Общество с ограниченной ответственностью

**«ГикБреинс»**

Специальность – «Разработчик. Программист. Программист мобильных устройств»

**ДИПЛОМНЫЙ ПРОЕКТ**

на тему «Визуальная демонстрация возможностей ПИД-регулятора при меняющихся внешних условиях на примере летящего самолёта в туннеле»

Коржовой Марии Владимировны

Москва

2024

**ОГЛАВЛЕНИЕ**

|  |  |
| --- | --- |
| Введение……….……………………………………………………………….. | 3 |
| Глава 1. Автоматическое управление моделируемым объектом………….... | 4 |
| 1.1. Основы имитационного моделирования (симуляции) в программировании………………………………………………………. | 4 |
| 1.2. ПИД регулятор - сущность, свойства, применение....……………. | 11 |
| Глава 2. Инструменты для построения модели и регулятора.………………. | 21 |
| 2.1. Принципы объектно-ориентированного программирования как база разработки актуальной программы...……………………………... | 21 |
| 2.2. Знакомство с языком программирования Python.………….......... | 25 |
| 2.3. Выбор структуры данных. Очередь.………………….................... | 28 |
| 2.4. Генерация случайных чисел в программировании………………. | 32 |
| 2.5. Использование бесконечного цикла.……………………...……… | 34 |
| 2.6. Основы системы контроля версий Git……………………………. | 41 |
| Глава 3. Реализация программы ……………………………………………... | 43 |
| 3.1. Описание базовых элементов – модель и регулятор …………….. | 43 |
| 3.2. Формирование изменяющихся внешних условий и графическое отображение результата………………………………………………… | 50 |
| Заключение…...…….…………………………………………………………... | 58 |
| Список использованных источников………………...……….….…………… | 61 |

**ВВЕДЕНИЕ**

Раньше разработчики моделей не могли пользоваться такими благами, какие есть сейчас, достижениями в сфере вычислительной техники, доступности симуляций и моделирования на персональных компьютерах, эффективными вычислительными методами, например, оптимизацией, которые позволяют создавать исследования в сфере процессного управления, системного анализа и проектирования.

В жизни с давних времен люди стремились к удобству и комфорту - они всегда искали способы устранить трудности, упростить свою работу, увеличить ее эффективность и качество, чтобы иметь больше свободного времени на то, что действительно важно для них. Один из ярких примеров такого подхода – внедрение ПИД-регулятора.

Целью данной работы является применение полученных во время обучения знаний на практике и эффективное управление предлагаемой моделью.

Задачи дипломного проекта:

– изучить информацию, связанную с темой исследования;

– выбрать математическое описание моделируемого объекта;

– подобрать коэффициенты ПИД регулятора;

– определить необходимый инструментарий для реализации симуляции;

– разработать программу для автоматического управления выбранной модели.

Дипломный проект состоит из введения, трёх глав, которые разделены на параграфы, заключения и списка использованных источников.

**ГЛАВА 1. АВТОМАТИЧЕСКОЕ УПРАВЛЕНИЕ МОДЕЛИРУЕМЫМ ОБЪЕКТОМ**

* 1. **Основы имитационного моделирования (симуляции) в программировании**

Моделирование – это сложный процесс, включающий в себя создание и функционирование модели. Модель, в свою очередь, является аналогом реальной системы и призвана помочь аналитикам предсказать последствия изменений в системе. Основная цель моделирования заключается в создании точной модели, которая отражает все основные свойства системы. Таким образом, моделирование – это акт создания такой модели.

Модель представляет упрощенное отображение реального объекта или явления, позволяющее изучить его свойства и взаимодействия. Важно, чтобы модель была более информативной, доступной для изучения, чем реальный объект, сохраняла первоначальную информацию и обладала способностью к получению новой. Модель – способ изучения объекта.

Модель является информативной, если выполняет некоторые требования, такие, как:

1. наглядность и открытость основных аспектов реального объекта и моделирования;
2. её можно исследовать и воспроизводить;
3. это делать достаточно просто;
4. она сохраняет оригинальную информацию и может получать новую.

Однако для использования результатов моделирования в отношении реального объекта модель должна быть адекватной, то есть точно отражать свойства и характеристики реального объекта. Замена реального объекта моделью позволяет проще, дешевле и безопаснее исследовать его свойства, выделяя наиболее значимые аспекты и игнорируя второстепенные.

Естественно, модель любого реального объекта или явления не может быть абсолютно точной, в полной мере отражая все его свойства и особенности поведения. Однако хорошо построенная модель способна передать общую картину и основные характеристики системы. Благодаря модели можно изучить структуру объекта, его взаимодействие с окружающей средой, научиться управлять и предсказывать поведение объекта при изменяющихся условиях.

То есть модель нужна при:

1. исследовании структуры реального объекта, его характеристик и законов взаимодействия с окружающей средой.
2. Она обучает управлению этим явлением или объектом.
3. Способна предсказывать его поведение и состояние при условиях, которые постоянно меняются, и в различных ситуациях.

Модели могут быть абстрактными и материальными по способу построения. Абстрактная модель существует только в нашем воображении и может быть описана на естественном или формализованном языке. Примером абстрактных моделей являются алгоритмы, представленные в виде блок-схемы. Материальные модели представляют собой макеты или устройства, имитирующие свойства и функции изучаемого объекта. Например, уменьшенные модели автомобиля или корабля.

Модели также могут быть вербальными, математическими и компьютерными. Вербальные модели являются утверждением, описывающим объект или явление на естественном или формализованном языке. Математические модели представляют собой систему математических операторов и уравнений. Компьютерные модели являются программами, имитирующими поведение системы, которую изучают. Они позволяют проанализировать и воспроизвести реальные объекты и процессы при воздействии на них различных факторов.

Компьютерная модель позволяет анализировать поведение системы с точки зрения времени или пространства. Она может использоваться для решения задач анализа и синтеза сложной системы, обучения и тестирования, представления работы системы в графическом виде, а также для проведения действий, невозможных в реальности.

Моделирование системы (имитация) – это использование модели для изучения производительности существующей или предлагаемой системы. Симуляция системы – это процесс, при котором модель используется для анализа производительности системы. Другими словами, симуляция – это имитация системы, которая позволяет проанализировать ее работу в рамках времени или пространства.

Компьютерная модель – это одна программа либо комплекс программ, позволяющих с помощью расчетов и графического отображения результатов воссоздать реальные объекты и процессы при воздействии различных условий и факторов. Подобные модели называются имитационными.

Компьютерное моделирование – метод осуществления анализа или синтеза сложных систем, основанный на исследовании компьютерных моделей таких сложных систем.

Использование компьютерной модели в качестве обучающего механизма предоставляет несколько возможностей:

1. для изучения сложных явлений и процессов,
2. фокусироваться на свойствах системы посредством гибкого представления и мультимедийных эффектов,
3. осуществлять мониторинг динамики посредством анализа динамики,
4. представления работы системы в визуальной форме, такой как графика, диаграммы, схемы,
5. достижение невозможного в реальной жизни из-за пространственно-временных рамок или страхов за безопасность окружающей среды или самого объекта.

Классификация компьютерных моделей может основываться на следующих моделях:

1. Модели оптимизации полезны при определении оптимальных средств работы со сложной системой и управления ею.
2. Прогностические модели делают предположения о состоянии объекта в определенные будущие моменты.
3. Дескриптивные модели, которые описывают изучаемый объект и факторы, оказывающие влияние на его поведение.
4. Игровые модели генерируют несуществующие ситуации, которые имитируют реальность.
5. Учебные модели, которые используются в визуальной подготовке для студентов.

Компьютерное моделирование включает в себя данные ключевые этапы:

* 1. Сначала необходимо поставить задачу, описав объект и определив цель моделирования. На данной стадии требуется выяснить цель создания модели, определить необходимые исходные данные и предвидеть ожидаемые результаты.
  2. После этого происходит построение информационной модели. На данной стадии есть необходимость определить параметры модели и отметить взаимосвязи между ними. Оценить важность каждого параметра для данной задачи и определить, можно ли пренебречь некоторыми из них. Далее требуется математически описать данные зависимости между вышеуказанными параметрами модели.
  3. Следующий этап – разработка методов и алгоритма построения компьютерной модели. На данной стадии происходит выбор или разработка метода получения первоначальных результатов, составление порядка получения результатов с использованием выбранных методов и проверить алгоритм на правильность.
  4. После этого приходит время для разработки компьютерной модели. На данной стадии происходит выбор средств реализации алгоритма на программном уровне на компьютере, а также разработка самой компьютерной модели и проверка точности реализованной компьютерной модели.
  5. Затем следует проведение эксперимента, для чего происходит разработка плана изучения и проведение эксперимента на основе реализованной компьютерной модели, а полученные результаты анализируются. В итоге делаются соответствующие выводы.

Компьютерной симуляцией называют программы, которые дают возможность потребителям коммуницировать с компьютерным образом научной модели в сфере природного или физического мира[[1]](#footnote-1).

Таким образом, компьютерные модели являются мощным инструментом для изучения сложных систем, предоставляя возможность анализировать и предсказывать их поведение и управлять ими. Они позволяют визуализировать физические процессы и взаимодействовать с ними в «реальной среде».

С помощью компьютерного моделирования возможно наблюдать физический процесс, который происходит на экране компьютера в "реальной среде". Изменение параметров взаимосвязанных объектов, свойств окружающей среды и способов взаимодействий между объектами позволяет изучать явление в разных условиях. Важным преимуществом такого моделирования является возможность симуляции условий, которые невозможно воссоздать в реальной лаборатории, таких как отсутствие сопротивления воздуха, трения или гравитации.

Моделирование позволяет отражать информацию в динамике более точно, чем диаграммы, и даёт возможность создавать визуализации различных явлений. Оно даёт возможность заметить то, что сложно увидеть в реальной жизни, то, что протекает с очень маленькой или большой скоростью, или скрыто.

Компьютерное имитационное моделирование даёт возможность проектировать, разрабатывать и оценивать сложные системы. Разработчики, инженеры, аналитики применяют его для понимания и оценки сценариев, отражающих ответы на вопрос: «Что, если?». С помощью него возможно реализовывать как реальные системы, так и предлагаемые системы с использованием компьютерных технологий, и оно оказывается полезным, когда преобразования в реально существующей системе трудно выполнить, для них нужны большие затраты или их выполнение непрактично. Среди примеров компьютерного моделирования, привычные для многих: прогноз погоды и симуляторы, которые применяются в обучении пилотов.

Имитационные модели включают в себя показатели производительности, системные объекты, функциональные отношения, входные переменные. Чтобы разработать имитационную модель, нужно:

1. конкретизировать проблему, заключающуюся в реально существующем объекте, или определить требования к создаваемому объекту,
2. ещё раз определить проблему, основываясь на имеющихся свойствах и ограничениях,
3. собрать данные системы,
4. спроектировать модель с помощью сетевых диаграмм, проверить её на правильность,
5. сравнить производительность созданной модели в разных условиях с существующим объектом,
6. написать документ модели для последующего использования, отображающий все свойства и способы применения модели в деталях,
7. определить необходимый и возможный способ проверки и опытов,
8. создать условия для опытов, начать эксперименты и получать результаты.

Другими этапами анализа моделирования также являются:

1. определение проблемы,
2. выбор входных данных, разработка объектов для осуществления моделирования. Рассмотрим два типа входных переменных: переменные решения и неуправляемые переменные. Первые поддаются контролю программистами, а неуправляемые являются случайными данными,
3. определение рамок для переменных решения,
4. нахождение выходных переменных,
5. подготовка данных из реально существующего объекта для внедрения в симуляцию,
6. создание блок-схемы, отражающей алгоритм моделирования,
7. определение соответствующих программных технологий для моделирования,
8. анализ имитационной модели путем сравнения ее результатов с реально существующим объектом,
9. осуществление испытаний на модели, при необходимости замена значений переменных для нахождения наилучшего решения,
10. применение полученных результатов в реальной системе.

К преимуществам применения моделирования и симуляции можно отнести:

1. лёгкость понимания, показывает, как работает объект без обращения к реальному объекту,
2. простота тестирования, даёт возможность преобразовывать систему, выявлять влияние изменений, при этом не обращаясь к реальному объекту,
3. лёгкость обновления,
4. простота идентификации ограничений,
5. простота диагностики проблем.

Моделирование позволяет понять взаимодействия в сложных системах, так как не все системы возможно изучить за один раз, а также исследовать новые процессы, процедуры, не затрагивая реальные объекты.

Недостатками использования моделирования и симуляции являются:

1. трудность предсказания результатов операций, так как используются случайные числа,
2. требование знаний и опыта в области проектирования моделей,
3. трудоёмкость процесса моделирования, необходима рабочая сила,
4. стоимость процедур моделирования,
5. трудность перевода результатов моделирования, для этих целей необходимы эксперты.

Кроме того, существуют различные типы имитационных моделей:

1. дискретные модели, в которых преобразования в объекте происходят в точное время;
2. непрерывные модели, в которых условия объекта непрерывно меняется с течением времени;
3. смешанные модели, включающие и дискретные, и непрерывные компоненты.

Проектирование модели – это сложный процесс, требующий глубоких знаний в соответствующей области, обучения и многолетнего опыта. В ходе операций, проводимых в объекте с применением случайных чисел, предсказать результат становится очень сложно. Если результаты, составленные при изучении информационной модели, не совпадают с измеряемыми параметрами реально существующего объекта, делают заключение о том, что на предыдущих стадиях создания модели были ошибки или неточности.

* 1. **ПИД регулятор - сущность, свойства, применение**

Вопросы регулирования и управления знакомы всем, кто занимался автоматизацией процессов. При попытке автоматизировать любой процесс, в первую очередь необходимо понять технологию этого процесса, представить, как всё происходит и взаимосвязано. Лишь после этого можно создавать технологическую схему и внедрять регуляторы.

