рис. 3. Тележка (а) и соответствующая структурная схема (б)

Продолжим разбираться с терминологией. Мы уже говорили, что тележка сама по себе прямо ехать не умеет. Ей мешают всяческие неровности, неодинаковость моторов и проч. Всю эту совокупность мешающих движению факторов назовем *помехами* и

обозначим как *z(t)*. На самом деле нам не столь важно, от чего конкретно произошел сбой в движении. Нам важно этот сбой как-то распознать и отреагировать на него нужным образом (откорректировать движение, т.е. значение выходного сигнала *y(t)*). А для этого нам важно знать, насколько траектория движения тележки отклонилась от требуемого курса (того, что задается величиной $x_з(t)$). Это отклонение называется *сигналом ошибки* и обозначается как *e(t)*.

На структурной схеме помеха z(t) изображена как нечто, воздействующее на объект управления.

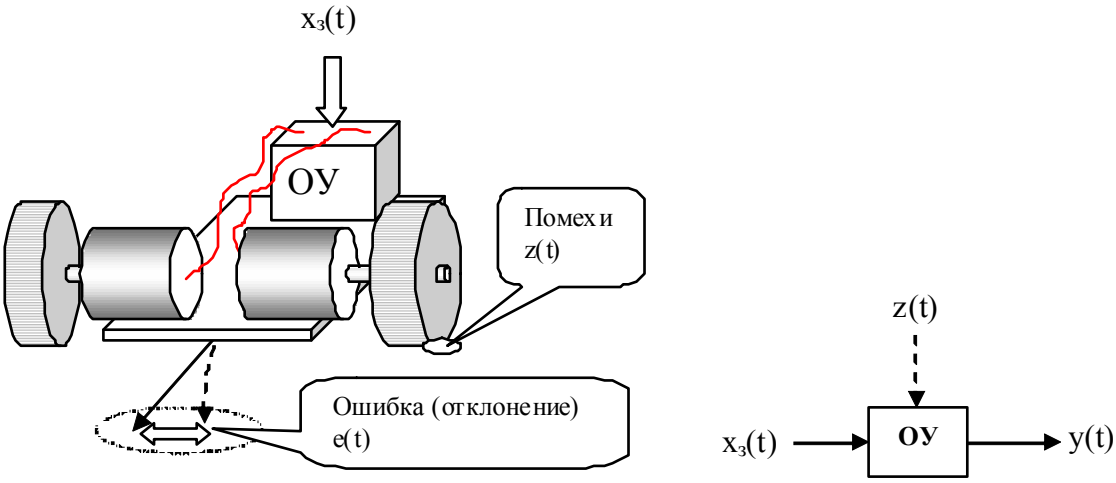


Рис. 4. Помехи и ошибки

Итак, чтобы заставить тележку ехать по заданной траектории, необходимо уметь определять ошибку в движении, вызванную помехами. Для этого можно (и нужно) оснастить тележку датчиками, которые и будут измерять отклонения от курса. Например, можно нанести на дороге линию разметки и установить на тележке датчики полосы. И тогда мы получим всем знакомую классическую задачу отслеживания линии. Сигналы от датчика полосы будут поступать на некоторый блок, называемый *устройством управления*. Устройство управления (УУ), получая сигнал от датчиков, определит отклонение и, в зависимости от его величины, заставит объект управления выдать соответствующий выходной сигнал. По этому сигналу тележка изменит направление своего движения, датчики определят новое положение (и новую ошибку) и все повторится сначала. Иными словами, мы получим систему с *обратной связью*.

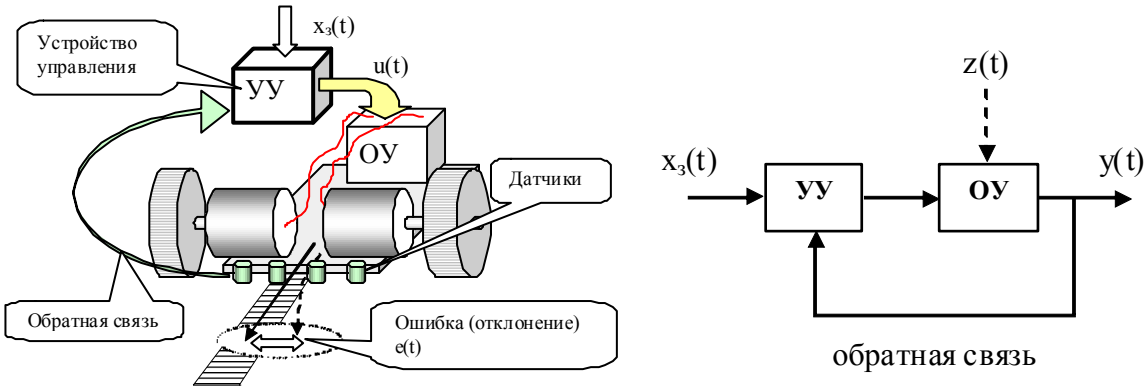


Рис. 5. Полная схема системы с обратной связью

Вообще, обратная связь – это краеугольный камень, основополагающий принцип ТАР, о ней мы поговорим особо. А сейчас следует заметить, что все, о чем говорилось выше – это прелюдия к "главной детали" – устройству управления. Собственно, именно созданию устройства управления (УУ) и посвящена эта статья.

Обратная связь. Итак, обратная связь – это способ учета ошибок в управлении. Осуществляется она с помощь датчиков, которые эти ошибки в управлении измеряют. Схематически это представлено на Рис. 6. Обратите внимание на тот узел, на который поступает сигнал обратной связи.

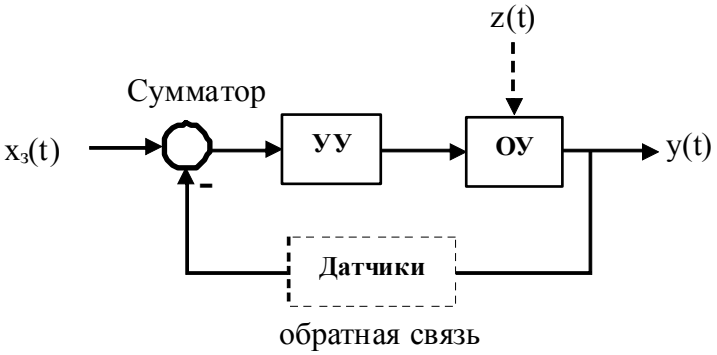
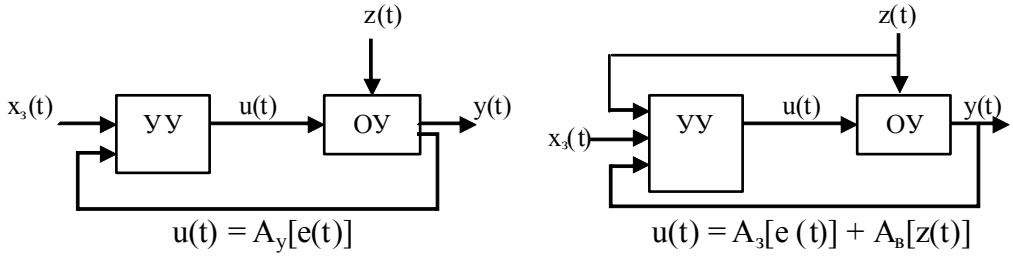


Рис. 6. Структурная схема системы

Этот узел представляет собой *сумматор* – устройство, складывающие два сигнала: задающее воздействие $x_з(t)$ и реальный выходной сигнал $y(t)$. При этом вход сигнала обратной связи помечен минусом, т.е. на устройство управления **УУ** поступает *разность* этих сигналов. Именно эта разность и представляет собой сигнал ошибки (разность между тем, что нужно и что имеется реально), на который должно реагировать **УУ**. Это – т.н. система с отрицательной обратной связью. Вообще, обратная связь – это особый и очень непростой вопрос. Существуют регуляторы и без обратной связи, сама обратная связь может быть и отрицательной (сигналы вычитаются), и положительной (сигналы складываются).

Сами схемы управления могут быть тоже различными. На Рис. 7 представлены две из них.



а) б)

Рис. 7. Варианты структурных схем

При этом на Рис. 7,а изображена система, в которой сигнал от **УУ** *u(t)* является функцией от ошибки *e(t)* (и мы пишем $u(t) = A_y[e(t)]$), а на Рис. 7,б сигнал управления *u(t)* зависит еще и от внешнего возмущения ($u(t) = A_з[e(t)]+A_в[z(t)]$). Это – уже более сложный вариант и мы не будем на нем останавливаться.

