Министерство науки и высшего образования Российской Федерации
ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»

Департамент математики, механики и компьютерных наук ФГАОУ ВО УрФУ ИЕНиМ.

Оценка работы _____
Руководитель от УрФУ Кумков С.С

Тема задания на практику

Моделирование движения динамических объектов по кусочно-прямолинейным траекториям

Отчет

Вид практики Производственная практика
Тип практики Производственная практика, научно–исследовательская работа

Руководитель практики от предприятия (организации) Кумков С.С.
Студент Дербенев Л.О.
Специальность (направление подготовки) 01.03.01 Математика
Группа МЕН-400101

Екатеринбург 2024

Содержание

Введение

В настоящее время активно развивается движение гражданских беспилотных летательных аппаратов (БПЛА). С их помощью проводится доставка различных грузов в удаленные поселки (чем раньше занималась только малая авиация), мониторинг состояния природных и искусственных объектов (лесов, озер, рек, трубо-, нефте- и газопроводов, ЛЭП и др.).

При этом конфликтов БПЛА и больших самолётов почти не возникает: в рабочем режиме они движутся на разных высотах, а вход БПЛА в аэропортовые зоны весьма жёстко регламентируется. Однако возникают конфликты БПЛА с другими БПЛА, а также с самолётами малой авиации и вертолётами, которые движутся на тех же высотах, что и БПЛА.

Ситуация такова, что движение БПЛА не диспетчеризируется централизовано, как движение больших самолётов, часто полет малого ЛА объявляется в уведомительном порядке. Поэтому текущая воздушная обстановка актуально недоступна операторам БПЛА, которые кроме того не имеют связи между собой и с диспетчерскими службами УВД. В то же время достаточно чётко прописаны регламенты того, какую информацию о собственном движении во время полёта БПЛА сообщает в эфир в виде широковещательных пакетов.

В таких условиях важной является разработка алгоритмов для бортовых компьютеров БПЛА для обработки полученной информации о движении окружающих аппаратов, малых и больших, и для последующей выработки и отработки манёвров, разрешающих возникающие конфликтные ситуации с другими летательными аппаратами.

В рамках создания таких алгоритмов важным является численное моделирование ансамбля БПЛА, движущихся по тем или иным траекториям в тех или иных условиях. Такое моделирование позволяет провести предварительную оценку качества предлагаемых манёвров уклонения.

Однако для адекватности проводимых оценок требуется, чтобы модели движения и маневрирования БПЛА достаточно точно соответствовали движению реальных аппаратов.

В рамках данной работы были предприняты шаги в направлении создания такого моделирующего программного комплекса и реализации основных моделей движения БПЛА.

1 Теоретическая часть

1.1 Рассматриваемые модели динамики

1.1.1 Материальная точка

Материальная точка — это идеализированное тело, обладающее массой, но не имеющее размеров. В физике материальная точка используется как модель для описания движения объекта, когда его размерами можно пренебречь по сравнению с масштабами задачи. В рамках создаваемого комплекса данная модель является вспомогательной, используемой при отладке его работы.

Материальная точка имеет следующую модель движения:

$$\ddot{r} = m \cdot u, \tag{1}$$

где $r = (x, y, z)^{\mathrm{T}}$ — радиус-вектор объекта, m — масса точки, $u = (u_x, u_y, u_z)^{\mathrm{T}}$ — управление, являющееся ускорением.

Покоординатная запись:

$$\begin{cases} \ddot{x} = u_x, \\ \ddot{y} = u_y, \\ \ddot{z} = u_z \end{cases}$$

1.1.2 Коптер

Недостаток модели материальной точки заключается в том, что нет никаких ограничений на максимальную скорость, развиваемую объектом, а также отсутствие учета сопротивления среды (воздуха) движению объекта.