Существует множество способов регулирования, но один из самых лёгких, заслуживающих доверия и зарекомендовавших себя на протяжении длительного времени – это ПИД-регулятор. Отчего выбор делается в пользу ПИД, а не П, ПД или ПИ, ведь данные регуляторы ещё легче и доступнее для пользователя? Ответ заключается в том, что есть возможность в любой момент из ПИД сделать любые комбинации из этих трех режимов. Кроме того, большее количество нынешних регуляторов разработаны программно, и их обработка осуществляется с помощью микропроцессора, а не с использованием операционных усилителей, как это происходило ранее. При этом использование операционных усилителей также интересно.

Компания Taylor, которая занимается созданием приборов, в 1939 году преподнесла новую версию пневматического контроллера Fulscope, который добавил предварительное действие к уже существующим режимам сброса и пропорционального управления. Сегодня эти три режима называются пропорциональным, интегральным и дифференциальным, или в сокращении аббревиатура ПИД, где интегральный означает сброс, а дифференциальный – предварительное действие. В российской литературе используется аббревиатура ПИД, объединяющая все три режима. Далее в этом же 1939 году компания Foxboro Instrument внесла в свой контроллер Stabilog, который имел режимы пропорциональности и сброса, режим гиперсброса, тем самым создавая второй ПИД-регулятор на мировом рынке. В нынешнее время все ПИД-регуляторы созданы с пропорциональным, интегральным и дифференциальным режимами, которые были представлены в 1939 году, как на рисунке 1.

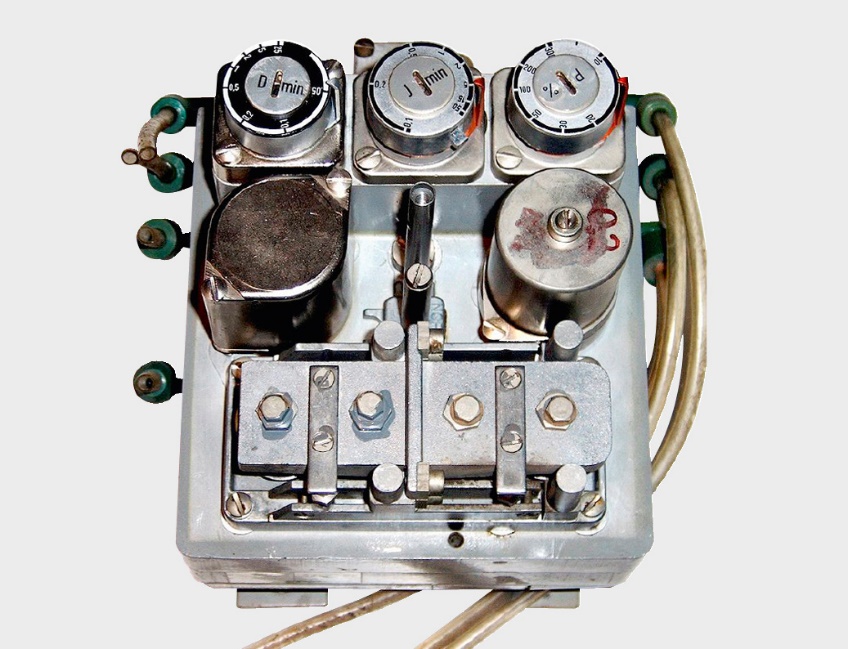


Рис. 1. Пневматический ПИД-контроллер[[2]](#footnote-2)

ПИД-регуляторы применяют в большом количестве промышленных приложений, основанных на автоматическом управлении процедурами. Они способны настраивать температуру, давление, количество и другие параметры процедур производства.

Пропорционально-интегрально-дифференцирующим (ПИД) регулятором называют механизм с обратной связью, который применяется в объектах автоматического управления при создании управляющего сигнала, который обеспечивает требуемые точность и качество переходного процесса. ПИД-регулятор создаёт управляющий сигнал, который состоит из трёх компонентов:

1. первый компонент пропорционален разности входного сигнала и выходного сигнала, также данный сигнал называют сигналом обратной связи, а их разность входного и выходного сигналов является сигналом рассогласования,
2. второй компонент равен интегралу данной разности,
3. третий компонент равен производной данной разности.

ПИД-регуляторы обладают тремя режимами управления:

1. пропорциональный,
2. интегральный,
3. дифференциальный.

Каждый из этих режимов откликается по-разному на появление ошибки. Изменение параметров регулятора позволяют настроить величину и характер отклика, исходящего от каждого режима управления.

При этом могут использоваться не все элементы данного регулятора, он может состоять из двух или одного компонента, тогда будет иметь название пропорциональный, пропорционально-дифференцирующий, пропорционально-интегрирующий и так далее.

Главным двигателем в контроллере является режим пропорционального управления, который изменяет выходной сигнал контроллера пропорционально ошибке, как на рисунке 2.

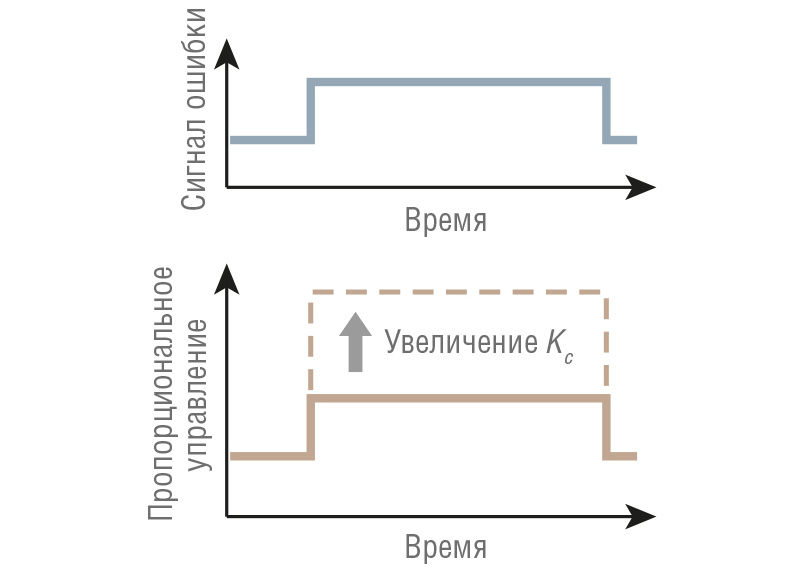


Рис. 2. Действие пропорционального компонента управления

При возрастании ошибки, управляющее воздействие тоже возрастает пропорционально. Такое свойство удобно, так как для устранения больших ошибок необходимо более интенсивное воздействие. Регулируемый параметр контроллера, определяющий масштаб пропорционального управления, имеет название усиление контроллера (Controller Gain). Более сильный коэффициент усиления повысит пропорциональное управляющее действие для определённый ошибки. В том случае, если усиление контроллера определено чрезмерно высоким, контур управления может стать нестабильным и начать колебаться. Если усиление контроллера чрезмерно низкое, он может начать неуместно и неправильно откликаться на преобразования параметров управления.

В большом количестве регуляторов преобразование коэффициента усиления также отражается на размере отклика в интегральном и дифференциальном режимах. Это объясняет название этой настройки – усиление контроллера. При этом существуют и такие контроллеры, где настройка пропорционального усиления не отражается на других режимах. Это конструкция контроллера, известная как алгоритм параллельного, или независимого усиления.

Если необходимо, ПИД-регулятор может быть настроен только на пропорциональное воздействие, при этом отключаются интегральный и дифференциальный режимы. Пропорциональные контроллеры просты в использовании и настройке. Выходной сигнал пропорционального контроллера представляет собой ошибку управления, которая умножается на коэффициент усиления контроллера, и прибавляется смещение, как на рисунке 3.

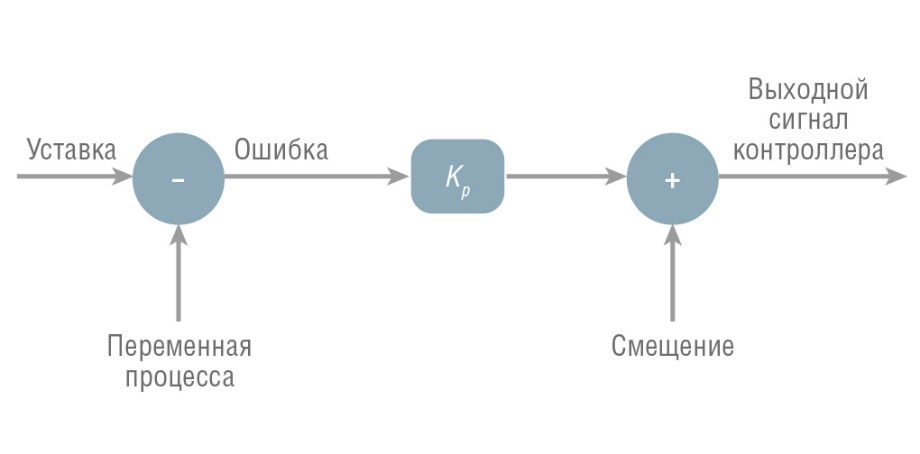


Рис. 3. Пропорциональное управление

Выходной сигнал, создаваемый пропорциональным компонентом, предотвращает отклонения регулируемой величины от контрольного значения, которые отмечаются в течение определенного периода времени. При увеличении отклонения, увеличивается выходной сигнал. В ситуации, в которой входной сигнал равен контрольному значению, выходной равен нулю.

При этом при применении лишь пропорционального регулятора величина регулируемой величины никогда не станет равна заданному значению. Имеет место ошибка, известная как статическая ошибка, которая соответствует отклонению регулируемой величины, предоставляющему выходной сигнал, сохраняющий выходную величину точно на этом уровне. Для примера регулятор температуры обычно заставляет выходной сигнал (мощность нагревателя) замедляться по мере приближения температуры к заданной температуре, что в конечном итоге приводит к возвращению системы к мощности, которая наблюдается в состоянии тепловых потерь. А температура не имеет возможности дойти до заданного значения, потому что в такой ситуации выходной сигнал будет равен нулю, и регулятор остынет.

Статическая ошибка тем меньше, чем выше коэффициент пропорциональности между входным и выходным сигналом, также его называют коэффициентом усиления. Но если коэффициент усиления чрезмерно высокий и имеются задержки, или запаздывание, в объекте начнутся автоматические колебания. И если и далее повышать коэффициент усиления, система утратит стабильность, прочность.

Данное отклонение при пропорциональном режиме будет держаться до того момента, пока разработчики не изменят выходной сигнал регулятора вручную, чтобы избавиться от этого отклонения. Таким образом разработчик вручную сбрасывает регулятор, или контроллер.

Интегрирующая компонент пропорционален интегралу по времени от отклонения регулируемой величины. Он применяется для избавления от статической ошибки. Данный компонент позволяет контроллеру с течением времени предусмотреть статическую ошибку.

Потребность устранить ручной сброс по вышеуказанной схеме легла в основу создания автоматического сброса, на данный момент его ещё называют режимом встроенного управления. Суть режима встроенного управления состоит в повышении или понижении выходного сигнала регулятора через определённое время с целью снизить имеющуюся ошибку, которая заключается в нахождении переменной процесса не в заданном диапазоне величин. Интегральный механизм меняет выходной сигнал контроллера до тех пор, пока ошибка не достигнет нуля, но это происходит в ситуации, когда интегральный механизм имеет достаточное количество времени. Скорость изменения выходного сигнала зависит от величины ошибки, то есть в условиях большой ошибки интегральный механизм изменяет выходной сигнал контроллера с высокой скоростью и в условиях малой ошибки делает это медленнее.

Для устранения указанной проблемы необходимо настроить скорость интегрального действия путем изменения интегральной настройки времени контроллера, то есть Ti, что означает Integral Time, или интегральное время. При большом значении Ti (длительное время интегрирования) интегральное действие будет происходить медленно, а при малом значении Ti (короткое время интегрирования) – быстро, как на рисунке 4.

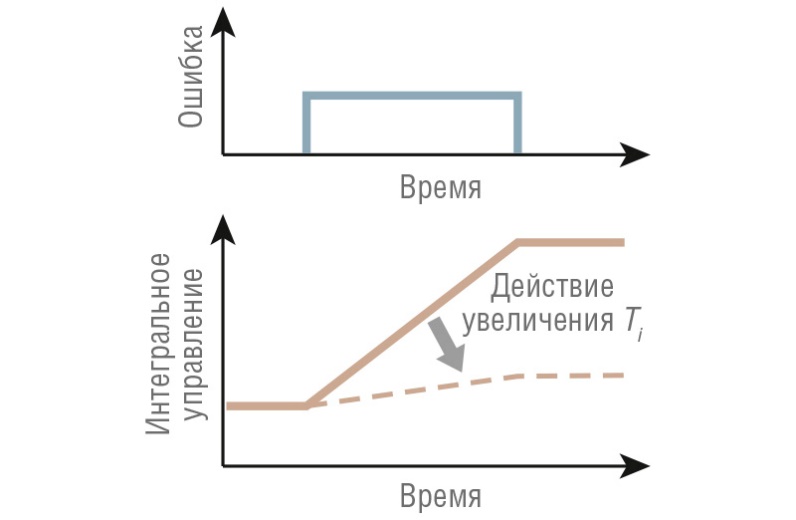


Рис. 4. Действие интегральной составляющей управления

В тех случаях, когда интервал времени интегрирования является чрезмерно большим, реакция контроллера будет медленной; в тех случаях, когда он является чрезмерно коротким, контур управления будет находиться в состоянии колебаний и не обретёт стабильное состояние. Большая часть контроллеров, в роли единицы измерения для интегральной составляющей управления применяет интегральное время в минутах, при этом немногие контроллеры устанавливают его в секундах. Некоторые контроллеры, чаще те, что используют параллельный алгоритм, применяют параметр «интегральное усиление», или Ki, в повторениях в минуту.

В тех случаях, когда система не подвергается внешним воздействиям, со временем контролируемая величина балансируется на заданном значении, при этом сигнал пропорциональной составляющей управления будет равен нулю, а выходной сигнал будет определяться в полной мере интегрирующей составляющей. Тем не менее, интегрирующая составляющая также может приводить к автоколебаниям при неправильном выборе её коэффициента.