Подытожим. Итак, для решения задачи управления мы создаем систему с *обратной связью (система с замкнутой цепью воздействий* или просто *замкнутая система)*. На вход **УУ** поступают как внутреннее (контрольное) воздействие, так и внешнее (задающее). Управление осуществляется по сигналу ошибки *e(t)*:

$$e(t) = x_з(t) - y(t)$$

Типовые законы управления

Настала пора разобраться с тем, каким образом устройство управления будет воздействовать на объект управления, как учитывать ошибки. Иными словами, как будет выглядеть закон управления. Вариантов таковых законов (принципов) управления может быть множество. Мы рассмотрим некоторые из них.

1. Пропорциональное управление

Итак, пусть наша машина отклонилась от траектории и наши датчики зафиксировали это отклонение, т.е. мы знаем сигнал ошибки $e(t)$ – насколько мы ушли от центра линии. При этом, обратите внимание, задающее воздействие $x_з(t)$ у нас равно 0, причем постоянно. Это может выглядеть несколько странно, однако все очень просто. Равенство $x_з(t)$ *нулю* означает, что машина не должна отклоняться от центра линии. Будем считать, что датчики фиксируют местоположение машины на линии. Когда мы строго на ней, то выходной сигнал $y(t)$ равен нулю. Отклонились влево-вправо - $y(t)$ будет уже иным. Как бы то ни было, в каждый момент времени мы знаем величину сигнала ошибки $e(t) = x_з(t) - y(t)$.

Теперь осталось понять, что нам с этим знанием делать, т.е. как реагировать на эту ошибку.

Самый простой и естественный вариант – реагировать на нее в соответствии с ее – ошибки – значением. Сильно отклонились (ошибка велика) – будем сильно выкручивать руль (изменим скорость вращения левого или правого мотора), отклонились чуть-чуть – и подвернем тоже слегка. Это – т.н. *пропорциональный закон управления*.

Пропорциональное управление является самым простым в реализации и наиболее часто используется в управляющих системах. Здесь регулятор просто берет отклонение (сигнал ошибки) $e(t)$, умножает его на константу и выдает его в качестве управляющего воздействия $u(t)$.

Из того, что управляющее воздействие $u(t)$ пропорционально сигналу ошибки, следует, что в качестве регулятора можно использовать самый обыкновенный *усилитель*.

Структурная схема системы представлена на Рис. 8.

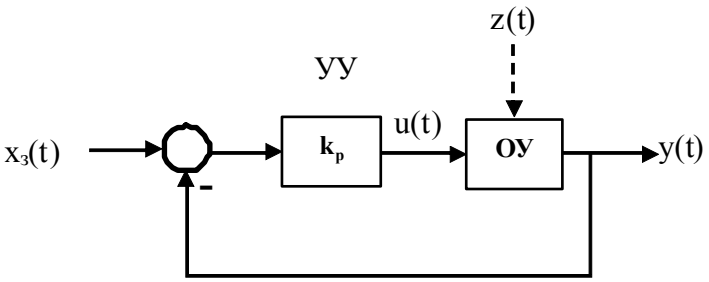


Рис. 8. Система с пропорциональным регулятором

Здесь в качестве УУ используется простой усилитель, который усиливает сигнал ошибки в k_p раз. Мы получили т.н. **П-регулятор**. Такой регулятор прост (всего-навсего усилитель, причем хоть в аппаратном, хоть в программном исполнении) и он сразу же реагирует на ошибку, т.е. регулятор быстр.

	Примечание для сведущих. Передаточная функция П-регулятора $W_{рег}(p) = u(p)/e(p) = k_p.$ Это – т.н. идеальное пропорциональное звено.

Но все не так просто. За все надо платить. У любого решения (не только технического) есть свои положительные и отрицательные стороны. Рассмотрим их.

Преимущества. Они очевидны. Это – упомянутые простота и быстродействие.

Недостатки: ограниченная точность (особенно при управлении объектами с большой инерционностью и запаздыванием), перерегулирование.

Если с преимуществами все достаточно ясно, то с недостатками следует разобраться особо.

Качество управления

Сделаем небольшое отступление от законов управления и поговорим о качестве управления. Вопрос заключается в том, каким образом мы можем судить о том, хорош регулятор или плох. Понятно, что если система управления не в состоянии удержать машину на трассе, то она плоха. Но ведь и по трассе можно ехать по-разному.

На Рис. 9 представлены различные варианты движения нашей платформы. Предположим, что из-за ошибки платформа сбилась с линии. Не будем придираться к рисунку и будем считать, что платформа все-таки видит линию и пытается на нее вернуться. (Платформа нарисована целиком вне линии лишь для наглядности. В противном случае сложно рисовать).

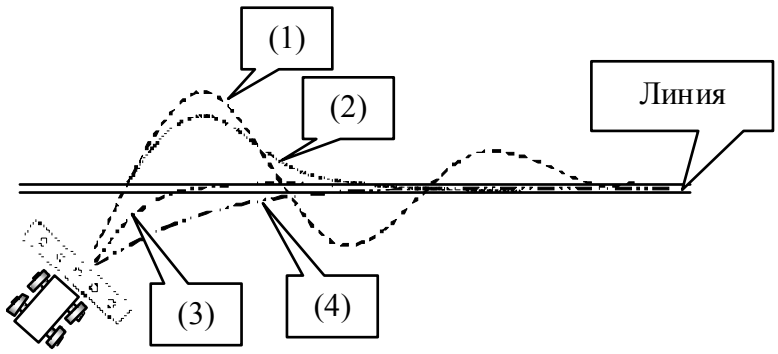


Рис. 9. Варианты движения тележки

В зависимости от величины коэффициента усилителя k_p нашего регулятора машина будет вести себя по-разному:

Траектория (1). k_p велик (скажем, $k_p = 10$). Система начинает совершать колебания. Причем амплитуда колебаний может быть большой. Величина максимального отклонения траектории от заданной называется *перерегулированием*. Чем это перерегулирование меньше, тем лучше.

Траектория (2). k_p меньше (например, $k_p = 5$). Система ведет себя получше. Но перерегулирование велико.

Траектория (3). $k_p = 2$. Машина, не совершая колебаний, выходит на трассу. Это – лучшее управление.

Траектория (4). k_p мало ($k_p = 1$). Выход на трассу тоже осуществляется без колебаний, однако значительно медленнее.

По этим примерам видно, что при пропорциональном управлении возможны неприятности: колебательность, перерегулирование и большое *время переходного процесса* (это время, за которое наша машина выйдет на трассу и перейдет в т.н. *установившийся режим*).

Примечание для сведущих. На самом деле, ситуация на Рис. 9 не является корректной. О качестве регулятора судят по реакции системы на единичное воздействие – специальную функцию, скачком принимающую значение, равное 1. И тогда мы получим то, что называется графиком переходного процесса:

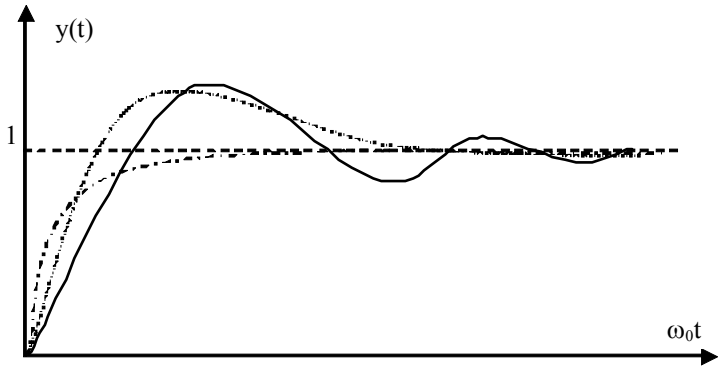


Рис. 10. Переходные процессы: реакция системы на единичное воздействие

Коли мы упомянули термины *установившийся и переходной режим*, то рассмотрим их подробнее. Тем более что все это крайне важно для понимания сути работы системы управления.

Режимы работы системы

В теории автоматического регулирования различают два основных режима работы системы – *установившийся* режим и *переходной* режим. Для пояснения разницы между ними вернемся к нашему примеру с движением нашей платформы по линии.

Установившийся режим – это рабочий режим системы, когда все предварительные действия закончены (мы четко выехали на трассу) и машина едет по относительно прямому участку.

Переходный режим – это начало работы, когда наша машина стартует или резко меняются «дорожные условия» (резкие повороты, изломы трассы и т.п.). Это то, что изображено на Рис. 9.

Это – очень важные понятия, т.к. мы хотим иметь «хорошую», качественную систему управления. Нам очень важно, чтобы на длинных, прямых участках (установившийся режим) машина ехала максимально быстро, с минимальными отклонениями от линии, т.к. каждое отклонение увеличивает путь. Это означает, что нас здесь интересует прежде всего **точность** управления.