Простейшей моделью, учитывающей эти обстоятельства, является модель, управляемая *командным сигналом* скорости:

$$\begin{cases} \dot{x} = V_x, \\ \dot{y} = V_y, \\ \dot{z} = V_z, \\ \dot{V_x} = \dfrac{u_x - V_x}{l_{xz}}, \\ \dot{V_y} = \dfrac{u_y - V_y}{l_y}, \\ \dot{V_z} = \dfrac{u_z - V_y}{l_{xz}}. \end{cases}$$

Здесь $u = (u_x, u_y, u_z)^{\mathrm{T}}$ — управление, командный сигнал скорости, имеющий смысл желаемой скорости по каждой из координат; l_{xz}, l_y — коэффициенты инерции для горизонтальной скорости (в плоскости XZ) и вертикальной (вдоль оси Y), соответственно. После выставки командного сигнала на некоторый выбранный постоянный уровень актуальная скорость ЛА за время порядка $3l$ выходит на этот уровень.

Командный сигнал ограничен по модулю, что соответствует максимально возможной скорости, которую может развивать ЛА.

Данная модель является простейшей моделью движения лёгкого коптера, который достаточно быстро может менять скорость своего движения, но имеет ограничение по максимальной скорости из-за относительно малой мощности двигателя.

1.2 Полетный план

Вся траектория задается как $\{(r_i, t_i)\}_{i=1}^{n}$, где $r_i = x_i,y_i,z_i$ - точка трехмерного пространства, t_i - момент времени, в который БПЛА должен оказаться в заданной точке. Считаем, что $t_i < t_{i+1}$.

Заданная траектория, по определению, является ломаной линией. Пусть $n > 1$, рассмотрим отрезок пути (r_k,t_k), (r_{k+1},t_{k+1}), $k = 1 \dots n-1$. По определению, летательный аппарат должен оказаться в точке r_k в момент времени t_k и в точке r_{k+1} в момент t_{k+1}. Или другими словами БПЛА должен пройти расстояние $S_k = |r_{k+1} - r_k|$ за время $\tau_i = t_{k+1} - t_k$. Значит мы можем найти скорость $V_k = \frac{S_k}{\tau_k}$.

Пусть теперь $n > 2$, рассмотрим два отрезка, заданные тремя точками (r_k,t_k), (r_{k+1},t_{k+1}), (r_{k+2},t_{k+2}) $k = 1 \dots n-2$. Найдем для скорости V_k и V_{k+1} для них, как это было сделано выше. Другими словами, наш БПЛА должен моментально сменить скорость в точке (r_{k+1},t_{k+1}) с V_k на V_{k+1}, но в реальной жизни летательные аппраты так не могут. Данную проблемы решает *штурманская прокладка*, она позволит плавно переходить с одного отрезка на другой и также плавно менять скорость. По сути аппарат будет прицеливаться на нее. О ее построении будет пункт чуть ниже.

??? И какие-то слова о том, что под углом летать не умеем - надо прокладку и прицеливаться на нее

1.3 ПИД-регулятор

ПИД-регулятор (Пропорционально-Интегрально-Дифференциальный регулятор) является одним из наиболее распространенных методов регулирования систем управления. Он состоит из трех основных компонентов: пропорциональной, интегральной и дифференциальной составляющих.

Пропорциональная составляющая определяет выходной сигнал контроллера пропорционально разности между желаемым и текущим значением управляемой величины. Это позволяет реагировать на ошибку управления и регулировать систему.

Интегральная составляющая интегрирует ошибку управления с течением времени, что позволяет уменьшить статическую ошибку системы и обеспечить точное следование заданному значению.

Дифференциальная составляющая учитывает скорость изменения ошибок и предотвращает быстрые колебания системы, обеспечивая более плавное и стабильное управление.

В итоге, комбинация трех компонентов ПИД-регулятора позволяет эффективно и точно управлять системой, обеспечивая минимальное перерегулирование и быстрое достижение заданного значения.