Чаще всего пропорционально-интегральный контроллер имеет название ПИ‑контроллер, его выходной сигнал составляет сумму пропорциональных и интегральных воздействий управления, как на рисунке 5.

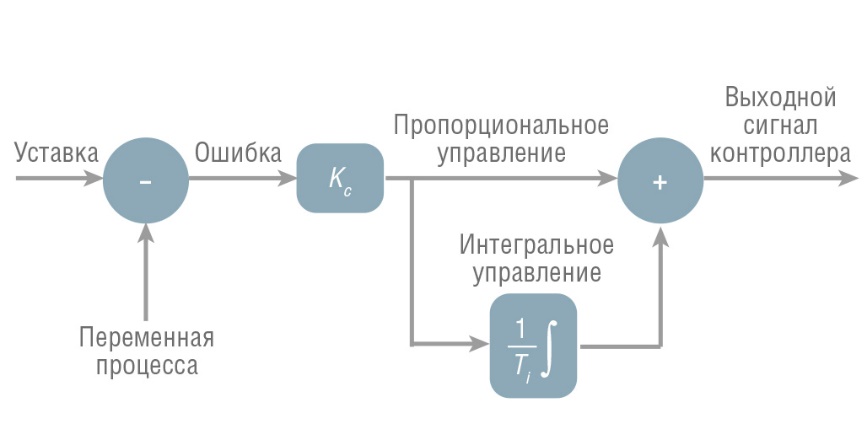


Рис. 5. Алгоритм ПИ-регулирования

В ПИД-контроллере третьим режимом управление является режим управления по производной. Дифференциальный режим нечасто применяется в управлении процессами, но при этом находит широкое применение в управлении движением.

В целях управления процессом дифференциальный режим необязателен, так он чувствителен к шумам и усложняет настройку методом проб и ошибок. Однако, его применение использование может повысить скорость реакции определенных типов управляющих контуров по сравнению с применением только ПИ-регулирования. К примеру, для ПИД-регуляторов типичным является температурное управление. Дифференциальный режим, или режим управления по производной даёт возможность отслеживать сигнал, который отражает скорость изменения ошибки, как на рисунке 6.

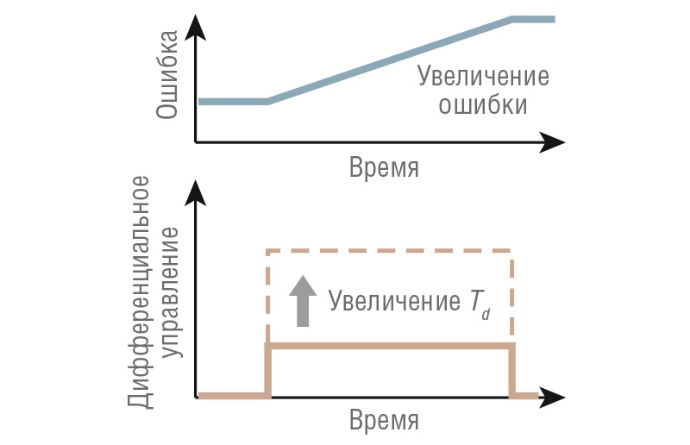


Рис. 6. Процесс дифференциального регулирования

Составляющая дифференциального режима пропорциональна скорости преобразования отклонения регулируемой величины и создана для борьбы с отклонениями от желаемого значения, которые могут появиться в будущем. Отклонения могут обуславливаться внешними воздействиями или задержкой воздействия регулятора на объект.

Именно поэтому режим дифференциального управления первоначально имел название управление по скорости.

Дифференциальный режим производит большее управляющее действие, в тех случаях, когда ошибка преобразуется с большей скоростью. В ситуации, когда величина ошибки не преобразуется с течением времени, действие составляющей дифференциального режима будет равно нулю. Для настройки дифференциальной составляющей используется параметр, называемый временем дифференцирования Td, или Derivative Time. Чем выше это значение, тем больший вклад вносит дифференциальный режим управления. Если время Td определено как ноль, это отключает данный режим в полной мере. При определении времени слишком большим появляются колебания и контур управления становится несбалансированным, нестабильным. Составляющая дифференциального режима управления контроллера настраивается с помощью двух единиц измерения, а именно минут и секунд.

Пропорционально-интегрально-дифференциальный контроллер, или ПИД-регулятор, функционирует, основываясь на сумме пропорциональных, интегральных и дифференциальных воздействий управления.

На рисунке 7 представлен алгоритм неинтерактивного, или идеального ПИД-регулятора, а на рисунке 8 показан алгоритм параллельного контроллера.

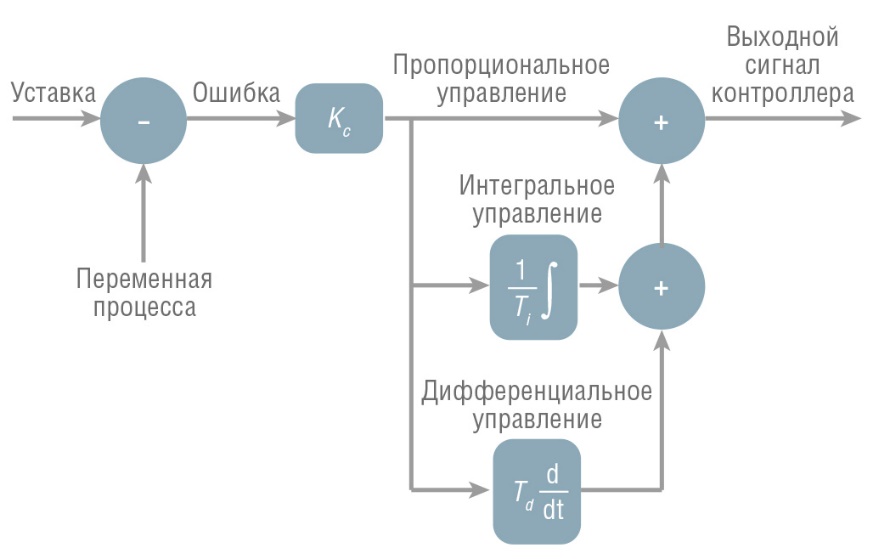


Рис. 7. Неинтерактивный алгоритм PID-управления

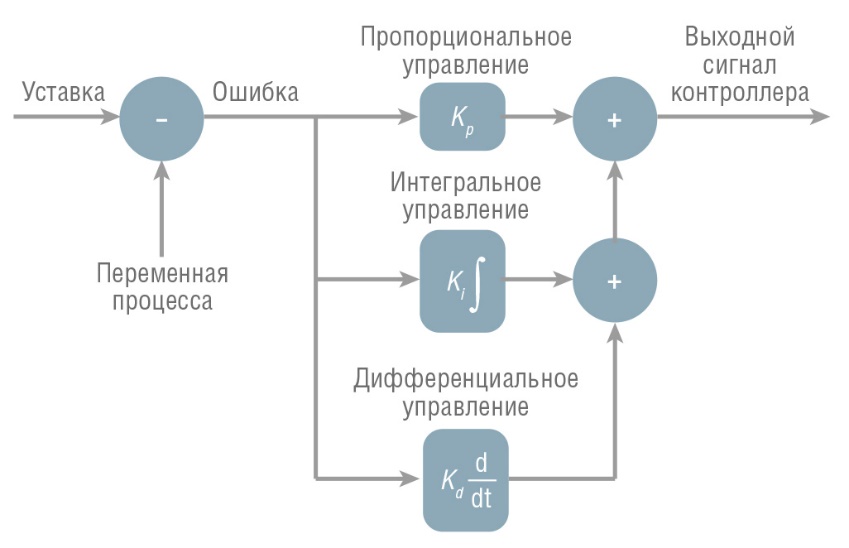


Рис. 8. Параллельный алгоритм PID-управления

Дифференциальный режим ПИД-контроллера осуществляет более интенсивное управление в начальный период времени раньше, насколько это возможно при использовании только пропорционального или пропорционально-интегрального режимов. Это помогает снизить влияние воздействий и уменьшает время, применяемое для возврата уровня к заданному значению.

**ГЛАВА 2. ИНСТРУМЕНТЫ ДЛЯ ПОСТРОЕНИЯ МОДЕЛИ И РЕГУЛЯТОРА**

* 1. **Принципы объектно-ориентированного программирования как база разработки актуальной программы**

Объектно-ориентированное программирование, или ООП, представляет собой методологию программирования, которая базируется на использовании объектов, в состав которых входят данные, или атрибуты, и функции, или методы, функционирующие с вышеуказанными данными. ООП - сильная методология программирования, применение которой способно создавать более модульный, повторно используемый и поддерживаемый код.

Объектно-ориентированное программирование, или ООП, можно назвать стилем программирования, который базируется на описании типов, моделей предметной области и их взаимосвязях. Они представлены порождением из прототипов или как экземпляры классов, составляющих иерархию наследования.

С точки зрения идеологии, объектно-ориентированное программирование является подходом к моделированию информационных объектов, который даёт возможность на высоком уровне абстракции решать важную задачу структурного программирования, а именно, структурирование информации для более удобной управляемости. Он обеспечивает эффективное управление процессом моделирования и разработку крупных программных проектов.

Управляемость для иерархических систем означает сокращение избыточности данных и обеспечение их целостности. Это обеспечивает создание удобно управляемых систем, которые легко понимаются.

То есть решение тактического вопроса управляемости даёт возможность решать и стратегическую задачу, а именно, передавать осознание задачи программистом в форму, которая в наибольшей степени удобна для дальнейшего применения.

В центр объектно-ориентированного программирования помещено понятие «объект». Объект представляет собой сущность, принимающая сообщения и реагирующая на них, применяя свои данные. Объект – является экземпляром класса. Класс представляет собой универсальный, комплексный тип данных, в состав которого входит единый набор полей и методов. Класс моделирует информационную сущность и предоставляет внутренний и внешний интерфейсы для работы с его содержимым.

Процедуры и функции, которые относятся к классу, описывают действия, которые объект вправе выполнять и которые можно осуществлять над объектом данного вида. Классы также могут наследоваться друг от друга, что позволяет классу-потомку получить все поля и методы класса-родителя, а также добавить свои собственные или переопределить уже существующие.

Объектно-ориентированное программирование нацелено на разработку больших программных комплексов, которые выполняются командой разработчиков. Проектирование объекта, создание отдельных элементов и их объединение в конечный продукт при этом нередко осуществляется разными программистами.

Основополагающие правила в области структурирования объектно-ориентированного программирования взаимосвязаны с разными сторонами фундаментального осознания предметной задачи, требующей оптимального управления моделью:

1. Инкапсуляция является свойством объекта, которое даёт возможность объединять в один класс данные и методы, в которых применяются эти данные. Она позволяет скрывать внутреннюю реализацию объекта и предоставлять лишь общедоступный интерфейс, применяемый для контакта с объектом. Данное свойство даёт возможность создавать защиту целостности системы и облегчает поддержку и преобразование кода, которое необходимо со временем. Также данное свойство помогает более быстрой и безопасной организации иерархического управления, то есть по команде, отвечающей на вопрос «что выполнять?», программа не ждёт конкретизации, как выполнять.
2. Следующий принцип – абстракция. Она относится к практике представления сложных объектов или понятий с помощью более простых и общих концепций. Абстракция даёт возможность концентрироваться на основополагающих свойствах системы и пренебрегать деталями, не относящимися к актуальной задаче. Абстракцию используют, если имеют цель выделить в моделируемом объекте важное для решения поставленной задачи. Под абстрагированием понимается обозначение важной информации и игнорирование незначимых данных. Объектно-ориентированное программирование сосредоточено на абстракции данных, которую часто называют просто «абстракцией». Это предполагает определение наиболее важных свойств объекта, доступных остальной программе.
3. Одним из свойств системы объектно-ориентированного программирования является наследование, которое позволяет создавать новый класс, основываясь на уже имеющемся с частично или полностью заимствованными свойствами, функциями.

Класс, от которого происходит наследование, именуется базовым, или суперклассом, или родительским. Новый класс называется потомком, или дочерним, или производным классом, или наследником. Наследование имеет отношение к области создания новых классов, перенимающих свойства и методы имеющихся классов. Оно даёт возможность повторно применять код и разрабатывать более специализированные классы, которые заимствуют полностью или частично свойства и функции более общих классов. Это упрощает и безопасно организует родственные понятия, позволяя учитывать только преобразования на каждом иерархическом уровне, не дублируя всю остальную функциональность, реализованную на предыдущих уровнях.

1. Полиморфизм – это свойство объекта, которое позволяет применять объекты с идентичным интерфейсом, не зная их тип и внутреннюю структуру. Это возможность одинаково обрабатывать объекты с разным исполнением, если они имеют общий интерфейс. То есть применяется единый интерфейс для реализации нескольких видов объектов. Полиморфизм даёт возможность разрабатывать более общий код, который при этом может работать с различными объектами, не зная отдельных, точных свойств и элементов каждого объекта. Полиморфизм помогает определить, где лучше распараллелить единое управление или, наоборот, объединить его.

В Python возможно применять принципы объектно-ориентированного программирования для написания классов и объектов, инкапсулирующие данные и функции. Классы обозначаются при помощи определённого слова «class», а создание экземпляров данных классов происходит с помощью имени класса, после которого указываются круглые скобки. К примеру:

class People:

def \_\_init\_\_(self, name, height, weight):

self.name = name

self.height = height

self.weight = weight

def HumanParametres(self):

print(f’Hi, my name is {self.name}, my height is {self.height } centimeters and my weight is {self.weight } kilograms.’)

people1 = People (‘Kate’, 180, 75)

people2 = People (‘Sasha’, 150, 60)

people1.HumanParametres()

# prints «Hi, my name is Kate, my height is 180 centimeters and my weight is 75 kilograms.»

people2.HumanParametres()

# prints «Hi, my name is Sasha, my height is 150 centimeters and my weight is 60 kilograms.»

В данном примере создаётся класс «People», имеющий три свойства (атрибута), а именно «name», «height», «weight» и одну функцию, или метод, «HumanParametres». Далее были созданы два экземпляра класса «People»: «people1» и « people2» и вызвана функция «HumanParametres» для каждого из этих двух экземпляров.