Когда же мы только стартуем или начинается сложный извилистый участок (переходный режим), то здесь важнее не точность, а **скорость** реакции системы. Нам важнее уметь как можно быстрее реагировать на изменения трассы, чтобы не «вылететь» с нее, а уж насколько точно это получается - неважно.

Кончено, «идеальная» система управления должна сочетать в себе все эти свойства – и точность, и скорость, однако на практике, как мы помним, надо быть готовым к компромиссам.

Итоги и программная реализация

Итак, пропорциональное управление просто в реализации, однако имеет склонность к перерегулированию. Последнее означает, помимо всего прочего, что на сложных извилистых участках мы будем тратить много времени на совершение колебательных движения, что чревато не только потерей скорости, но и риском вылета с трассы. Тем не менее, без П-управления нам не обойтись.

Функция для контроллера, реализующего П-управление, может выглядеть так:

```
float kp = 10;  // Коэффициент пропорционального звена

float Pctl(float error)
{
  float up;
  up = kp*error;
  return up;
}
```

Функции передается один параметр – величина сигнала ошибки *error*, а возвращает она значение управляющего сигнала *u* – величину ошибки, умноженную на коэффициент усилителя П-регулятора k_p.

2. Пропорционально-дифференциальное управление

Продолжим наши рассуждения относительно качества регулятора. Очевидно, что П-регулятор не может удовлетворить все наши потребности. Представим себе ситуацию, в которой ошибка управления начинает резко возрастать. Например, мы въезжаем на участок трассы с резкими поворотами. В такой ситуации надо реагировать как-то иначе. Более извилистый участок означает, что "крутить руль" надо более интенсивно. Более того, здесь уже не до точности управления – здесь важнее как можно быстрее отреагировать на поворот и не сойти с трассы (не потерять ее). Это означает, что надо уметь реагировать не только на ошибку управления, но и на *скорость ее изменения*.

Нам очень важно знать не только *настоящее* (текущую ошибку), но и *будущую ошибку* (знание о скорости изменения ошибки и даст нам эту информацию). Если мы знаем, что ошибка увеличивается (это и есть скорость изменения ошибки), то надо увеличивать управляющее воздействие.

Элементом, способным измерить эту скорость, является т.н. *дифференциатор*.

Дифференциатор – это достаточно простой в реализации элемент. Его задача – умножить разность между текущим значением выходного сигнала и значением выхода в предыдущий момент времени на какой-то постоянный коэффициент:

$$u_d(t) = k_d*(y(t) - y(t-1))$$

Здесь $u_d(t)$ – значение управляющего сигнала, k_d – постоянный коэффициент, $y(t)$ – текущее значение выходного сигнала (в момент времени t), $y(t-1)$ – предыдущее значение выходного сигнала (в момент времени t-1).

Работает эта компонента (Д-компонента) достаточно очевидно. Если на выходе у нас постоянный сигнал ($y(t)=y(t-1)$), то значение $u_d(t)$ равно нулю и никакого изменения не происходит. Если же что-то начинает меняться ($y(t)\neq y(t-1)$), то соответственно начинает изменяться величина дифференциальной компоненты. Причем, чем больше разнятся между собой значения выходного сигнала, тем больше будет вклад этой компоненты.

Отсюда, кстати, следует вывод о том, что использование одной лишь Д-компоненты в регуляторе совершенно неприемлемо. Если ошибка управления постоянна (всегда $y(t)=y(t-1)$), то эта компонента никак не прореагирует на это. Поэтому Д-компонента работает в паре с пропорциональной компонентой.

И тогда мы получаем следующую схему регулятора, который называется *пропорционально-дифференциальным* (ПД-регулятором):

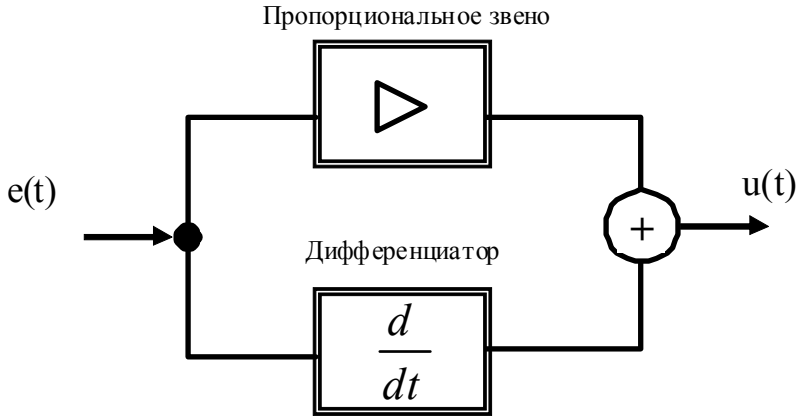


Рис. 11. ПД-регулятор. Состоит из П- и Д- компонент

Здесь сигнал ошибки поступает как на пропорциональное звено (усилитель), так и на дифференцирующее звено (дифференциатор). Далее выходные сигналы этих компонент складываются, формируя управляющее воздействие *u(t)*. Таким образом, работа ПД-регулятора описывается следующим соотношением:

$$u_{pd}(t)= u_p(t) + u_d(t) = k_p*e(t) + k_d*(y(t) - y(t-1))$$

	Примечание для сведущих. Передаточная функция ПД-регулятора $W_{рег}(p) = k_n + k_д p = k_p + k_d T_d p$ Здесь T_d – т.н. постоянная времени. Кроме того, речь здесь идет о т.н. *идеальном* дифференцирующем звене. У реального звена передаточная функция сложнее.

Пропорционально-дифференциальный закон управления уже более интересен, нежели просто пропорциональное управление. Но есть у него и свои серьезные особенности. Обозначим пока его достоинства и недостатки.

Достоинства. Этот закон управления имеет наивысшее быстродействие. ПД-регулятор реагирует не только на величину отклонения *e(t)*, но, что наиболее важно, на скорость ее изменения.

Недостатками ПД-регулятора являются малая точность и чувствительность к шумам.

Как водится, с достоинствами все гораздо проще, чем с недостатками. О недостатках надо поговорить подробнее. Дело в том, что дифференциальная компонента управления является самой проблемной и капризной из всех типов управления. И связано это как раз с тем, что Д-компонента очень чувствительна к скорости изменения ошибки, ведь, как ни странно, не на каждую ошибку надо реагировать.

1. Проблема шумов. Дело в том, что все шумит. Шумят датчики, выдавая не всегда то, что необходимо. Шумит трасса (линия нарисована не всегда четко, имеются всякие неоднородности), шумят исполнительные схемы и механизмы. Д-компонента реагирует на все эти шумы, заставляя систему совершать ненужные действия, реагировать на то, на что реагировать вовсе не обязательно. Особенно остро эта проблема касается *высокочастотных* шумов. Перечисленные выше шумы относятся как раз к категории высокочастотных – они возникают на короткое время, но могут иметь большое значение. А Д-регулятор упорно будет на них реагировать. В то же время изгибы трассы относятся к *низкочастотным* помехам (по сравнению со скоростью процессов в системе управления), и регулятору надо бы реагировать именно на них. Выходом является установка низкочастотного фильтра на входе Д-компоненты (этот фильтр будет отсекать высокочастотные помехи), однако это уже выходи за рамки данной статьи.

2. Чувствительность к частоте сбора информации. Поскольку в ПД-законе фигурирует сигнал "в предыдущий момент времени", то очень важно, чтобы интервал времени сбора информации выдерживался как можно более строго. Если время "будет плавать", то ни о какой адекватности реакции системы речи быть не может. Считается, что временные интервалы должны выдерживаться с точностью не менее 1%. Это само по себе не так просто. При программной реализации выходом может является использование прерываний. Именно в обработчике прерываний должна содержаться короткая (и быстрая) процедура сбора информации от датчиков.

Таким образом, главная проблема ПД-регулятора заключается в том, что он усиливает шумы. И если от сильных шумов в системе не удается избавиться, то Д-компоненту, пожалуй, лучше вовсе не использовать.

Итоги и программная реализация

Итак, пропорционально-дифференциальное управление обеспечивает наилучшее быстродействие. Более того, благодаря тому, что ПД-регулятор реагирует не только на величину сигнала ошибки, но и на скорость его изменения, при управлении достигается эффект упреждения (прогнозирования поведения ошибки). Недостатками же ПД-закона является ограниченная точность и чувствительность к шумам.