Пусть r - заданное значение, которое нужно поддерживать, $e = (r - y)$ - невязка или ошибка регулирования. Тогда для линейной, стационарной системы ПИД-регулятор имеет вид:

$$u(t) = P + I + D = K_p \cdot e(t) + K_i \cdot \int_0^T e(t)d\tau + K_d \frac{de}{dt}, \tag{2}$$

где K_p, K_i, K_d – коэффициенты усиления пропорциональной, интегрирующей и дифференцирующей составляющих.

Рассмотрим простую линейную задачу:

$$\dot{x} = Ax + Bu, \quad x \in R^n, u \in R^m \tag{3}$$

$u = 0$ - разрешенное управление. Отсюда следует, что $x = 0$ является точкой равновесия. Задача вывести систему в точку равновесия.

В данной работе был использован линейный регулятор, то есть $u = Kx$, где $K \in R^{m \times n}$. Теперь задача имеет вид:

$$\dot{x} = Ax + BKx = (A + BK)x \tag{4}$$

Данная система является устойчивой $\Longleftrightarrow \forall \lambda$ - собственное значение, выполняется: $Re\lambda < 0$.

В данной работе будем использовать только пропорциональный регулятор:

$$u = -k_x \cdot (x - x_w) - k_V \cdot (V_x - V_{x,w})$$

1.4 Построение регулятора для материальной точки

Построим регулятор для материальной точки. Мы имеем

$$\ddot{r} = u$$

или распишем в эквивалентной форме

$$\ddot{x} = u_x$$
$$\ddot{y} = u_y$$
$$\ddot{z} = u_z$$

Пусть $X = (x,y,z,v_x,v_y,v_z)^{\mathrm{T}}$, где v_i – соответствующие компоненты скорости.

Запишем задачу:

$$\dot{X} = AX + Bu = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot X + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot u = \begin{pmatrix} O_3 & I_3 \\ O_3 & O_3 \end{pmatrix} \cdot X + \begin{pmatrix} O_3 \\ I_3 \end{pmatrix} \cdot u$$

То есть можно сказать, что $K \in R^{3 \times 6}$ или другими словами у нас 18 параметров, но заметим, что так как мы используем пропорциональный регулятор, то не равны нулю лишь $K_{x \to u_x}$, $K_{y \to u_y}$, $K_{z \to u_z}$, $K_{v_x \to u_x}$, $K_{v_y \to u_y}$, $K_{v_z \to u_z}$:

$$K = \begin{pmatrix} K_{x \to u_x} & 0 & 0 & K_{v_x \to u_x} & 0 & 0 \\ 0 & K_{y \to u_y} & 0 & 0 & K_{v_y \to u_y} & 0 \\ 0 & 0 & K_{z \to u_z} & 0 & 0 & K_{v_z \to u_z} \end{pmatrix}$$

То есть получаем следую запись

$$\dot{x} = V_x$$
$$\dot{y} = V_y$$
$$\dot{z} = V_z$$
$$\dot{V}_x = K_x x + K_{V_x} V_x$$
$$\dot{V}_y = K_x x + K_{V_x} V_x$$
$$\dot{V}_z = K_x x + K_{V_x} V_x$$

1.5 Прокладка траектории полета

Как уже упоминалось в одном из пунктов выше, летательные аппараты не умеют моментально менять скорость и ровно двигаться по заданным траекториям, из-за внешних и физических факторов. В данном пункте рассмотрим и построим траекторию полета.

1.5.1 «Наивная» прокладка

В данной ситуации прокладкой является та ломанная, которую нам дал пользователь. Как видно из раздела про Полетный план и моделирование движения, БПЛА будет постоянно сходить с заданной траектории, в местах, где нужно быстро изменить скорость и направление движения. Поэтому данный подход нам не подходит

1.5.2 Прокладка с упреждением поворотов

В этом случае главным отличием от «Наивной прокладки» будет перерасчет траектории с упреждением поворота. То есть мы будет пред-вычислять желаемые скорость и позицию в момент поворота.