В целом, принципы ООП являются важным инструментом для создания более модульного, повторно используемого и поддерживаемого кода. Понимание применения объектно-ориентированного программирования позволяет повысить навыки программирования, и разрабатывать более весомые, приспосабливающиеся и эффективные программы.

* 1. **Знакомство с языком программирования Python**

Программированием называется процедура разработки программ, для которой программисты пишут исходный код на каком-либо языке программирования.

На данный момент присутствует более 300 языков программирования, но около десяти являются самыми популярными.

Python – один из самых популярных языков программирования, он характерен простотой в изучении и применении. Разработчики отдают предпочтение Python по разным причинам:

1. из-за высокой эффективности,
2. из-за простоты изучения,
3. из-за возможности использовать его на различных платформах.

Есть возможность скачивания программы на языке Python бесплатно, они совместимы со всеми типами систем и способствуют ускорению разработки.

Python относится к полностью объектно-ориентированным языкам, это значит, что всё в коде является объектами. Неординарной особенностью языка можно назвать обязательное использование отступов для обозначения блоков кода.

Основные положения программирования, которые используются в Python, включают в себя:

1. Синтаксис языка Python лаконичен, что позволяет не часто на практике использовать документацию. В Python происходит использование отступов для обозначения блоков кода и применение конкретных ключевых слов, например: for, while, if, else, это существует для того, чтобы управление потоком программы было наиболее эффективным.

2. Python – это язык программирования, который позволяет нам работать с различными типами данных и использовать различные структуры данных для организации и управления ими. Присутствуют различные типы данных, такие как числа, строки и списки. Кроме того, мы можем использовать структуры данных, такие как словари и множества. В Python есть и примитивные типы данных, такие как булев тип, которое называется «bool», целые числа неопределённой длины, которые называются «int» и «long», комплексные числа под названием «complex», числа с плавающей запятой, имеющие название «float».

Также в данном языке имеются контейнерные типы данных, к ним можно отнести:

1. строки,
2. списки,
3. словари,
4. множества,
5. кортежи.

При этом все понятия называются объектами, включая функции, классы. Новые типы данных можно определять путем написания классов, они обозначаются при помощи определённого слова «class», или определения нового типа в модуле расширения. Создание структуры классов обеспечивает возможность использования наследования. Почти все встроенные типы данных и типы расширений поддерживают наследование.

3. Системы управления и ветвления применяются в случаях, когда необходимо осуществлять управление потоком программы. В Python применяются цикл «while» для повторения части кода, до тех пор, пока установленное условие реализуется, цикл «for» для прохождения по последовательности элементов, операторы «if» и «else» для реализации конкретного кода, только если исполняется конкретное условие. Условный оператор «if» может иметь несколько вариантов выполнения с помощью блока «elif», расшифровывается как «else if». То есть если в коде имеется не одно условие, то может использоваться необязательный блок «elif», его можно применять неограниченное количество раз. Если ни одно условие не выполняется, то реализуется необязательный блок «else», то есть «иначе».

4. Python первоначально создавался не как язык функциональной разработки, но тем не менее данный язык обеспечивает функциональный стиль программирования. В Python функции являются объектами первого класса, что означает, что они могут быть присвоены переменной, переданы в качестве аргумента другой функции или возвращены из функции. Кроме того, Python поддерживает функции высших порядков, которые являются функциями, в состав которых входят другие функции как аргументы, или которые возвращают функции как результат. Еще одной функциональной возможностью языка является рекурсия, то есть возможность функции вызывать саму себя.

В Python функции представляют собой повторно используемые части кода, реализующие конкретные задачи. В данном языке есть возможность создавать свои собственные функции, применяя перед названием ключевое слово «def», при этом можно применять и встроенные функции, например, «append», это функция, служащая для добавления элемента в список.

5. Еще одной важной особенностью Python является объектно-ориентированное программирование (ООП). В данной парадигме программирования отдельное внимание уделяется применению объектов для целей представления данных и поведения. В Python есть возможность создавать свои собственные классы и объекты, обеспечивающие организацию кода и создание более сложные программ. Вся концепция языка Python опирается на объектно-ориентированную модель программирования. Применение объектно-ориентированного программирования в данном языке хорошо продумано, но при этом достаточно специфично в сопоставлении с иными языками, поддерживающими объектно-ориентированную модель. В Python все сущности кода называются объектами.

В контексте Python классы являются описанием объекта, включающим его характеристики и методы. Классы представляют собой план или описание для создания объекта и содержат в себе описание характеристик объекта и методов для работы с ним. Основные принципы ООП, на которых базируется парадигма, в Python включают инкапсуляцию, наследование и полиморфизм. Инкапсуляция в Python реализуется с помощью способности хранения публичных и скрытых характеристик, свойств, а также методов для работы с ними. Хотя на самом деле все свойства в Python являются публичными, существует соглашение об именовании для обозначения скрытых атрибутов. Наследование и полиморфизм – два важнейших принципа, позволяющие определять производные объекты и изменять методы объекта соответственно, без потребности повторной разработки кода.

Все эти концепции программирования применимы к языку Python и позволяют разрабатывать более гибкие и масштабируемые программы.

* 1. **Выбор структуры данных. Очередь**

Однако, помимо функционального и объектно-ориентированного стиля программирования, в Python также существуют различные структуры данных, которые играют важную роль при разработке программ. К ним относятся линейные структуры данных, такие как массив, динамический массив и связанный список. Линейными они являются, так как элементы в них имеют свой порядок и не меняют его сами по себе.

Массивы имеют высокую скорость при произвольном доступе и обладают хорошей производительность при добавлении и удалении элементов с конца. Однако простые массивы ограничены по размеру и не позволяют динамически изменять свой размер. Для этого существуют динамические массивы. Связанный список представляет собой структуру данных, хорошо подходящую для частого добавления и удаления элементов из середины.

Если вам требуется фиксированное количество элементов, то массив является подходящим вариантом. Однако, предположим, что необходимо добавить ещё один объект в массив. Для простого массива данное действие не является возможным. Также, предположим, что необходимо удалить элемент из массива. Для простого массива данное действие также не является возможным. То есть существует привязка к определенному количеству элементов. В таком случае требуется массив, среди функций которого есть возможность менять его размер.

Существуют линейные структуры данных с ограниченными концами: стеки и очереди. Оба они работают похожим образом, как аналоги в реальном мире. Например, стек, будь то стопка тарелок или стек данных, позволяет «положить» что-то сверху, данная функция имеет название «push», а также достигнуть верхнего элемента и «вынуть» данный объект данная функция имеет название «pop». Очередь, подобно очереди людей, работает по принципу добавления в конец и удаления из начала.

Кроме линейных структур данных, существуют также нелинейные структуры данных, такие как словарь данных, упорядоченные и неупорядоченные множества. Такие структуры не являются линейные, что значит, последовательность элементов, которую добавляют в них, не будет такой же при получении их обратно. Словарь данных работает аналогично бумажному словарю, где каждому ключу, то есть искомому слову, соответствует определение, или значение. Упорядоченное множество содержит только ключи, отсортированные по определенному порядку. Неупорядоченное множество представляет собой «мешок» с объектами, порядок которых не имеет значения. Вышеуказанные структуры данных являются хорошо подходящими для быстрого поиска.

Зачастую разработчикам необходимо итеративно обрабатывать коллекции данных, и в таких случаях выбор структуры данных не играет особой роли, так как в какой последовательности находятся элементы, не важно, отсчёт начинается с первого элемента и заканчивается на последнем в структуре данных.

Но в ситуациях, когда важно, в какой последовательности находятся элементы в структуре данных, динамический массив является наиболее предпочтительным выбором, так как он может меняться до любого размера и может легко замещаться на другие структуры данных. Однако, в некоторых случаях особенности конкретной задачи могут потребовать использования других структур данных.

Например, в играх одной из самых распространённых задач можно назвать поиск пути, то есть нужно добраться из одной точки в другую. Для этих целей широко используется следующий алгоритм: в нём применяется структура данных, которая содержит не весь путь, а частичные пути, отсортированные так, что максимально возможный частичный путь будет располагаться в передней части контейнера. Далее происходит оценка данного пути, и в том случае, если путь ещё не окончен, алгоритм разделяет его на несколько частичных путей большего размера и добавляет их в структуру данных.

Применение динамического массива в роли контейнера будет не оптимальным выбором по различным причинам:

* 1. Процедура удаления объектов из начала динамического массива является одной из самых медленных.
  2. Осуществление повторной сортировки данного массива после каждого добавления тоже является медленной операцией.

Таким образом, лучше выбрать наиболее подходящую структуру данных, о которой говорилось ранее. Она подходит для такого типа доступа и может быть использована вместо динамического массива. В данной структуре удаляются элементы с начала и добавляем их в конец, а автоматическая сортировка происходит на основе лучшего пути. Это очередь с приоритетом, встроенная в язык и обеспечивающая удобную отладку. В зависимости от нужд применения выбор структуры данных может варьироваться.

Если представить, что человек стоит в очереди. Первый человек получает то, для чего он стоял, после этого покидает очередь. Затем второй человек тоже получает то, для чего он стоял, после этого покидает очередь. Все остальные люди подходят к очереди и становятся в ее конец. Именно это и является основной концепцией структуры данных под названием «очередь».

Очередь представляет собой структуру данных с принципом FIFO, или First In First Out, что означает: «первый зашел, первый и вышел». Она представляет собой очередь, в которую объекты добавляются в конец, а извлекаются из начала. Очередь имеет несколько основных операций: Front, Pop\_Front, Back, Push\_Back. Front обеспечивает доступ к первому объекту в очереди, Pop\_front извлекает объект из начала очереди, Back обеспечивает доступ к последнему объекту в очереди, Push\_Back добавляет объект в конец очереди.

Если у разработчика есть необходимость включать или исключать элементы с обоих концов очереди, тогда может быть использована двухсторонняя очередь, её также называют «deque». В такой очереди также доступны операции Push\_Front и Pop\_Back. Двухсторонние очереди являются частью основных языков программирования, включая Python.

Очередь и двухсторонняя очередь могут быть реализованы весьма сложно. Так как элементы могут добавляться и удаляться с обоих концов, внутренний контейнер должен быть способен нарастить или укоротить очередь как с начала, так и с конца. В большинстве созданных контейнеров используются множественные страницы памяти. Когда один из концов превышает текущую страницу, создаётся новая страница. В ситуации, когда в странице нет необходимости, она уничтожается. Например, в языке Java для связанного списка используется немного дополнительной памяти, а не страниц памяти, но такая концепция также успешно работает.

Очередь является не необходимой структурой данных, но её применяют во многих случаях. Она оптимальна, когда требуется иметь дело только с конечными элементами данных, а серединные элементы не играют важной роли.

* 1. **Генерация случайных чисел в программировании**

В процессе разработки на Python часто возникает потребность создания случайных чисел. Случайные числа используются в различных областях программирования, например, для создания моделей и видеоигр. Для определения, что такое случайная последовательность, стоит сказать, что это набор объектов, каждый из которых никак не связан с остальными. Однако, в программировании последовательность всё же часто не является строго случайной, а зависит от предыдущих значений.

Совершенно случайные числа генерируются истинными генераторами случайных чисел, такими как бросание кубика или доставание шаров из пакета. В компьютере таких устройств нет, поэтому генерируются только «псевдослучайные» числа. В Python, как и во многих других языках программирования, применяется генератор псевдослучайных чисел, который выдает числа, которые кажутся случайными, но по сути являются воспроизводимыми. Для создания случайных чисел существует формула, поэтому их сложно назвать полностью случайными, но при этом данные числа удобны для многих программ. В Python для этих целей применяется встроенная библиотека random, включающая в себя функции для создания случайных чисел. В Python библиотека random содержит генератор псевдослучайных чисел для различных типов данных, например, вещественных чисел с плавающей запятой.

Для начала работы с библиотекой random в программе нужно его импортировать. Для этого в самом начале кода прописывается оператор библиотеки import random. Далее можно использовать функции модуля для генерации случайных чисел различных типов. Например, функция random() возвращает случайное число с плавающей запятой в промежутке от 0.0 до 1.0, данная функция является базовой в данной библиотеке.

Пример применения библиотеки и функции random:

import random

print(«Получение случайного числа с плавающей запятой в промежутке от 0.0 до 1.0 с помощью функции random.random()»)

print(random.random())

Для получения случайного целого числа в заданном промежутке чисел нужно применять функцию randint(), принимающую два аргумента: начало и конец промежутка. Функция возвращает случайное целое число в этом промежутке. Следующий пример отражает, как создать случайно сгенерированное число в диапазоне от 0 до 11. В данном примере для начала импортируется модуль random, а далее применяется функция randint() для создания случайного числа от 0 до 11. В последней строчке число отражают в консоли.

import random

print(«Получение случайного целого числа в промежутке от 0 до 11», random.randint(0, 11))

Еще один пример создания случайного целого числа – применение функции randrange(). Данная функция принимает на вход два аргумента: начало и конец промежутка, и осуществляет возврат случайного числа в этом промежутке.

Пример применения функции randrange() для создания случайного целого числа от 0 до 11:

import random

print((«Получение случайного целого числа в промежутке от 0 до 11»,random.randrange(0, 11))

При этом в промежутке функции randrange() конец промежутка не включается. Что значит, в примере выше функция создаст случайное число от 0 до 11, не включая 11.

В теории вероятностей особое внимание уделяется понятию распределения вероятностей, которое играет важную роль. Оно дает информацию о вероятности наступления каждого из допустимых событий. Распределение вероятностей позволяет моделировать не только дискретные события (подбрасывание монеты, число человек в автобусе), но и непрерывные (объем потребления воды в течение месяца). В рамках моего проекта для этой цели будет использоваться функция random().