Функция для контроллера, реализующего ПД-управление, может выглядеть так:

```
float kp = 10;  // Коэффициент пропорционального звена
float kd = 1;   // Коэффициент дифференциального звена
float old_y = 0; // Предыдущее значение сигнала

float PDctl(float error, float y)
{ float up, ud;

  // Пропорциональная компонента
  up = kp*error;

  // Дифференциальная компонента
  ud = kd*(y-old_y);
  old_y = y;

  return up+ud;
}
```

Функции передается два параметра – величина сигнала ошибки *error* и значение выходного сигнала *y*. Возвращает она значение управляющего сигнала *u* – сумму пропорциональной и дифференциальной компонент регулятора. Глобальная (а можно сделать ее и статической) переменная *old_y* служит для хранения предыдущего значения выходного сигнала.

3. Интегральное управление

Итак, основная проблема рассмотренного выше ПД-регулятора заключается в чувствительности к шумам. Можно ли сделать так, чтобы шумы сглаживались и при этом точность регулирования в установившемся режиме была максимальна? Видимо, да (и в этом прелесть риторических наводящих вопросов). Для этого можно использовать т.н. *интегрирующее звено* (или просто *интегратор*).

Интегратор занимается тем, что складывает (накапливает, суммирует, интегрирует) сигнал ошибки *e(t)*. Управляющий сигнал *u(t)* в каждый момент времени пропорционален интегралу ошибки *e(t)*. Из этого можно сделать вывод о том, что И-регулятор реагирует на длительные отклонения управляемой величины, а кратковременные отклонения им сглаживаются.

Действительно, если считать, что интеграл – это площадь фигуры, определяемой некоторой функцией (в нашем случае, поскольку мы интегрируем ошибку – функцией *e(t)*), то площади фигур а) и б) на Рис. 12 примерно одинаковы.

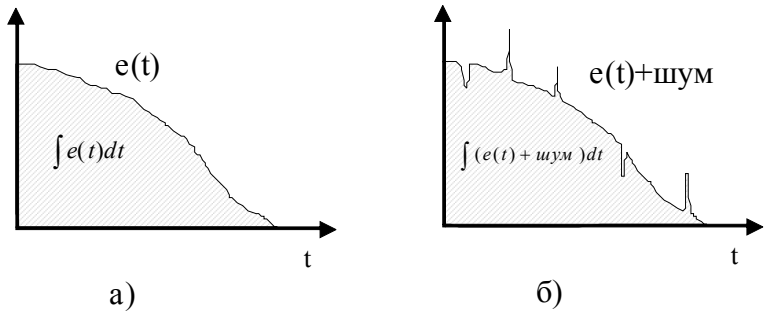


Рис. 12. Интегралы от функции ошибки примерно равны: а) функция ошибки без высокочастотных помех, б) функция ошибки с высокочастотными помехами

Итак, интегральный закон управления (И-регулятор) выглядит следующим образом:

$$u_i(t) = k_i * \int e(t)dt \approx k_i * \sum e(t)\Delta t$$

	Примечание для сведущих. Передаточная функция И-регулятора $W_{рег}(p) = k_i/p = k_и/(T_i p)$ Здесь T_i – постоянная времени интегратора. Кроме того, и здесь мы имеем дело с *идеальным* интегратором. У реального интегрирующего звена передаточная функция иная.

Однако интеграция ошибок – это процесс опасный. Ошибки имеются всегда, и просто накопление их приводит к снижению стабильности системы или вообще делает систему нестабильной.

Чистое И-регулирование приведет к тому, что колебания системы будут становиться все больше и больше, пока система не пойдет вразнос. Именно поэтому интегратор используется вместе с пропорциональным звеном. В этом случае мы получаем пропорционально-интегральный закон управления (ПИ-регулятор):

$$u_{pi}(t) = u_p(t) + u_i(t) = k_p * e(t) + k_i * \int e(t)dt \approx k_p * e(t) + k_i * \sum e(t)\Delta t$$

На Рис. 13 представлена структурная схема ПИ-регулятора.

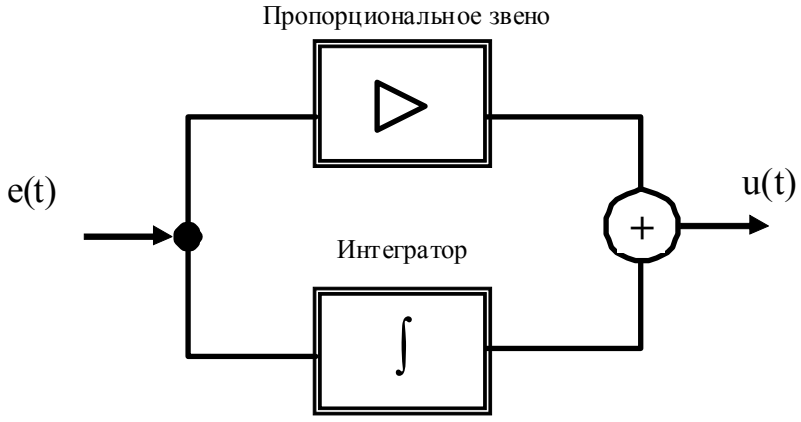


Рис. 13. ПИ-регулятор. Состоит из П- и И- компонент

	Примечание для сведущих. Передаточная функция ПИ-регулятора $W_{рег}(p) = k_п + k_и/p$

Итак, интегральное управление используется, чтобы добавить "долгосрочной точности" управлению и практически всегда используется совместно с пропорциональным управлением. Параллельное соединение двух звеньев позволяет использовать достоинства П- и И- регуляторов.

Кстати, ПИ-регуляторы имеют наибольшее распространение в промышленной автоматике.

Преимущества: лучшая по сравнению с П-регулятором точность в установившемся режиме, а при определенном соотношении коэффициентов $k_п$ и $k_и$ ПИ-регулятор обеспечивает хорошие показатели и в переходных режимах.

Недостатки: худшие свойства в переходных режимах (меньшее быстродействие и большая колебательность).

Для того, чтобы система не шла вразнос из-за постоянного накопления ошибок, работу интегратора обычно ограничивают, т.е. определяют минимальное и максимальное значения накопленного сигнала. Это обычно позволяет предотвратить т.н. "вылет системы" - неограниченный рост управляющего воздействия.

Итоги и программная реализация

Итак, ПИ-регулятор хорош в установившемся режиме. Он максимально точен и не реагирует на высокочастотные шумы. Правда, в переходных режимах его использование может вызвать появление колебательных процессов. К тому же он достаточно медленный.

Функция для контроллера, реализующего ПИ-управление, может выглядеть так:

```
float kp = 10;    // Коэффициент пропорционального звена
float ki = 0.001; // Коэффициент интегрального звена

#define iMin -0.2  // Минимальное значение интегратора
#define iMax 0.2   // Максимальное значение интегратора

float iSum = 0;  // Сумма ошибок (значение, накопленное в интеграторе)

float PIctl(float error)
{
  float up, ui;

  // Пропорциональная компонента
  up = kp*error;

  // Интегральная компонента
  iSum = iSum+error; // Накапливаем (суммируем)
  if(iSum<iMin) iSum = iMin;  // Проверяем граничные значение
  if(iSum>iMax) iSum = iMax;
  ui = ki*iSum;

  return up+ui;
}
```

Переменная iSum – это наш интегратор, который хранит сумму всех предыдущих ошибок. Величины iMin и iMax - это минимальное и максимальное разрешенные значения состояния интегратора. Они-то и ограничивают интегральное воздействие.

4. Пропорционально-интегрально-дифференциальное управление

Осталось собрать все воедино, и мы получим, наконец, наиболее гибкий закон управления – *пропорционально-интегрально-дифференциальный*. Варьирование его параметров позволяет реализовывать все остальные законы. Он объединяет, естественно, все достоинства и недостатки законов, его составляющих.

Каждый из элементов регулятора (пропорциональное, интегральное и дифференциальное звенья) выполняет свою задачу и оказывает свое специфическое воздействие на функционирование системы: пропорциональный закон отвечает за настоящее (реагирует на текущую ошибку), дифференциальный – за будущее (реагирует на тенденцию изменения ошибки), а интегральный – за прошлое (накапливая предыдущие ошибки и сглаживая высокочастотные шумы).

Выходы этих элементов складываются между собой и формируют управляющий сигнал для устройства. На Рис. 14 представлена схема полученного ПИД-регулятора.