Снова рассмотрим задачу с двумя отрезками заданными тремя точками. Пусть снова $n > 2$, рассмотрим два отрезка, заданные тремя точками (r_k,t_k), (r_{k+1},t_{k+1}), (r_{k+2},t_{k+2}) $k = 1 \dots n-2$. Найдем скорости V_k и V_{k+1} для них, как в пункте про полетный план. Давайте для простоты переобозначим точки и скорости следующим образом:

$$
\begin{aligned}
A &:= r_k, \\
B &:= r_{k+1}, \\
C &:= r_{k+2}, \\
V_1 &:= V_k, \\
V_2 &:= V_{k+1}
\end{aligned}
$$

Примерно все выглядит следующим образом:

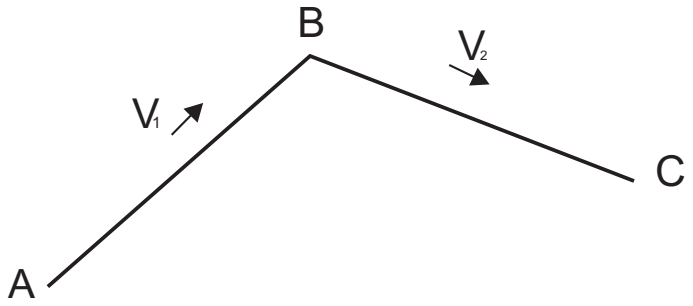

Рис. 1. Два отрезка пути

По сути своей, нужно равноускоренно провести БПЛА по касательной окружности к данным отрезкам пути. Для этого понадобится узнать точки касания, боковое ускорение, угловую скорость и угол поворота. Каждый летательный аппарат имеет заданное боковое ускорение, обозначим его a_n. Первым делом найдем радиус по формуле:

$$r = \frac{V_1^2}{a_n}$$

Затем найдем косинус угла α между отрезками AB И BC по известной формуле:

$$\cos\alpha = \frac{\langle V_1, V_2\rangle}{|V_1||V_2|}$$

Теперь, зная радиус и угол между отрезками пути, можно найти длину h – расстояние от точки касания до точки стыка двух отрезков пути.

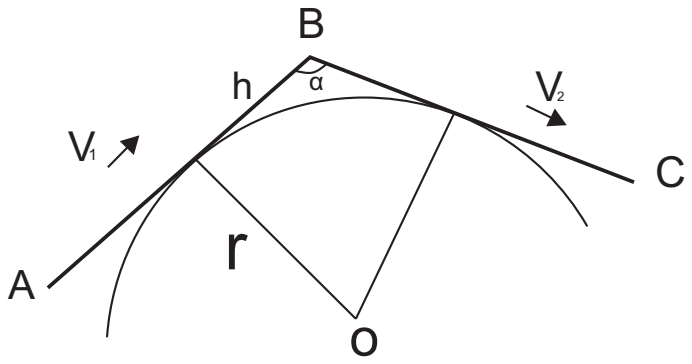

Рис. 2. Касательная окружность к двум отрезкам пути.

Как видно из рисунка 2, так как радиус ,проведенный в точку касания, перпендикулярен касательной линии, длину h можно найти, вычислив тангенс половинного угла α, затем умножив его на радиус r. По известной тригонометрический формуле

$$\tan\frac{\alpha}{2} := \sqrt{\frac{1-\cos\alpha}{1+\cos\alpha}}$$

. Значит

$$h := r \cdot \sqrt{\frac{1-\cos\alpha}{1+\cos\alpha}}$$

Теперь можно найти координаты точек касания отступив от точки B, на расстояние h вдоль отрезков. Для этого найдем нормированные вектора a,c, которые смотрят вдоль на точки A, C соответственно:

$$a := \frac{BA}{|BA|},$$
$$c := \frac{BC}{|BC|}$$

.