* 1. **Использование бесконечного цикла**

Исполнение кода в Python, как правило, следует принципу последовательности: сначала выполняется первая строка, затем вторая, третья и так далее. Однако, в Python есть конструкции, дающие возможность изменять этот порядок и выполнять более сложные операции. Примером таких конструкций являются циклы, которые позволяют выполнить один и тот же блок кода несколько раз. В Python существует два основных вида циклов, это while и for.

Цикл в любом случае состоит из двух обязательных элементов:

1. Условие цикла – это начальный параметр, при выполнении которого цикл запускается и продолжается, пока оно остается истинным.

2. Тело цикла – это фактически программа, выполняемая внутри цикла.

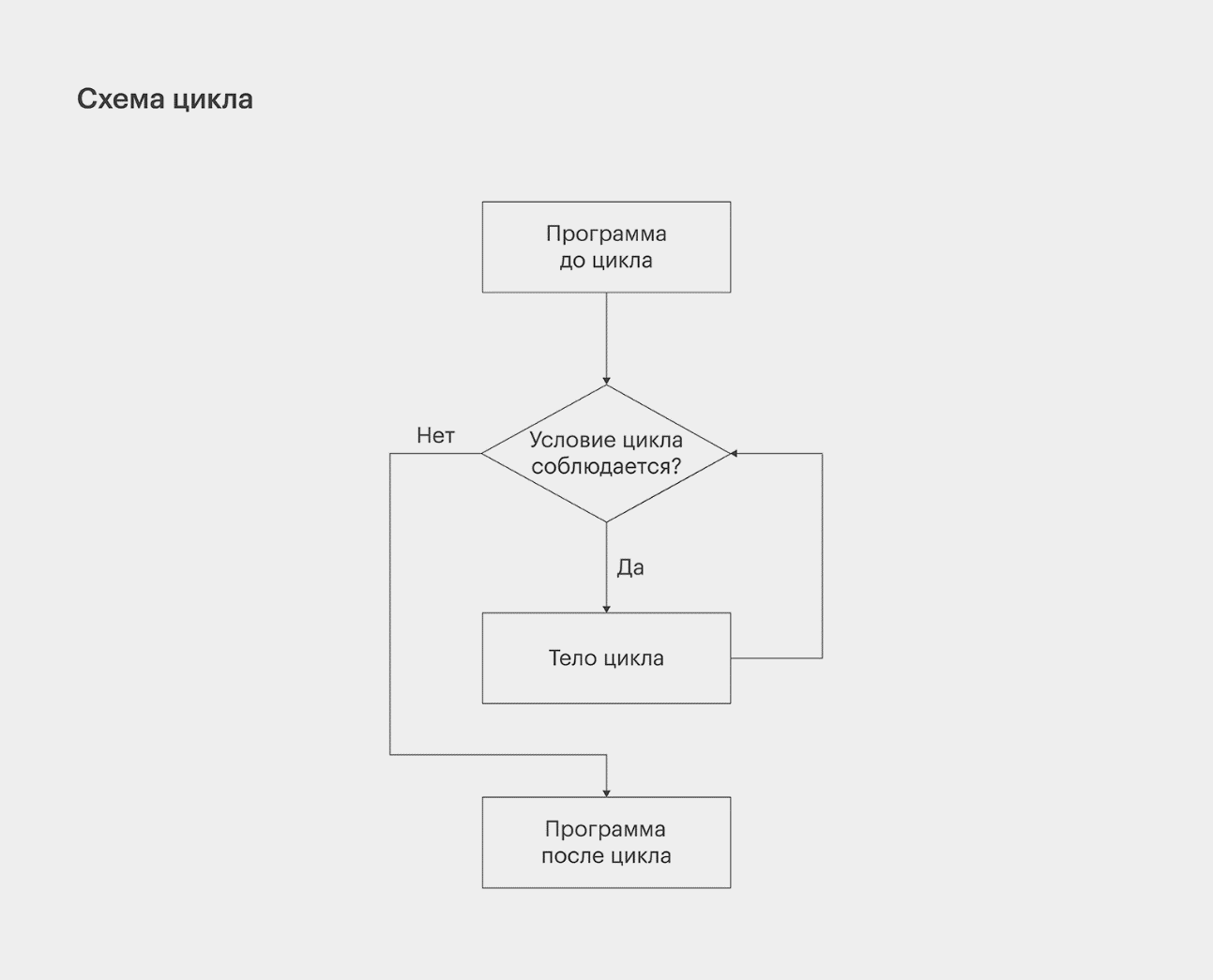


Рис. 9. Схема цикла

В синтаксисе языка Python, конец строки с условием обычно отмечается двоеточием, а весь блок кода, относящийся к условию, выделяется отступом. Цикл while представляет собой самый простой и понятный вид цикла, который также известен как цикл с предусловием. В Python цикл while применяется для многократного исполнения определенных действий до тех пор, пока заданное условие является верным. Данный механизм позволяет программе последовательно перебирать определенный блок кода. В начале цикла программа проверяет условие, определенное в операторе while. Если условие верно, то запускается цикл, и выполняется тело кода, указанное после while. Это тело будет выполняться до тех пор, пока условие является верным. Когда условие становится неверным, программа выходит из цикла и заканчивает выполнение тела кода. К примеру:

x = 2

while x < 7:

print(x)

x += 1

Несмотря на то, что цикл while прост, в данной простоте есть опасность создать бесконечный цикл. К примеру:

x = 2

while x < 7:

print(x)

В данном коде происходит присваивание переменной x значения 2, однако никаких дальнейших действий с ней не выполняется. Условие цикла while всегда будет истинным, так как оно зависит от значения переменной x, которая установлена на 2. Следовательно, цикл никогда не закончится.

Для того, чтобы не допустить такую ситуацию, при применении цикла while необходимо уделять внимание тому, чтобы условие в конечном итоге перестало выполняться. Дополнительно можно использовать оператор break для прерывания выполнения цикла внутри блока кода.

В отличие от цикла while, цикл for является более популярным среди программистов. Он позволяет итерироваться по некоторому набору данных, такому как список, строка, диапазон или любой другой итерируемый объект. В отличие от чистого условия, в цикле for устанавливается некий массив данных, по которому итератор будет перемещаться на каждой итерации.

Вывод: для избежания бесконечного выполнения цикла следует внимательно следить за условием в цикле while и, при необходимости, использовать оператор break. В свою очередь, цикл for широко применяется в программировании, позволяя легко итерироваться по различным наборам данных.

На каждой итерации цикла программа проверяет, остались ли в объекте ещё элементы, которые не были обработаны.

К примеру, существует список чисел: [10, 15, -8, 3]. Можно применить к данному списку цикл for для выведения каждого объекта отдельно. При каждой итерации цикла переменная num будет обновляться и хранить в себе очередной элемент списка. Сначала она содержит первый объект, затем второй, и так далее, пока список не будет полностью пройден.

numList = [10, 15, -8, 3]

for num in numList:

print(num)

Если необходимо использовать цикл for к диапазону чисел, можно применять функцию range(). Эта функция может принимать от одного до трёх аргументов. Если передается только один аргумент, то будет сформирован диапазон от нуля до значения, предшествующего данному аргументу.

for i in range(3):

print(i)

Если в функцию передано два аргумента, то она создаст диапазон значений, начиная с первого аргумента и заканчивая числом, предшествующим второму аргументу.

for i in range(31, 36):

print(i)

Если же функции передано три аргумента, то первые два аргумента будут работать аналогично предыдущему случаю. А третий аргумент будет определять шаг, с которым числа будут идти друг за другом.

for i in range(0, 10, 3):

print(i)

В некоторых ситуациях может возникнуть необходимость досрочно прервать выполнение цикла, даже если его условие все еще верно. В таких ситуациях применяется оператор break.

Например, создадим строку «Привет, цикл!» и будем выводить в консоль каждый символ этой строки. Если при итерации по строке встретится запятая, то цикл будет завершен досрочно. Этот пример представлен ниже:

stringHi = «Привет, цикл!»

for i in stringHi:

if i == ',':

break

print(i)

Если в строке запятая будет отсутствовать, цикл пройдет по каждому символу строки и только после этого закончится.

Иногда в работе с циклами возникает необходимость обойти определенные действия и перейти к следующему шагу без их выполнения. Для таких ситуаций в программировании применяется оператор continue. Рассмотрим пример: существуют числа от 6 до 30 включительно и выведем в консоль только те, которые не делятся ни на 4, ни на 5.

for i in range(6, 30):

if i%4 == 0 or i%5 == 0:

continue

print(i)

Если условие if истинно, то есть число без остатка делится на 4 или на 5, то остальная часть кода не выполняется, и значение переменной i не выводится в консоль.

Как правило, оператор else используется вместе с if, но оно также может применяться с while или for. В данной ситуации код в блоке else выполнится после завершения всех итераций цикла.

Если цикл преждевременно прерывается с помощью оператора break, то код в блоке else не выполнится.

Вернёмся к ранее упомянутому коду со строкой «Привет, цикл!» и добавим в него else.

stringHi = «Привет, цикл!»

for i in stringHi:

if i == ',':

break

print(i)

else:

print(«В тексте нет запятых»)

В предыдущей версии программы, где была запятая в строке, сработал оператор break и условие в блоке else не выполнилось. Если убрать запятую запятую из строки, выполнится условие в блоке else.

Все итерации цикла успешно выполнятся, что приведёт к выполнению кода, находящегося в блоке else. Этот блок будет исполнен даже в случае, если цикл не совершил ни одной итерации.

Есть ситуации, когда применение бесконечного цикла может быть полезным решением. Допустим, при написании игры, которая должна работать до тех пор, пока игрок не завершит игровой процесс. В таком случае должно быть указано условие для завершения цикла с помощью оператора break. Чтобы сделать цикл бесконечным, условие его выполнения должно всегда оставаться истинным. Существует несколько способов прописать такое условие, например, с помощью конструкции while true.

В Python возможно вложение циклов друг в друга, что означает вызов одного цикла внутри другого. Такое вложение усложняет логику программы и требует более внимательного подхода к анализу процесса выполнения.

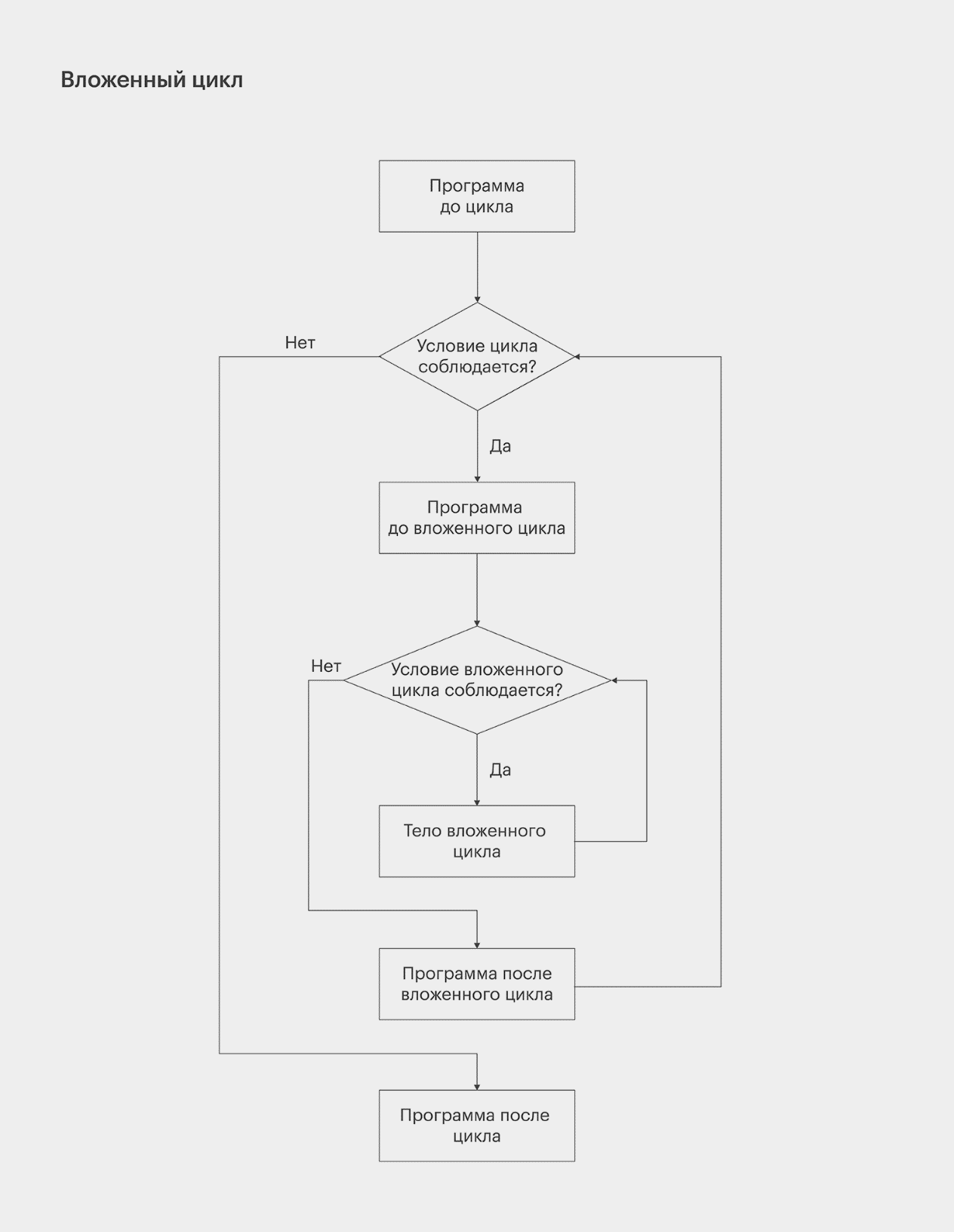


Рис. 10. Схема вложенного цикла

Циклы в программировании очень важны, составляют большую его часть. Несмотря на то, что чаще всего использование бесконечного цикла может быть ошибочно и опасно, в моей программе он будет использоваться, так как необходима практически безостановочная длительность цикла.