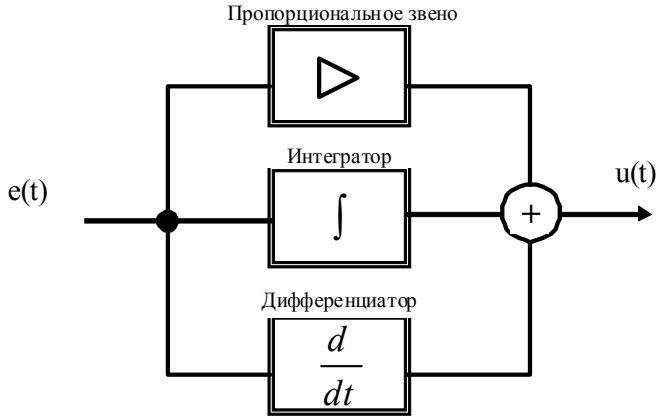


Рис. 14. ПИД-регулятор

Закон ПИД-управления выглядит как сумма входящих в состав ПИД-регулятора управляющих компонент:

$$u_{pid}(t) = u_p(t) + u_i(t) + u_d(t) = k_p * e(t) + k_i * \int e(t)dt + k_d * (y(t) - y(t-1)$$

	Примечание для сведущих. Передаточная функция ПИД-регулятора $W_{рег}(p) = k_п + k_и/p + k_д p$.

Итоговая структурная схема системы, использующей ПИД-регулятор, изображена на Рис. 15.

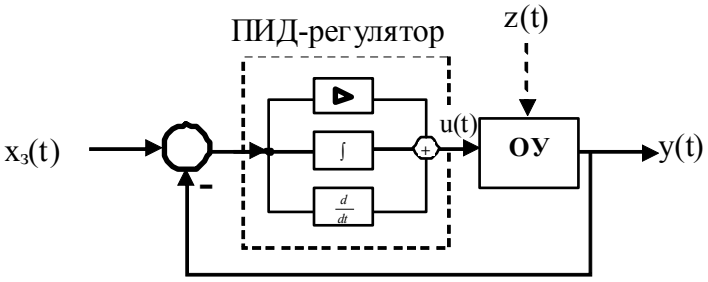


Рис. 15. Система с ПИД-регулятором

ПИД-регулятор – это старое изобретение. Он был создан еще в 1910 году, однако значительно позже, лишь в 1942 г. была разработана методика его настройки (Зиглер и Никольс).

Несмотря на свою распространенность, долгое время ПИД-регулятор был достаточно сложным и дорогим устройством. Но после появления микропроцессоров в 80 - х гг. развитие и распространение (где надо и где не надо) ПИД-регуляторов стало происходить нарастающими темпами. Сейчас ПИД-регулятор относится к наиболее распространенному типу регуляторов. Считается, что почти 90% регуляторов, находящихся в настоящее время в эксплуатации, используют ПИД алгоритм. Причиной столь высокой популярности является прежде всего их низкая стоимость. Правда, в последние годы наблюдается тенденция вытеснения "классических" ПИД-регуляторов их аналогами - регуляторами, использующих т.н. "нечеткую логику". Нечеткие регуляторы, не уступая в быстродействии, все-таки проще в настройке и более универсальны.

Итоги и программная реализация

Функция, реализующая ПИД-управление, представляет собой простую композицию всех рассмотренных выше компонент:

```
// Параметры пропорционального звена
float kp = 10; // Коэффициент пропорционального звена

// Параметры интегратора
float ki = 0.001; // Коэффициент интегрального звена

#define iMin -0.2  // Минимальное значение интегратора
#define iMax 0.2   // Максимальное значение интегратора

float iSum = 0;  // Сумма ошибок (значение, накопленное в интеграторе)

// Параметры дифференциатора
float kd = 1;   // Коэффициент дифференциального звена
float old_y = 0; // Предыдущее значение сигнала

float PIDctl(float error, float y)
{ float up, ui, ud;

  // Пропорциональная компонента
  up = kp*error;

  // Интегральная компонента
  iSum = iSum+error; // Накапливаем (суммируем)
  if(iSum<iMin) iSum = iMin;  // Проверяем граничные значение
  if(iSum>iMax) iSum = iMax;
  ui = ki*iSum;

  // Дифференциальная компонента
  ud = kd*(y-old_y);
  old_y = y;

  return up+ui+ud;
}
```

Функция ПИД-регулятора действительно очень проста. Основной же проблемой при реализации ПИД-управления является настройка его параметров.

Настройка ПИД-регулятора

Общие соображения

Настройка параметров ПИД-регулятора (коэффициентов k_p, k_i и k_d) – это самый сложный процесс. Эта настройка зависит от состава нашей системы, характеристик ее составных частей, от решаемой задачи. Даже одна и та же тележка должна иметь свои настройки для каждой трассы. Для извилистых трасс с резкими поворотами должна активно работать дифференциальная компонента (большой коэффициент k_d), для трасс с плавными поворотами и прямыми участками важнее интегральная компонента.

Рассмотрим, как должна выглядеть настройка коэффициентов.

1. Настройка теоретическая (как решает эту задачу математик). Для настройки необходимо нарисовать структурную схему системы. При этом необходимо знать все характеристики всех входящих в систему компонент (для сведущих: надо знать передаточные функции всех звеньев). Далее вычисляется то, что называется передаточной функцией системы. По этой передаточной функции можно определить качество переходных процессов, точность и т.п. Т.е. рассчитать коэффициенты регулятора так, чтобы наша задача решалась оптимальным образом.

Это – идеальная ситуация. На практике все выглядит гораздо хуже. Основная проблема в том, мы вряд ли сможем получить адекватную математическую модель системы, вряд ли мы учтем все нелинейности, случайности и т.п. А если учитывать и описывать все по максимуму, то мы просто погрязнем в математических расчетах.

2. Настройка инженерная (как должна решаться задача грамотным специалистом). Необходимо создать отладочный стенд, состоящий из тестового оборудования, системы его связи с компьютером, осциллографом и прочими полезными измерительными приборами. Далее следует подавать на вход системы ступенчатое воздействие (ту самую единичную функцию, которую мы как-то упоминали выше) и анализировать отклик системы. Этот отклик, называемый переходной функцией, и будет анализироваться, по нему и будут подбираться коэффициенты регулятора.

Этот путь пригоден для грамотных инженеров, обладающих и познаниями в ТАР, и временем, и оборудованием.

3. Настройка реальная (как это обычно делается). На практике (если только мы имеем дело не со специалистом в теории управления, которому все то, что здесь написано читать вовсе не нужно) все выглядит гораздо примитивнее. Коэффициенты будут определяться сугубо экспериментальным путем, используя натурные испытания. Просто-напросто, тележка будет ставиться на линию, а мы будем смотреть, как она едет. И изменять значения коэффициентов до тех пор, пока она не поедет как надо (или мы поймем, что на этой кривой трассе это убогое оборудование лучше уже не поедет).

Настройка компонент

1. Настройка пропорциональной компоненты

Обнулим коэффициенты k_i и k_d. Они нас пока не интересуют. Будем разбираться с k_p, варьируя его значение, скажем, от 1 до 100. Установим сначала значение k_p, равное 1. Если система очень медленно выходит на линию, то k_p надо увеличивать. Если же начинаются колебания, то k_p надо уменьшать.

При этом некоторые рекомендуют следующую методику изменения значений коэффициента. Установим сначала маленькое значение k_p. Допустим, колебаний еще нет. Далее увеличиваем это значение в 10 раз, пока не начнутся колебания. Теперь уменьшаем значение коэффициента k_p, но не в 10 раз, а в 2 раза. И так до тех пор, пока колебания не прекратятся. И так далее. Т.е. мы ищем искомое значение, сначала используя большие шаги, а затем все меньшие.

2. Настройка интегральной компоненты

Значение коэффициента интегральной компоненты k_i должно быть мало по сравнению с k_p. В качестве начального значения коэффициента k_i рекомендуется брать число от 0.0001 до 0.01. Процедура поиска (подбора) коэффициента k_i точно такая же, как и коэффициента пропорциональной составляющей (сначала большие шаги, а затем маленькие). Слишком большое значение коэффициента k_i также проявляется в появлении колебаний.

3. Настройка дифференциальной компоненты

Если мы уверены, что шумы в нашей системе на слишком велики, то можно ввести и дифференциальную компоненту. Для настройки k_d установим сначала значение этого коэффициента, равное 0. Далее установим какое-нибудь небольшое значение коэффициента пропорционального звена k_p (например, k_p =1). Главное, что значение k_p должно быть таким, чтобы система при нулевом значении k_d не совершала колебаний.

Далее установим какое-нибудь небольшое начальное значение коэффициента k_d (например, k_d =0.1).