* 1. **Основы системы контроля версий Git**

Контроль версий, или контроль исходного кода – это практика, дающая возможность наблюдать за изменениями в исходном коде и управлять ими. Он особенно полезен, когда необходимо хранить разные версии проекта и иметь возможность переключаться между ними. Чтобы выполнить хранение версий, используются копии информации, созданные на компьютере или сервере. Возможность восстановления предыдущих версий реализуется путем замены текущей информации сохраненными версиями. Система контроля версий предлагает функцию замены информации с применением сохраненных вариантов.

Система управления версиями – это программа, сохраняющая историю изменений в документе или файле. Владелец системы управления версиями имеет возможность доступа и возврата к предыдущим версиям, сделанным любым участником проекта. Это особенно важно, когда нужно восстановить старую версию файла или сравнить две различные версии документа. К тому же она даёт возможность наблюдать изменения, сделанные в документе или файле, увидеть, кто их сделал, и помогает предотвратить конфликты при одновременном изменении одного и того же файла.

Git – наиболее известная и широко используемая система контроля версий, однако она не является единственной. Git – это распределенная система для отслеживания изменений в файлах и директориях проекта, позволяющая программистам управлять и мониторить редактирования в проекте. Программа Git берет на себя всю работу по контролю версий проекта и позволяет осуществлять переключение между ними. Стоит отметить, что Git хранит не целые файлы, а только различия между ними, что экономит память. Автором программы является Линус Торвальдс, создатель операционной системы Linux.

Для работы с Git сначала необходимо сделать инициализацию репозитория в папке проекта. После инициализации можно включать изменения в репозиторий, создавать разные версии проекта. История изменений доступна для просмотра, и есть возможность возврата к любой предыдущей версии. Ещё есть функция объединения проектов нескольких программистов в один.

Все команды вводятся с помощью кода в терминале. Инициализация осуществляется командой «git init», указывается папка, в которой git будет мониторить изменения. При инициализации появляется скрытая папка .git. Команда «git add» добавляет изменения в индекс для последующего коммита. Эту команду следует использовать после добавления файлов.

Команда «git commit» имеет функцию фиксации или сохранения изменений. «git commit» применяет только индекс, поэтому можно использовать «git add» для создания следующего коммита.

С помощью команды «git commit» выполняется сохранение всех данных, которые включены в индекс при использовании команды «git add», во внутренней базе данных. Этот процесс также приводит к сдвигу указателя текущей ветки на сохраненный слепок данных.

Большая часть современных сред разработки предоставляют интуитивно понятный интерфейс для работы с системой контроля версий Git, поэтому нет необходимости запоминать все команды.

**ГЛАВА 3. РЕАЛИЗАЦИЯ ПРОГРАММЫ**

**3.1. Описание базовых элементов – модель и регулятор**

В данном проекте были использованы важные принципы объектно-ориентированного программирования: инкапсуляция, абстракция, наследование.

Инкапсуляция важна, так как является свойством объекта, которое даёт возможность объединять в один класс данные и методы, в которых применяются эти данные. Она позволяет скрывать внутреннюю реализацию объекта и предоставлять лишь общедоступный интерфейс, применяемый для контакта с объектом. Данное свойство даёт возможность создавать защиту целостности системы и облегчает поддержку и преобразование кода, которое необходимо со временем.

Абстракция также занимает важное место в программировании. Она относится к практике представления сложных объектов или понятий с помощью более простых и общих концепций. Абстракция даёт возможность концентрироваться на основополагающих свойствах системы и пренебрегать деталями, не относящимися к актуальной задаче. Под абстрагированием понимается обозначение важной информации и игнорирование незначимых данных. Это предполагает определение наиболее важных свойств объекта, доступных остальной программе.

Ещё одним из свойств системы объектно-ориентированного программирования является наследование, которое позволяет создавать новый класс, основываясь на уже имеющемся с частично или полностью заимствованными свойствами, функциями. Наследование имеет отношение к области создания новых классов, перенимающих свойства и методы имеющихся классов. Оно даёт возможность повторно применять код и разрабатывать более специализированные классы, которые заимствуют полностью или частично свойства и функции более общих классов. Это упрощает и безопасно организует родственные понятия, позволяя учитывать только преобразования на каждом иерархическом уровне, не дублируя всю остальную функциональность, реализованную на предыдущих уровнях.

Показателем применения инкапсуляции в проекте является то, что было создано 5 классов: «Data», «FlyingModel», «PIDRegulator», «Processing», «Graphic» и 5 отдельных файлов с одноименными названиями, а также шестой файл под названием «Main», в котором сведены все эти классы.

Примером использования абстракции является то, что в ходе написания классов был выделен ряд основополагающих функций и переменных, наиболее ясно и полно описывающих реализованную в классе абстракцию. Одновременно характеристики, которые не были существенными для разрабатываемой программы, не принимались во внимание и, соответственно, не реализовывались.

Применение принципа наследования позволило установить удобную и, что не менее важно, понятную пользователю взаимосвязь между классами. Так, класс «Data» является родительским классом для всех остальных классов. Он содержит все переменные, необходимые для работы программы. Таким образом, классы-наследники могут использовать все данные из класса «Data» без повторной инициализации. Также удобством такой реализации является то, что все данные, расположенные в классе «Data», можно менять единожды без необходимости искать эти переменные в других файлах, так как все они находятся в одном месте.

Целью созданной программы является демонстрация применения регулятора для управления объектом. В качестве регулятора был выбран наиболее распространенный и часто используемый ПИД-регулятор. Его применяют в самых разных областях науки, техники, промышленности, а также DIY. В качестве объекта управления использовалась модель самолета. В разработанной программе ПИД-регулятор осуществляет автоматическое управление летящим в туннеле самолетом. При вылете самолета за границы туннеля программа завершается. Полёт реализован в виде бесконечного цикла, если не наступят условия выхода из цикла, полёт не завершится никогда. Работоспособность и эффективность разработанного процесса регулирования показана с помощью изменяющихся внешних условий – изменение со временем размеров туннеля, по которому летит самолет. В целом, созданная программа похожа на бесконечную аркаду (жанр компьютерных игр).

Программа разделена на 6 файлов – 5 классов и исполняемый файл Main.

Класс «Data». Он содержит все переменные программы. Данный класс помимо принципов ООП создан для удобства пользователя. При необходимости изменения какого-либо параметра, относится ли он к ПИД-регулятору, к модели самолета или к отображению графика полета, его не придётся искать в разных местах, почти все они находятся в одном файле. Класс импортирует библиотеку random для генерации случайных чисел и класс deque из библиотеки collections для создания массивов данных в форме очередей. Описать все приведенные в классе «Data» переменные лучше с учетом того, к какому классу/объекту/процессу они относятся:

1. универсальная переменная времени (timeStep). Она относится ко всему процессу симуляции полёта и используется как в классе регулятора для вычисления интегральной и дифференциальной составляющих сигнала управления, так и в классе модели самолета для вычисления скорости и позиции самолета по первой и второй производной по ускорению соответственно. Данная переменная задаёт время дискретизации. Непрерывный с первого взгляда процесс полёта на самом деле разбит на множество частей, время между двумя соседними частями задаётся переменной timeStep. На графике такое разбиение отображается точками, каждая новая точка появляется через время timeStep. Если говорить с точки зрения кода, timeStep – время в секундах между итерациями бесконечного цикла. В программе данной переменной присвоено значение 0.02, при таком временном шаге полёт на графике визуально кажется достаточно плавным, не скачкообразным.
2. Переменные, относящиеся к графику полёта. graphicWindowSize – размер графика по оси X, он принят за 100. Размер графика по оси Y задан в классе «Graphic» и задаётся диапазоном значений по Y, принят за [-1, 101]. Такой диапазон по Y был выбран для того, чтобы точки с граничными абсциссами, это значения 0 и 100, были заметны на графике. Переменная pointMovedForward отвечает за то, как сильно самолёт на графике будет выдвинут вперед относительно левой границы графика. Для удобного отображения pointMovedForward установлено значение 10. Все вышеперечисленные тут переменные имеют размерность так называемых точек и являются постоянными, со временем они не изменяются. Некоторое пояснение: в разработанной программе линия на графике строится по точкам, которые между собой соединены последовательно; каждая точка имеет свои координаты по оси X и оси Y; соответственно для задания линии надо указать два массива: первый – это координаты точек линии по оси X, второй – это координаты точек линии по оси Y.
3. Переменные, относящиеся к изменяющимся внешним условиям – размерам туннеля. tunnelLenght задаёт начальную длину (вдоль оси X на графике) туннеля. Со временем полёт усложняется, и длина туннелей становится меньше. tunnelHeightMin – минимальная высота (вдоль оси Y на графике) туннеля. Фактическая высота туннеля – это случайное число. Но для того, чтобы случайно сгенерированная высота не была слишком маленькой или не равнялась 0 (иначе самолёт «физически» не поместится в данный туннель), накладывается ограничение на минимальное значение высоты. Со временем полёт усложняется, и минимальная высота туннелей также становится меньше. аcceleration – параметр усложнения полёта. Он задаёт количество шагов/итераций, через которое произойдёт усложнение условий полёта. reductionStep – параметр, отвечающий за то, на сколько уменьшится длина и минимальная высота туннеля при усложнении условий полета. lowTunnelEgde и upTunnelEgde – нижняя и верхняя граница туннеля по Y. Они определяются случайным образом с учётом наложенных ограничений. Границы туннеля обновятся, то есть появится новый туннель, как только пройдёт tunnelLenght точек полёта. Значения вышеуказанных переменных, кроме случайных, можно увидеть в коде программы.
4. Переменные, относящиеся к модели самолета. mass – масса модели самолета в килограммах. gravity – ускорение свободного падения в м/с2. startPosition и startSpeed - начальная позиция (по оси Y) и скорость модели самолета, данные переменные на каждом шаге полёта обновляются. Значения приведенных в данном абзаце переменных можно увидеть в коде программы.
5. Переменные, относящиеся к ПИД-регулятору. proportionalCoefficient, integralCoefficient, derivativeCoefficient – пропорциональный, интегральный и дифференциальный коэффициенты ПИД-регулятора. integralSum – интегральная сумма, необходимая для вычисления интегральной составляющей управляющего сигнала. previousError – значение ошибки на предыдущем шаге, необходимое для вычисления дифференциальной составляющей. Вышеуказанные переменные относятся к классическому набору переменных для реализации ПИД-регулятора. Они были подобраны эмпирически, с учетом конкретного объекта регулирования – использующейся в программе модели самолета с определенными параметрами. Однако для корректной работы авторегулирования потребовалось введение дополнительного параметра averageLimit – расстояние, на которое установка для ПИД-регулятора не должна приближаться к краю туннеля. Значения вышеуказанных переменных можно увидеть в коде программы.
6. Массивы координат для построения линий и вычисления установки ПИД-регулятора. С целью экономии памяти и ускорения работы программы массивы координат было решено реализовать в форме очереди. Данная структура данных подходит наилучшим образом, так как с каждым новым шагом, итерацией полёта в форме бесконечного цикла добавляются новые данные, которые сразу появляются на графике. Одновременно с этим пропадает необходимость в данных, которые были добавлены последними, так как построенные линии при каждом шаге смещаются и последние точки выходят за границы графика.

graphX – значения по оси X окна графика. На каждом шаге цикла ось X увеличивается. lowEdgeY и upEdgeY – значения по оси Y нижней и верхней границы туннеля. averageTunnelWidthY – средние значения высоты туннеля по оси Y. Данные массивы относятся, помимо прочего, к туннелям и имеют длину размерности окна графика по X, равную graphicWindowSize. То есть эти линии строятся по всей длине графика.

modelX и modelY – значения по оси X и Y модели самолета. averageSmoothY – значения уставки для ПИД-регулятора или сглаженное среднее значение высоты туннеля по оси Y. Данные массивы относятся к модели самолета и имеют длину, равную значению выдвижения вперед относительно левой границы графика pointMovedForward. То есть линия полета самолета строится не на всей длине графика (по оси Х), а только на её pointMovedForward первых точках.

Значения приведенных в данном абзаце массивов в форме очередей можно увидеть в коде программы.

Класс «PIDRegulator». В данном классе реализуется регулятор. Класс наследует класс «Data» для преемственности необходимых переменных и содержит одну функцию computePID. Для красоты и удобства данная функция реализована таким образом, что не принимает никаких входных значений. Однако классический ПИД-регулятор на входе принимает два значения, текущее положение объекта и уставку – желаемое, необходимое положение объекта. Данные параметры вписаны внутрь самой функции computePID. В качестве текущего положения самолета выбирается последнее значение очереди modelY, то есть положения самолета по оси Y. В качестве уставки – последнее значение очереди averageSmoothY, то есть сглаженного среднего значения высоты туннеля по оси Y. Затем происходит вычисление необходимых параметров регулятора и на выходе функция выдаёт число, соответствующее необходимому воздействию на объект для достижения им уставки. В настоящей работе выход функции computePID можно обозначить по-разному, например, как количество оборотов мотора, которое необходимо применить к самолету, чтобы он достиг необходимого положения (уставки). Также физически выход функции computePID можно трактовать как количество топлива, впрыскиваемого в двигатель самолета, или как положение педали газа. В целом это не имеет особого значения, так как ПИД-регулятору не важно, чем именно управлять, как указывалось в первой главе. При корректно настроенных ПИД коэффициентах процесс регулирования с большой вероятностью будет эффективным, что и демонстрируется в настоящей работе.

Класс «FlyingModel». В данном классе реализуется объект регулирования – модель летящего самолёта. Класс наследует класс «Data» для преемственности необходимых переменных и содержит одну функцию flying. Данная функция имеет один входной параметр – движущую силу, воздействующую на самолет. Вопрос о разных физических трактовках данного входного параметра поднимался в предыдущем абзаце. Забегая вперед, стоит указать, что входом функции flying модели самолета является выход функции computePID ПИД-регулятора. Сама модель самолета основывается на физических силах, действующих на летящий самолет. Из соображений простоты и понятности было решено наисложнейший процесс моделирования полёта самолета свести к воздействию трех сил: уже упомянутая движущая сила, сила инерции (учитывает уже накопленную энергию) и сила тяжести. Согласно второму закону Ньютона, ускорение, с которым движется тело, прямо пропорционально равнодействующей всех сил и обратно пропорционально массе этого тела. Таким образом, на основе единственного входного параметра – движущей силы, можно вычислить ускорение, которое приобретёт модель самолета под действием этой силы. Далее взяв первую и вторую производную ускорения по времени, можно получить значения скорости и позиции. Последний параметр как раз и является выходом функции flying.

**3.2. Формирование изменяющихся внешних условий и графическое отображение результата**

Далее рассмотрим класс «Graphic». В данном классе реализуется визуализация полета самолета – построение графика полета в реальном времени. Класс наследует класс «Data» для преемственности необходимых переменных и содержит две функции updateGraphic и writeFinalFrame. Класс импортирует библиотеку matplotlib.pyplot для построения графика в реальном времени и библиотеку os для определения директории нахождения исполняемого файла. Для создания графика реального времени в начале класса вызываются необходимые функции библиотеки matplotlib.pyplot. Сначала функцией plt.ion график переключается в интерактивный режим для возможности менять график в уже открытом окне без необходимости его снова открывать. Далее создается фигура fig, к ней добавляется график ax, очищаются оси графика и назначается диапазон оси Y графика. После чего на созданном графике строятся линии по точкам в виде зависимости y от х. То есть для построения линии на графике необходимо указать два основным параметра – абсциссы и ординаты точек линии. Данные параметры обычно указываются в виде массива. Затем при необходимости точки последовательно можно соединить, и получится итоговая линия. Также библиотека поддерживает множество дополнительных графических настроек, таких как цвет линии, размер, стиль, прозрачность и многое другое.

Всего на графике при каждом цикле отображается три линии и одна точка. lowTunnelBorder и upTunnelBorder – нижняя и верхняя линии туннеля, обозначены черным цветом. modelTail – линия «хвоста» или «шлейфа» самолета, представляющая pointMovedForward последних точек позиции самолета. Она обозначена красным цветом. modelHead – точка в виде черного треугольника, обозначающая текущую позицию модели самолета. Вид окна графика со всеми вышеуказанными объектами приведен на рисунке 11. Стоит отметить, что все массивы координат точек линий как по X, так и по Y, реализованы в форме очередей. Однако такая структура хранения данных не позволяет обратиться ко всем элементам сразу, поэтому для передачи в качестве входных параметров функции отрисовки линий (функция plot) очереди предварительно необходимо перевести в формат списка.

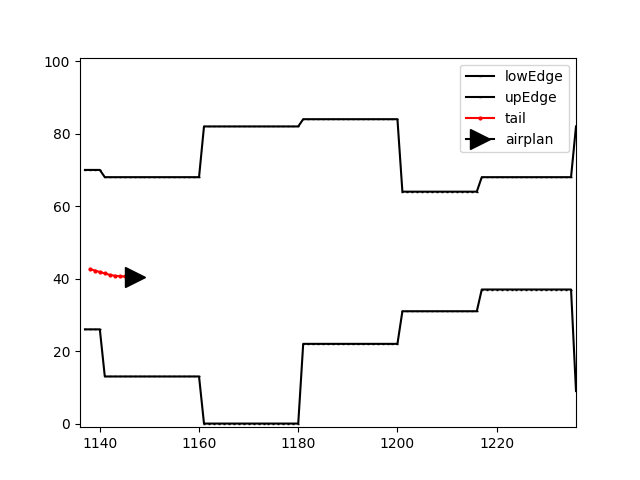


Рис. 11. Окно графика с подписями всех линий и точки.

Основной функцией данного класса является updateGraphic. Как следует из названия она предназначена для обновления графика на каждом шаге цикла. У неё нет ни входных, ни выходных параметров. Данная функция переопределяет оси X и Y всех строящихся объектов на значения, хранящиеся в соответствующих очередях. Для нижней линии туннеля lowTunnelBorder — это graphX и lowEdgeY, для верхней upTunnelBorder – это graphX и upEdgeY, для «шлейфа» самолета modelTail – это modelX и modelY, для самого самолета modelHead – это последние значения modelX и modelY. После этого осуществляется обновление изменённой фигуры, то есть происходит перерисовка.

Дополнительно также была реализована функция writeFinalFrame для возможности фиксирования момента, при котором произошло столкновение самолета с туннелем или досрочное окончание полета. Функция сохраняет картинку последнего шага в формате .png в директории исполняемого файла и затем закрывает график. Название сохраняемой картинки определено как LastResult, что не является уникальным именем – при каждом срабатывании данной функции картинка будет перезаписываться.

Класс «Processing». В данном классе реализуется обработка, перезаписывание и проверка данных. Класс наследует класс «Data» для преемственности необходимых переменных и класс «Graphic» для возможности вызова функции writeFinalFrame, так как эта функция связана с объектом графика plt. Данный класс содержит 4 функции randoms, updateData, checkAndDelay и buttonExit. Класс импортирует библиотеку random для генерации случайных чисел и библиотеку time для задержки между итерациями бесконечного цикла.

Функция randoms вызывается для создания нового туннеля путем генерации новой пары чисел, первое из которых определяет нижнюю границу туннеля по оси Y, а второе верхнюю. Она отличается от первоначального определения случайным образом границ туннеля в классе «Data» тем, что дополнительно к уже имеющимся ограничениям, связанным с минимальной высотой туннеля и размером графика, накладываются новые ограничения. Основное из них связано с предположением, что новый туннель должен иметь связь со старым в виде общего прохода, иначе для самолета не останется пролёта, и он неминуемо столкнётся с границами. Формально это ограничение можно сформулировать так – нижняя граница нового туннеля не должна превышать верхнюю границу старого (рис. 12а), а верхняя граница нового не должна быть ниже нижней границы старого (рис. 12б). Также накладывается ограничение на разброс высот туннеля для более стабильного полета. Все эти условия реализованы с помощью запоминания старых границ и генерации новых с их учётом. Стоит отметить, что в данной функции каждый раз перезаписываются переменные lowTunnelEgde и upTunnelEgde, относящиеся именно к классу «Processing». Эти же переменные в классе «Data» остаются без изменений. Их локальное расположение в классе «Processing» обусловлено тем, что они используются только внутри этого класса.

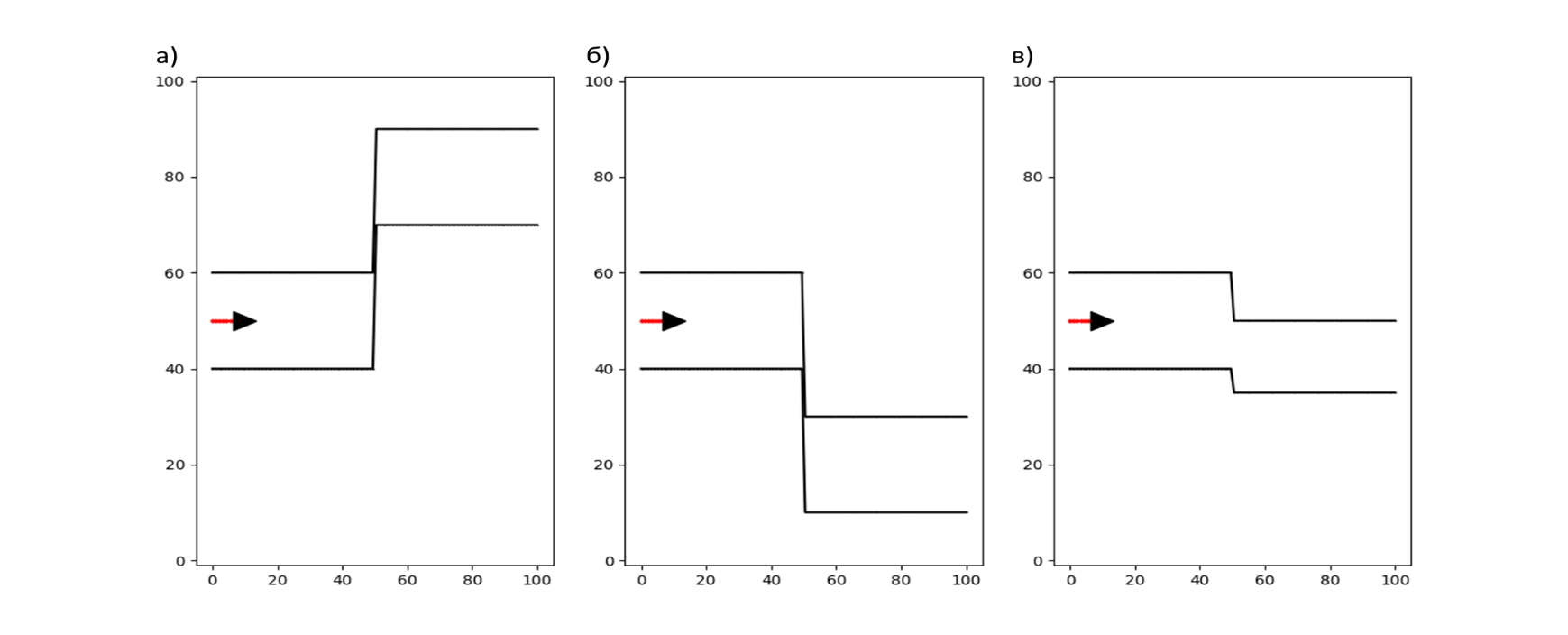


Рис. 12. Пример соотношения нового туннеля с текущим: а,б – общего прохода по оси Y между туннелями нет, в – общий проход имеется.

Функция updateData предназначена, как следует из названия, для обновления данных в массивах координат точек линий. Так как массивы реализованы в форме очередей, то добавление нового элемента автоматически удаляет самый «старый» элемент, что очень удобно в случае бесконечного цикла. В случае хранения в форме списков массивы бы бесконечно увеличивались (если их не удалять специально). Данная функция имеет два входных параметра: новая позиция модели самолета по оси Y и следующая точка по оси X графика. Данные переменные добавляются в соответствующие очереди. Также обновляются остальные очереди исходя из значений границ туннеля по оси Y. Здесь стоит обратить внимание на переменную, которая добавляется в очередь averageSmoothY – очередь сглаженного среднего значения высоты туннеля по оси Y, она же очередь уставки для ПИД-регулятора. В процессе регулирования значения данной очереди по сути являются ключевыми, так как на их основе через ПИД-регулятор вычисляется управляющее воздействие, которое потом передаётся на модель самолета. Можно сказать, что от того, как именно выбираются значения для этой очереди, во многом зависит эффективность регулировки. Выбираются они на основе среднего значения обычных средних значений по tunnelLenght вперед идущим точкам (для этого вводится специальная переменная aveSmoothSum – сглаженное среднее значение) с учетом граничных условий: выход этого сглаженного среднего значения за пределы допустимого диапазона или, другими словами, опасного приближения к границам туннеля. При приближении или выходе aveSmoothSum за границу в итоговую очередь averageSmoothY вместо aveSmoothSum добавляется максимально разрешенное значение, которое равно этой границе + или – (в зависимости от того, нижняя или верхняя граница) допустимый предел приближения к границе averageLimit. Таким образом, уставка ПИД-регулятора или требуемая позиция самолета всегда будет внутри туннеля и не выйдет за него, что должно обеспечить корректную работу процесса регулирования.

Функция checkAndDelay предназначается для нескольких проверок и временной задержки между итерациями бесконечного цикла. Основная проверка отвечает за условия выхода из бесконечного цикла и заключается в определении, вышел ли самолет за границы туннеля. Для этого сравнивается его текущее положение с положением верхней и нижней границы туннеля. В случае превышения границы происходит завершение полета – вызывается встроенная в Python функция exit. Предварительно в консоль выводится уведомление об окончании полета со счётом, выраженным в пройденных итерациях/точках по оси Х, а также сохраняется картинка момента столкновения самолета с помощью уже упомянутой функции writeFinalFrame класса Graphic (рис. 13). Если же полет нормальный, то далее проверяется необходимо ли генерировать новый туннель. Если число отсчетов по оси Х кратно длине туннеля tunnelLenght, то есть предыдущий туннель полностью достроился и имеет длину tunnelLenght, значит пора генерировать новый туннель с помощью функцию randoms. Проверка реализована через операцию остаток от деления. Далее идет проверка необходимо ли ускорять, усложнять полет. Она реализована как предыдущая, только число отсчетов по оси Х проверяется на кратность с acceleration (количество шагов, пройдя которые необходимо усложнять условия). При выполнении условия происходит усложнение полета за счёт уменьшения длины tunnelLenght и минимальной ширины tunnelHeightMin туннеля на параметр reductionStep. Однако для того, чтобы со временем уменьшение длины и минимальной ширины туннеля не привело к его схлопыванию (в таком случае самолету физически не останется места для пролёта) данное условие имеет ограничение. При достижении предельных минимальных параметров размера туннеля условие усложнения пропускается ключевым словом pass. В конце после всех проверок с помощью вызова функции sleep библиотеки time происходит временная задержка в выполнении программы, другими словами, остановка выполнения программы на заданное время в секундах. Задержка помимо прочего необходима для визуально удобного отображения полета на графике с возможностью пользовательского контроля частоты кадров. Важно отметить, что при многопоточности функция задержки фактически останавливает только тот поток, в котором она вызывается, а не всю программу.