Будем увеличивать коэффициент k_d до тех пор, пока не станут проявляться ошибочные колебания, вызванные малыми шумами. При этом колебания от слишком большого коэффициента происходят значительно быстрее, чем колебания от недостаточного коэффициента. Рекомендуется устанавливать коэффициент в половину или четверть от того, при котором начинаются колебания от слишком большой его величины. Главное в этом процессе – вовремя убедиться в том, что поведение системы является адекватным (машина ведет себя на трассе хорошо).

Замечания

Описанные процедуры настройки коэффициентов содержат много субъективизма. Главным критерием того, что мы нашли подходящие коэффициенты (не оптимальные, конечно), является внешнее поведение тележки. Но помимо таких сугубо экспериментальных подстроек, имеются и вполне объективные принципы, которые надо соблюдать при создании регулятора.

1. **Скорость работы программы.** Управляющая программа должна работать максимально быстро. Это означает, что необходимо использовать максимально лаконичные алгоритмы и простые операции.

2. Равномерность работы программы. Не менее важно то, чтобы формирование управляющего воздействия происходило в как можно более равномерные моменты времени. Нельзя допускать, чтобы программа "задумалась" на непредсказуемое время, т.к. в этом случае сбой дадут и интегральное, и дифференциальное звено. Для этого, повторим, можно использовать прерывания, которые гарантированно обеспечат нам равномерный сбор информации от датчиков.

3. Частота сбора данных и выполнения цикла управления. Помимо стабильности частоты сбора, необходимо определиться с тем, какова должна быть частота управления. Дело в том, что если частота управления слишком маленькая, то мы не получим эффективную систему. Более того, при малой частоте управления мы можем получить систему, которую вообще невозможно стабилизировать. Но и при слишком большой частоте мы тоже получим плохой результат: начнут возникать серьезные шумы в дифференциальной компоненте, а также начнет переполняться интегратор.

Для определения частоты управления существует следующее правило: продолжительность итерации управляющего цикла должна быть между 1/10 и 1/100 желаемого времени стабилизации системы. Например, если мы хотим, чтобы система стабилизировалась за 0.1 с. (за 0.1 секунды тележка должна устойчиво выйти на центр трассы), то частота управления должна быть от 100 до 1000 Гц (время итерации от 0.01 до 0.001 сек.).

Примеры использования ПИД-регулятора

Сам по себе ПИД-регулятор достаточно прост в реализации. К тому же мы теперь знаем, как настраивать его коэффициенты. Основная проблема его использования заключается лишь в том, как научиться его использовать, т.е. включать в управление. ПИД-регулятор работает со входными величинами (сигналом ошибки и значением выходного сгнала), принимающими непрерывные значения, причем как положительные, так и отрицательные. То же касается и формируемого ПИД-регулятором управляющего воздействия. Следовательно, необходимо научиться интерпретировать сигналы от датчиков и управляющий сигнал в терминах этих величин.

Рассмотрим далее ряд примеров использования ПИД-регулятора.

1. Движение вдоль стены

Пусть имеется тележка, которая должна уметь двигаться вдоль стен. Для этого снабдим тележку дальномером. Для того, чтобы маршрут тележки пролегал вдоль стены на некотором расстоянии L, воспользуемся показанием дальномера. Предположим, что на таком расстоянии требуемым показанием дальномера будет некоторая величина Y. Именно показания дальномера и будет регулируемой величиной.

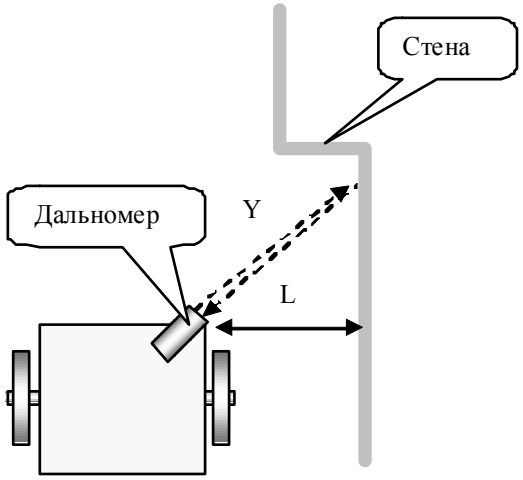


Рис. 16. Движение вдоль стены

Итак, с регулируемой величиной *y(t)* мы разобрались. Сигнал ошибки *e(t)* также не вызывает трудностей – это просто отклонение текущего показания дальномера от требуемой величины Y.

Сложнее с тем, что должен представлять собой сигнал управления *u(t)*, точнее, как нам интерпретировать его значение, формируемое ПИД-регулятором.

Пусть наша тележка использует дифференциальный привод. Это означает, что все повороты мы должны осуществлять, изменяя скорость вращения левого и правого двигателей. Будем интерпретировать сигнал *u(t)* следующим образом: если $u(t)<0$, то будем *притормаживать* левый двигатель на величину, пропорциональную модулю *u(t)*; если $u(t)>0$, то будем *притормаживать* правый двигатель на величину, пропорциональную *u(t)*. Естественно, что если $u(t)=0$, то тележка едет прямо (никакие двигатели не притормаживаются).

2. Движение по полосе. Метод центра масс

В предыдущей задаче у нас была "хорошая" регулируемая величина – показания дальномера. Фактически, мы имели дело с одним датчиком, выдающим одно число – расстояние до стены. С линией дело обстоит сложнее. Для езды по линии надо иметь датчик, позволяющий не только эту линию определить, но и сказать, насколько мы

отклонились от нее. А это значит, что нужно иметь что-то вроде целой линейки датчиков освещенности. И чем больше чувствительных элементов на этой линейке, тем лучше точность измерения отклонения (один или два датчика дадут слишком грубую картину, сводящую на нет все достоинства ПИД-управления).

Датчики. Метод центра масс. Пусть робот оснащен линейкой из n датчиков полосы (a_1, a_2,…, a_n), расположенных на расстояниях x_i от некоторой начальной точки.

Для простоты можно полагать, что датчики расположены равномерно на отрезке.

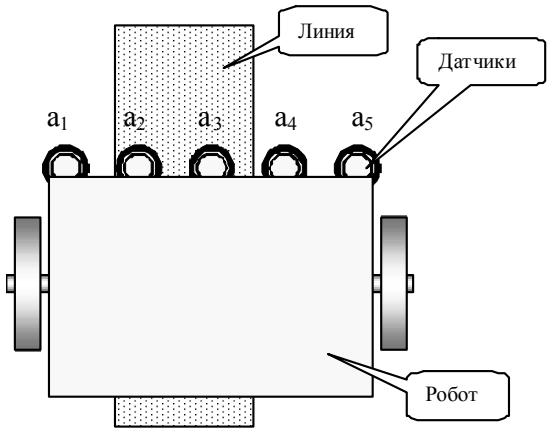


Рис. 17. Линейка датчиков

Если каждый датчик выдает сигнал a_i, соответствующий детектированию искомой полосы, то можно определить положение «центр масс», т.е. величину итогового смещения полосы относительно линейки:

$$x_c = \frac{\sum_{i=1}^{n} x_i \cdot a_i}{\sum a_i}$$

Здесь a_i – аналог массы материальной точки из классической механики.

Величина x_c и будет определять координаты центра определяемой линии относительно линейки.

Регулируемую величину отклонения робота от центра линии y определим из следующих соображений.

Пусть диапазон регулируемой величины y должен лежать в пределах $[y_{min}, y_{max}]$.

Если принять, что текущая координата x центра полосы относительно линейки датчиков находится в диапазоне $[x_{min}, x_{max}]$, то

$$y(x) = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \cdot (y_{\max} - y_{\min}) + y_{\min}$$

Будем считать, что n датчиков равномерно распределены на линейке, причем координата первого датчика равна 0. Кроме того, примем, что значение управляющей величины u должно находиться в диапазоне от -1 до +1 (0 означает, что ошибка равна нулю – мы находимся строго над центром линии).

Тогда для нашего случая y_{min}=-1, y_{max} = 1, x_{min}=0, x_{max} = n–1. Отсюда мы получаем простое соотношение:

$$y(x) = \frac{2x}{n-1} - 1$$

Характерно, что нас не интересуют характер величины сигналов a_i. Более того, не особо критична и равномерность распределения датчиков на линейке. Все это оказывает влияние на работу регулятора (скажем, ПИД-регулятора), на вход которого и будет подаваться сигнал *y*.

Управление. Подав на вход ПИ-регулятора вычисленное выше значение сигнала *y*, мы получим управляющий сигнал *u(t)*. Работать с ним мы будем точно так же, как и в задаче движения вдоль стены – притормаживая двигатели в зависимости от знака величины *u(t)* и ее абсолютного значения.

3. Управление по энкодерам

Рассмотренные выше примеры были в достаточной степени умозрительными, качественными. Далее мы приведем описание более конкретного примера реализации ПИД-управления. Пример будет содержать ряд сугубо технических моментов, которые мы старательно обходили выше, чтобы не погрязнуть в деталях. Теперь же настало время обратить на эти детали внимание.

Итак, пусть имеется система, состоящая из электродвигателя, драйвера двигателя и установленных на двигателе энкодеров. Разумеется, нужен еще сам контроллер. В нашем примере это будет Arduino.

Рассмотрим такую простую задачу, как управление скоростью вращения двигателем, а именно – поддержание некоторой заданной постоянной скорости.

Датчик скорости. Начнем с основного датчика – энкодера. Задача энкодера проста: при вращении двигателя энкодер выдает импульсы. Для этого на вал двигателя устанавливается либо магнитный датчик, либо может использоваться крыльчатка, вращение которой регистрируется оптическим датчиком. Как бы то ни было, на один оборот двигателя энкодер выдает некоторое количество импульсов и, определяя, сколько импульсов пришло за некоторый временной интервал, мы сможем определить угловую скорость вращения.

Подключим выход энкодера к контроллеру, а именно к контакту №3.

```
// Подключение энкодера
#define PIN_encoder   3
```

Для того, чтобы можно было регистрировать и считать импульсы энкодера, используем обработчик прерывания. Эта процедура, которая будет вызываться всякий раз, когда возникнет такое событие, как изменение значение сигнала от датчика энкодера. Для этого напишем саму функцию-обработчик, а в функции инициализации контроллера *setup()* настроимся на это прерывание:

```
// Счетчик
int encoderCnt = 0;

void doEncoder(void)
// Системная функция. Обработчик прерывания для энкодера
{ static byte pred_e = 0;
  byte e = digitalRead(PIN_encoderLeft);
  if(e != pred_e)
  {
    pred_e = e;
    encoderCnt++;
  }
}

void setup()
{
…
  // Настройка энкодеров
  pinMode(PIN_encoder, INPUT);
  digitalWrite(PIN_encoder, HIGH);  // включаем подтяжку входа к «1»
```

```
  // Настройка прерываний для энкодеров
  // Энкодер подключен к ноге №3 – прерывание №1
  attachInterrupt(1, doEncoder, CHANGE);
…
}
```

Как видно, функция-обработчик прерывания занимается только тем, что увеличивает значения счетчика. Этого вполне достаточно. Теперь наша задача состоит в том, чтобы разобраться со временем.

Временные измерения. Нам необходимо не просто считать импульсы, а определять их количество, пришедшее за некоторый временной интервал. Для этого необходима процедура, которая срабатывала бы в строго определенные моменты времени. Речь идет о создании функции обработчика прерывания от таймера. И, разумеется, необходимо настроиться на прерывание от таймера в функции *setup()*.

```
// "Скорость" – количество импульсов, зарегистрированное за 0.02 сек.
int RealSpeed = 0;
void timerIsr()
{
    RealSpeed = encoderCnt;
    encoderCnt = 0;
  }
}

void setup()
{
…
  // Прерывания будут происходить каждые 20'000 микросекунд
  //(0.02 сек., или с частотой 50 Гц)
  Timer1.initialize(20000);
  Timer1.attachInterrupt( timerIsr ); // Настройка обработчика
…
}
```

Теперь в любой момент времени мы можем определить скорость вращения колеса. На самом деле, это – некая условная скорость, т.е. некоторое число импульсов, пришедшее за единицу времени (в нашем примере единица времени – это 0.02 сек.).

Управление. Определимся теперь, как этой скоростью управлять. Будем считать, что мы используем некий драйвер двигателя. Для простоты будем считать, что на некоторый выход контроллера мы будем подавать сигнал управления U. Этот сигнал поступает на вход драйвера двигателя, который, в свою очередь, формирует соответствующее напряжение на выходе V.

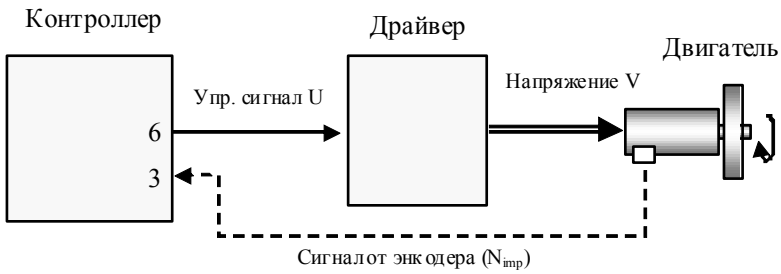


Рис. 18. Управляющие сигналы на выходе контроллера

С программной точки зрения управляющий сигнал представляет собой импульсы определенной скважности и частоты. Управляя скважностью импульсов, мы получаем на

выходе драйвера требуемое напряжение. С программной точки зрения для этого мы используем функцию *analogWrite()*. Второй параметр функции – это величина от 0 до 255.

```
#define PIN_MOTOR  6  //  PWM
…
  analogWrite(PIN_MOTOR, speed);
…
```

Параметры управления. Итак, на данный момент мы можем как управлять скоростью, так и измерять скорость вращения. Важно отметить следующее. В нашей системе для имеющихся двигателей и программных настроек регулируемая скорость находится в диапазоне от 0 до примерно 20 импульсов за единицу времени. Это – значение выходного сигнала. А значение сигнала управления находится в пределах от 0 до 255. Это, разумеется, очень условно. На самом деле двигатель начнет вращаться при достижении лишь некоторого значения напряжения. Это означает, что реальная нижняя граница сигнала управления вовсе не 0, а, скажем, 50.

Запишем эти важные величины.

```
// Диапазон скоростей
float minSpeed =  0;
float maxSpeed = 20;

// Диапазон сигнала управления
float minU = 50;
float maxU = 255;
```

Регулятор. Осталось реализовать сам ПИД-регулятор. С программной точки зрения он почти такой же, как было описано выше.

```
// ПИД-регулятор
struct TPID
{
  float Kp; // Коэффициенты П, И и Д - звеньев
  float Ki;
  float Kd;

  float integral;
  float pred_err;
  float min_integral,
  float max_integral;
  TPID(float Kp0, float Ki0, float Kd0, float mini, float maxi)
  {
    Kp = Kp0;
    Ki = Ki0;
    Kd = Kd0;
    integral = 0;
    pred_err = 0;
    max_integral = maxi;
    min_integral = mini;
  }
  float Eval(float err); // Выход ПИД-регулятора
  float y2u(float y);    // Вычисление управляющего воздействия
};

float TPID::Eval(float err)
{ float y;

  integral = integral + err;  // добавить ошибку в сумму ошибок
  if(integral>max_integral) integral=max_integral;
  if(integral<min_integral) integral=min_integral;
```

```
  float rdiff = Kd*(err - pred_err);
  // вычисление управляющего воздействия
  y = (Kp*err + Ki*integral + rdiff);

  pred_err = err; // текущая ошибка стала "прошлой ошибкой"
  // для след. вычисления
  return y;
}
```

Функция *Eval(float err)*, определяющая выход ПИД-регулятора, уже подробно обсуждалась. На вход подается ошибка, на выходе – некоторая величина *Y*. Однако величина *Y* – это еще не управление. Мы уже говорили, что управление – это определение величины *U*, лежащей в диапазоне от 50 до 255 (от minU до maxU). Следовательно, необходимо преобразовать величину *Y* в *U*. Для этого введена функция *y2u*().

```
float maxErr = maxSpeed;

float TPID::y2u(float y)
{ float u;
  float maxY = Kp*maxErr + Ki*max_integral + Kd*maxErr;
  float minY = -maxY;

  u = (y-minY)/(maxY-minY)*(maxU-minU)+minU;
  if(u>maxU) u = maxU;
  if(u<minU) u = minU;
  return u;
}
```

Функция проста. Мы предполагаем, что между величинами *Y* и *U* существует пропорциональная зависимость. Для вычисления этой пропорции необходимо знать предельные значения *Y* и *U*. Если с *U* мы уже разобрались (*minU* и *maxU*), то оценить диапазон значений, выдаваемый ПИД-регулятором несколько сложнее. Будем считать, что в худшем случае наша ошибка по модулю может достигать величины *maxSpeed* (разность между требуемой и текущей скоростями). Тогда можно оценить минимальное и максимальное значения сигнала *Y* – *maxY и minY* так, как это показано в листинге.

Разумеется, это – весьма грубые оценки, однако на практике такой подход работает достаточно удовлетворительно.