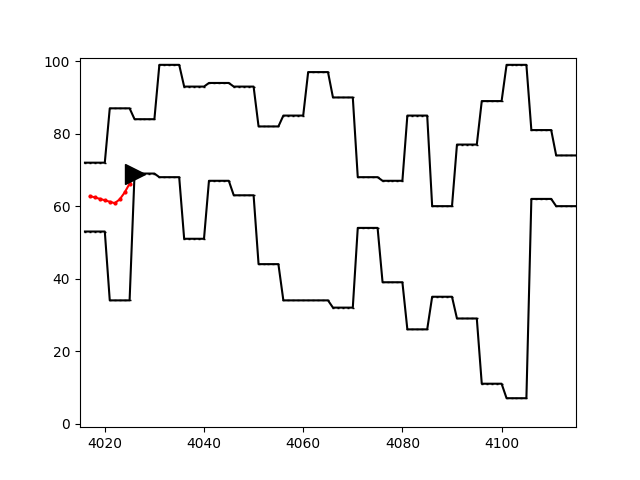


Рис. 13. Момент столкновения самолёта с границей туннеля, в консоль вывелось «Полет окончен (」°ロ°)」Счёт — 4026».

И наконец функция buttonExit служит для досрочного окончания бесконечного цикла, то есть полёта, путём нажатия соответствующей кнопки. По содержанию функция аналогична с действиями при столкновении самолета с туннелем: в консоль выводится уведомление о досрочном окончании полета с достигнутым счётом, сохраняется картинка момента досрочного окончания и происходит выход из программы. Данная функция отмечена декоратором @classmethod для возможности её вызова напрямую через обращение к классу, без необходимости создания его экземпляра.

И наконец основной исполняемый файл программы Main. Он наследует все ранее упоминавшиеся классы, а также библиотеку threading для создания многопоточности, библиотеку sys для доступа к аргументам командной строки, а также классы QApplication и QPushButton класса QtWidgets библиотеки PyQt5 для создания приложения с кнопкой. Файл запускает бесконечный цикл полёта и приложение с кнопкой его досрочного завершения. При написании кода выяснилось, что в бесконечный цикл трудно, а может и невозможно, добавить прерывание этого цикла по внешнему событию (например, по нажатию какой-нибудь клавиши клавиатуры). Поэтому было решено вынести процесс полёта в отдельный поток. В качестве внешнего события прерывания была выбрана кнопка, причем для корректной работы её необходимо запускать в основном потоке.

Итак, за процесс полёта отвечает функция autopilot. В ней сначала назначаются экземпляры класса графика, обработки данных, модели самолета и ПИД-регулятора. Также определяется счётчик итераций i. Далее располагается бесконечный цикл, то есть процесс полёта. Тело цикла состоит из последовательно вызываемых функций. Сначала ПИД-регулятор на основе значений текущего положения самолета и уставки (определение двух этих значений реализовано внутри самой функции computePID класса PIDRegulator) вычисляет необходимое управляющее воздействие на самолет engineSpeed. Далее это воздействие передается самолету, отчего последний летит в новую позицию NewModelPos. Эта переменная представляет собой новую Y координату самолета. Потом происходит обновление данных с учетом достигнутой самолетом позиции NewModelPos и номера итерации i, который представляет собой, по сути, новую Х координату графика. Затем происходит обновление графика для визуального отображения вышеприведенных вычислений. В конце осуществляются проверки на столкновение с границей, на условие генерации нового туннеля и на условие усложнения полета, после чего при отсутствии столкновения вызывается временная задержка. После вызова всех функций одной итерации/шага полета происходит увеличение счётчика и цикл запускается по новой.

После определения функции autopilot приведена основная часть кода, помещенная внутрь условия «if \_\_name\_\_ == "\_\_main\_\_":» для запуска программы только напрямую. Здесь происходит запуск приложения кнопки с помощью встроенных функций библиотеки PyQt5. Это довольно стандартный процесс: создаётся главный поток приложения с GUI, добавляется виджет кнопки, нажатие кнопки соединяется с определенным действием, виджет переводится в видимый режим. Кнопка имеет название Exit и при её нажатии вызывается функция buttonExit класса Processing, которая сразу же завершает программу. Для корректной работы виджета кнопки приложение с GUI должно быть запущено в основном потоке, поэтому процесс полёта был помещен в другой поток. Для этого с помощью класса Thread библиотеки threading был создан и запущен поток с именем autopilotThread, внутри которого была выполнена функция autopilot.

**ЗАКЛЮЧЕНИЕ**

Моделирование – это сложный процесс, включающий в себя создание и функционирование модели. Модель, в свою очередь, является аналогом реалной системы и призвана помочь аналитикам предсказать последствия изменений в системе. Благодаря модели можно изучить структуру объекта, его взаимодействие с окружающей средой, научиться управлять и предсказывать поведение объекта при изменяющихся условиях.

В жизни с давних времен люди стремились к удобству и комфорту - они всегда искали способы устранить трудности, упростить свою работу, увеличить ее эффективность и качество, чтобы иметь больше свободного времени на то, что действительно важно для них. Один из ярких примеров такого подхода – внедрение ПИД-регулятора.

Пропорционально-интегрально-дифференцирующим (ПИД) регулятором называют механизм с обратной связью, который применяется в объектах автоматического управления при создании управляющего сигнала, который обеспечивает требуемые точность и качество переходного процесса.

Вопросы регулирования и управления знакомы всем, кто занимался автоматизацией процессов. При попытке автоматизировать любой процесс, в первую очередь необходимо понять технологию этого процесса, представить, как всё происходит и взаимосвязано. Лишь после этого можно создавать технологическую схему и внедрять регуляторы.

Существует множество способов регулирования, но один из самых лёг-ких, заслуживающих доверия и зарекомендовавших себя на протяжении длительного времени – это ПИД-регулятор. ПИД-регуляторы применяют в большом количестве промышленных приложений, основанных на автоматическом управлении процедурами. Изменение параметров регулятора позволяют настроить величину и характер отклика, исходящего от каждого режима управления.

Объектно-ориентированное программирование, или ООП, представляет собой методологию программирования, которая базируется на использовании объектов, в состав которых входят данные, или атрибуты, и функции, или методы, функционирующие с вышеуказанными данными.

В данном проекте были использованы важные принципы объектно-ориентированного программирования: инкапсуляция, абстракция, наследование. Принципы ООП являются важным инструментом для создания более модульного, повторно используемого и поддерживаемого кода. Понимание применения объектно-ориентированного программирования позволяет повысить навыки программирования, и разрабатывать более весомые, приспосабливающиеся и эффективные программы.

Целью созданной программы является демонстрация применения регулятора для управления объектом. В качестве регулятора был выбран наиболее распространенный и часто используемый ПИД-регулятор. Его применяют в самых разных областях науки, техники, промышленности, а также DIY. В качестве объекта управления использовалась модель самолета. В разработанной программе ПИД-регулятор осуществляет автоматическое управление летящим в туннеле самолетом. При вылете самолета за границы туннеля программа завершается. Полёт реализован в виде бесконечного цикла, если не наступят условия выхода из цикла, полёт не завершится никогда. Работоспособность и эффективность разработанного процесса регулирования показана с помощью изменяющихся внешних условий – изменение со временем размеров туннеля, по которому летит самолет. В целом, созданная программа похожа на бесконечную аркаду (жанр компьютерных игр). Программа разделена на 6 файлов – 5 классов и исполняемый файл Main.

Таким образом, проект показал, что моделирование помогает воссоздать условия, приближённые к реальным. В нашем случае при построении модели самолёта учитывались его масса, ускорение свободного падения. Удалось визуально продемонстрировать возможности ПИД-регулятора при меняющихся внешних условиях на примере летящего самолёта в туннеле. То есть были построены модели самолёта, туннеля с изменяющимися сторонами и применён ПИД-регулятор для того, чтобы компьютер сам рассчитывал необходимые обороты, которые должен совершить самолёт для достижения нужного положения.

В настоящей работе выход ПИД-регулятора можно обозначить по-разному, например, как количество оборотов мотора, которое необходимо применить к самолету, чтобы он достиг необходимого положения (уставки). Также физически выход можно трактовать как количество топлива, впрыскиваемого в двигатель самолета, или как положение педали газа. В целом это не имеет особого значения, так как ПИД-регулятору не важно, чем именно управлять. При корректно настроенных ПИД коэффициентах процесс регулирования с большой вероятностью будет эффективным, что и демонстрируется в настоящей работе.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Аршам Х., Анализ возмущений в дискретно-событийном моделировании, моделирование и моделирование, 11(1), 21-28, 1991.
2. Аршам Х., Анализ «что-если» в компьютерном моделировании Models: A Comparative Survey with Some Extensions, Mathematical and Computer Modeling, 13(1), 101-106, 1990.
3. Аршам Х., Фейервергер А., Маклиш Д., Креймер Дж. и Рубинштейн Р., Анализ чувствительности и проблема «что, если» в имитационном анализе, Математическое и компьютерное моделирование, 12 (1), 193–219, 1989.
4. Аршам Х., Метод моделирования для оценки возмущенных сетей стохастической активности, Моделирование, 58(8), 258-267, 1992.
5. Аршам Х., Экстраполяция производительности при моделировании дискретно-событийных систем, Журнал системных наук, 27(9), 863-869, 1996.
6. Бажанов В. Л., Вайшнарас А. В. Программа «ММ-настройка» для определения параметров ПИД-регуляторов по методу масштабирования // Автоматизация в промышленности. 2007. № 6
7. Ботов Антон (перевод) статьи «PID Without a PhD» Tim Wescott: сайт. – URL: https://wiki.roboforum.ru/index.php?title=Перевод\_статьи\_%22Просто\_о\_ПИД-алгоритмах%22
8. Демарко Том. «Человеческий фактор: успешные проекты и команды»: сайт. – URL: https://www.redcross-irkutsk.org/upload/catalog/files/products/871.pdf
9. Доусон М. Программируем на Python. – СПб.: Питер, 2014. – 416 с.: ил. – 3-е изд.
10. Жасталап А. Т. Использование компьютерного моделирования при изучении раздела «Механика» в средней общеобразовательной школе // Актуальные исследования. 2021. №16 (43). С. 113-118: сайт. – URL: https://apni.ru/article/2242-ispolzovanie-kompyuternogo-modelirovaniya
11. Журнал «СТА» №4 / 2019 СТР. 92: сайт. – URL: https://read.cta.ru/cta2019-4/94/
12. Кнут Дональд. «Искусство программирования»: сайт. – URL: https://djvu.online/file/g5Yhh3ccJ3fcy
13. Кормен Томас, Лейзерсон Чарльз. «Алгоритмы: построение и анализ»: сайт. – URL: https://lib.fbtuit.uz/assets/files/.-by.......z-lib.org.pdf
14. Купер Алан. «Психбольница в руках пациентов»: сайт. – URL: https://дз.ею/tmp/Bibl\_progr\_Sb\_187kn/Психбольница%20в%20руках%20пациентов.%202018.pdf
15. Макконнелл Стив. «Совершенный код. Мастер-класс»: сайт. – URL: https://djvu.online/file/12izBScOYDZZQ
16. Макконнелл Стив. «Сколько стоит программный проект»: сайт. – URL: https://djvu.online/file/eLxopvZ3htdsN
17. Мартин Роберт. «Чистый код»: сайт. – URL: https://k0d.cc/storage/books/Architecture/Чистый%20код.%20Создание,%20анализ%20и%20рефакторинг%20(Мартин%202013).pdf
18. Никулин Е. А. Основы теории автоматического управления. Частотные методы анализа и синтеза систем / Учеб. пособие для вузов – СПб.: БХВ-Петербург, 2004. – 640 с.: илл. – с.573-574
19. Окулов С. М. «Алгоритмы компьютерной арифметики»: сайт. – URL: https://glavkniga.su/filecont/49584.pdf
20. Паронджанов В. Д. «Учись писать, читать и понимать алгоритмы»: сайт. – URL: https://studylib.ru/doc/1696677/uchis.\_-pisat.\_--chitat.\_-i-ponimat.\_-algoritmy
21. Полоцкий Л. М. Автоматизация химических производств. / Л. М. Полоцкий, Г. И. Лапшенков. – М.: Химия, 1982. – 296 с.
22. Стефани Е.П. Основы расчета настройки регуляторов теплоэнергетических процессов. – М.: Энергия, 1972
23. Фримен Эрик, Фримен Элизабет. «Паттерны проектирования»: сайт. – URL: https://djvu.online/file/eszMdSTdfS8BI
24. Ziegler J.G., Nichols N.B. Optimum settings for automatic controllers. // Transactions of the ASME, Vol.64. pp. 759-768, 1942.
25. Исключения в python. Конструкция try - except для обработки исключений // Python 3 для начинающих: сайт. – URL: https://pythonworld.ru/tipy-dannyx-v-python/isklyucheniya-v-python-konstrukciya-try-except-dlya-obrabotki-isklyuchenij.html
26. Практика хорошего кода // Habr: сайт. – URL: https://habr.com/ru/articles/206868/
27. Что такое красивый код, и как его писать? // Habr: сайт. – URL: https://habr.com/ru/articles/266969/
28. Шпаргалка по принципам ООП // Tproger: сайт. – URL: https://tproger.ru/translations/oop-principles-cheatsheet/
29. Don't repeat yourself // Wikipedia: сайт. – URL: https://en.wikipedia.org/wiki/Don't\_repeat\_yourself
30. Download PyCharm: Python IDE for Professional Developers by JetBrains: сайт. – URL: https://www.jetbrains.com/pycharm/download/
31. Download Python | Python.org: сайт. – URL: https://www.python.org/downloads/

1. Жасталап А. Т. Использование компьютерного моделирования при изучении раздела «Механика» в средней общеобразовательной школе // Актуальные исследования. 2021. №16 (43). С. 113-118. URL: https://apni.ru/article/2242-ispolzovanie-kompyuternogo-modelirovaniya [↑](#footnote-ref-1)
2. Журнал «СТА» №4 / 2019 СТР. 92 https://read.cta.ru/cta2019-4/94/ [↑](#footnote-ref-2)