Управляющая программа. Определим где-нибудь (в глобальной области) объект – ПИД-регулятор. Как и положено, начнем эксперименты с того, что не будем использовать интегральное и дифференциальное управления, ограничившись лишь пропорциональным (K_p=0.5, K_i = 0, K_d = 0).

```
TPID pid(0.5, 0, 0, -100.0, 100.0);
```

Будем считать, что наша задача – поддерживать постоянной скорость в 10 импульсов за единицу времени:

```
float goalSpeed = 10;
```

Основная программа (функция *loop()*) будет выглядеть следующим образом:

```
void loop()
{
  // Определение ошибки управления
  error = goalSpeed - RealSpeed;

  // Выход регулятора
  Y = pid.Eval(error);

  // Вычисление управляющего воздействия
  U = pid.y2u(Y);

  // Подача управляющего сигнала на вход объекта управления
  analogWrite(PIN_MOTOR, U);
  // Отладочный вывод реальной скорости
  Serial.print(RealSpeed);
}
```

Эксперименты. Запустив программу мы увидим, что реальная скорость вращения (*RealSpeed*) отличается от требуемой (*GoalSpeed*). Причем иногда достаточно существенно. Мы можем изменять значение *GoalSpeed* и результат будет аналогичным – скорость *RealSpeed* тоже изменится, но рассогласование останется. Более того, при попытках притормозить колесо мы увидим, что регулятор вовсе перестает справляться с задачей.

Проблемы П-регулятора. Увы, это – особенность пропорционального управления. Систематические ошибки (рассогласование) никуда не денутся. Действительно, суть П-управления заключается в том, что система однозначно определяет, каким должен быть сигнал управления при определенном рассогласовании – ошибке, т.е. регулятор «знает» лишь то, как надо ему реагировать в текущий момент времени, не реагируя, вообще говоря, ни на что больше. Например, когда мы притормаживаем колесо, он, разумеется, пытается «исправить ситуацию»: ошибка возрастает, что приводит к увеличению управляющего воздействия. Однако при нашей чудовищной дискретности и грубости управления регулятор с малыми отклонениями уже не справляется. Причина тому, на самом деле, заключается в том, что зависимость между сигналом управления U и скоростью вращения совсем не линейна и наша функция *y2u()* должна выглядеть иначе.

Суть этого можно объяснить так. Предположим, что на входе регулятора имеется некоторая ошибка *err*. П-регулятор отрабатывает ее и, исходя своего коэффициента K_p, а также особенностей преобразования выхода регулятора в управляющую величину, выдает управление U. Однако это значение по ряду причин (помехи, неучтенные факторы и т.п.) не совпадает с тем значением, которое должно быть ($U_{необх}$). Эта разность, т.е. ошибка управления U- $U_{необх}$ является для регулятора неустранимой погрешностью.

Роль интегральной компоненты. Ситуацию значительно улучшает введение *интегральной компоненты*. Установив значение коэффициента интегрального звена K_i, скажем, в 0.2, мы получим уже достаточно адекватное управление. Дело в том, что в такой ситуации начнется накопление ошибки. В итоге управляющий сигнал может оказаться достаточно большим для того, чтобы скорость вращения даже при значительном притормаживании оставалась постоянной. Тем более, устраняются систематические ошибки, описанные выше. При этом, правда, могут возникнуть колебания в контуре управления, но это уже другой вопрос.

Дифференциальная компонента. При известной доле внимания можно заметить, как меняются параметры регулируемой величины при притормаживании колеса. Разумеется, если не полениться вывести эту величину куда-то вовне в виде, пригодном для построения графика, то все будет гораздо строже и красивее. Так или иначе, можно заметить колебания фактического значения скорости, причем иногда весьма

значительные. И в этой ситуации будет полезно ввести ненулевой коэффициент для дифференциальной компоненты K_d.

Усложнение задачи. Может возникнуть вопрос: до этого мы говорили о движении по полосе, об отработке траекторий и прочих аналогичных интересных задачах, а в этом параграфе рассказывалось лишь о примитивной стабилизации скорости вращения. На самом деле, если мы умеем эффективно управлять (а стабилизация – это лишь частный случай управления) одним двигателем, то и все прочие задачи мы также сможем решить. Например, можно завести еще один ПИД-регулятор для второго двигателя, ввести соответствующие аналогичные счетчики и т.п. вспомогательные объекты. А далее, к примру, определить не одну постоянную величину *GoalSpeed*, а задать для каждого двигателя свою функцию от времени: *GoalSpeedLeft(t)* и *GoalSpeedRight(t)*. Эти функции позволят определить, таким образом, произвольную траекторию движения тележки.

Для этого целесообразно перейти от весьма условных единиц измерения скорости к нормальным – в м/с. Здесь необходимо знать параметры энкодеров (количество импульсов на один оборот колеса) и геометрию самой тележки – диаметр колес и расстояние между ними. Но это уже задача для системы управления более высокого уровня.

Замечания. В конце этого раздела важно отметить следующие моменты.

1. Параметры системы. В зависимости от решаемой задачи и, главное, от особенностей «железа» значения коэффициентов ПИД-регулятора, величины квантования сигнала (т.е. частоты срабатывания нашего таймера – процедуры обработки прерывания `timerIsr()`) и проч., следует подбирать заново. Об этом уже говорилось выше, и об этом забывать не стоит.

2. «Хорошие» и «плохие» мотор-редукторы. Хорошо, когда у вас имеется хорошее железо. И плохо в противном случае. Эта банальное замечание относится на самом деле к весьма серьезному вопросу об управляемости вашего объекта управления, т.е. прежде всего мотор-редуктора.

Представьте себе, что мотор-редуктор не умеет вращаться медленно. Т.е. существует некоторое ощутимое пороговое значение подаваемого напряжения, когда колеса начнут вращаться. Эта ситуация характерна для низкооборотистых двигателей с редуктором, у которого малое передаточное число. Особенно явно это проявляется, когда колеса не висят в воздухе, а находятся под нагрузкой. Предположим далее, что мы постепенно начинаем увеличивать управляющее напряжение на двигатель, начиная с некоторого малого значения (это – забота нашего ПИД-регулятора). Начиная с некоторого момента двигатель попытается начать вращаться. Причем весьма неуверенно. И начнется биение («дрожание») ротора, на котором, как мы помним, находится крыльчатка энкодера. Такое «дрожание» крыльчатки приведет к тому, что начнутся ложные срабатывания датчика энкодера. Т.е. фактического движения нет, а система регистрирует импульсы, полагая что колесо вращается.

Разумеется, много зависит от конструкции энкодера, от того, как устроены чувствительные элементы, регистрируем ли мы фазу вращения и проч. Однако такая проблема существует, причем не только для простейших дешевых энкодеров. Бороться с этим можно, установив подходящее значение минимального значения сигнала управления (в наших примерах это была величина `minU`) или минимального значения скорости.

3. Проблема ускорения. Очевидно, что двигатель следует разгонять постепенно. Равно как и тормозить. Дело не только в том, что если мы сразу выставим требуемый сигнал управления (т.е. интересующую нас скорость), то возникнут ударные нагрузки на мотор-редуктор. Дело еще и в том, что, поскольку в начальный момент времени двигатель имеет нулевую скорость, сразу же возникает большой сигнал ошибки, с которым ПИД-регулятор начнет изо всех сил бороться. А это – перерегулирование, колебания, большие управляющие воздействия и прочие слабо контролируемые неприятности. Фактически, при таком «резком» старте мы подаем на вход единичное воздействие (см. Рис. 10). Со

всеми вытекающими. Реакцию на единичное воздействие хорошо изучать в теории, а на практике такое совершенно неприемлемо.

Итак, двигатель следует разгонять постепенно, причем начиная не с нулевой скорости, а, как было сказано в предыдущем замечании 2, с некоторого стартового значения.

Источники

1. Ботов А. Перевод статьи «Просто о ПИД-алгоритмах», roboforum.ru Wiki, URL: http://roboforum.ru/wiki/Перевод_статьи_%22Просто_о_ПИД-алгоритмах%22
2. Дорф Р., Бишоп Р. Современные системы управления. Пер. с англ. Б. И. Копылова. – М.: Лаборатория Базовых Знаний, 2002, -832 с., ISBN: 5-93208-119-8, 0-201-30864-9
3. Лукас В.А. Теория автоматического управления: Учеб. пособие для вузов. -2-е изд., перераб. и доп. –М.: Недра, 1990. -416 с.
4. Первозванский А. А. Курс теории автоматического управления: Учебное пособие для вузов. М.: Наука, 1986. 616 с.
5. Филиппов С.А. Робототехника для детей и родителей. – СПб.: Наука, 2011. -263 с.
6. Wescott Tim «PID Without a PhD» // http://www.embedded.com/2000/0010/0010feat3.htm