Находим точки касания A',C':

$$A' := a \cdot h,$$
$$C' := c \cdot h$$

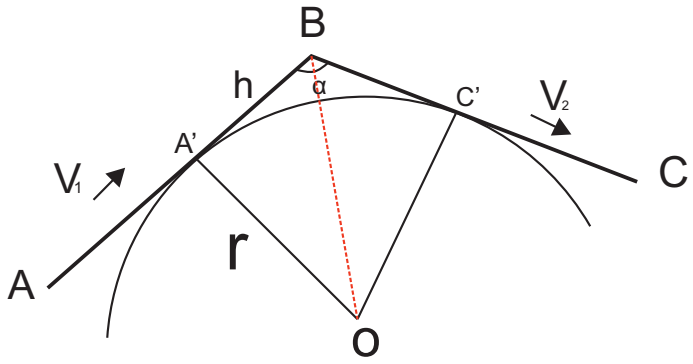

Рис. 3. Касательная окружность к двум отрезкам пути.

Так как траектория пути находится в трехмерном пространстве, нужно использовать формулу поворота для трехмерного пространства:

$$x' = (\cos\theta + n_x^2(1-\cos\theta))x + (n_x n_y(1-\cos\theta) - n_z\sin\theta)y + (n_x n_z(1-\cos\theta) + n_y\sin\theta)z$$
$$y' = (n_x n_y(1-\cos\theta) - n_z\sin\theta)x + (\cos\theta + n_y^2(1-\cos\theta))y + (n_y n_z(1-cos(\theta)) - n_x\sin\theta)z$$
$$z' = (n_x n_z(1-\cos\theta) + n_y\sin\theta)x + (n_y n_z(1-cos(\theta)) - n_x\sin\theta)y + (\cos\theta + n_z^2(1-\cos\theta))z$$

где $n = (n_x,n_y,n_z)^{\mathrm{T}}$ – нормированный вектор оси вращения, θ – угол, на который нужно повернуть, $r' = (x',y',z')^{\mathrm{T}}$ – новые координаты точки на окружности,$r = (x,y,z)^{\mathrm{T}}$ - старые координаты. Также, нужно учесть, что поворот в данной формуле происходит вокруг точки начала, то есть нужно будет сдвинуть старые координаты на радиус вектор центр окружности $r_0 = (x_0,y_0,z_0)^{\mathrm{T}}$, а затем новые координаты сдвинуть обратно. То есть, формула будет выглядеть следующим образом:

$$\begin{aligned}
x' &= x_0 + (\cos\theta + n_x^2(1-\cos\theta))(x-x_0) + (n_x n_y(1-\cos\theta) - n_z\sin\theta)(y-y_0) + \\
&+ (n_x n_z(1-\cos\theta) + n_y\sin\theta)(z-z_0) \\
y' &= y_0 + (n_x n_y(1-\cos\theta) - n_z\sin\theta)(x-x_0) + (\cos\theta + n_y^2(1-\cos\theta))(y-y_0) + \\
&+ (n_y n_z(1-cos(\theta)) - n_x\sin\theta)(z-z_0) \\
z' &= z_0 + (n_x n_z(1-\cos\theta) + n_y\sin\theta)(x-x_0) + (n_y n_z(1-cos(\theta)) - n_x\sin\theta)(y-y_0) + \\
&+ (\cos\theta + n_z^2(1-\cos\theta))(z-z_0)
\end{aligned}$$

Для нахождения вектора оси поворота n, нам понадобятся, уже известные, вектора a и c, а точнее их векторное произведение, ведь по сути нам нужно найти нормированный вектор, перпендикулярный плоскости, в которой происходит поворот. А мы знаем что вектора a,c лежат в этой плоскости. Находим вектор оси следующим образом:

$$n = \frac{c \times a}{|c \times a|}$$

Также, нужно отметить, что порядок умножения именно такой, ведь это задаст правильную ориентацию поворота.

??? картинку с осью и углом поворота

??? возможно стоит написать про то как правильно вычислять скорость

2 Практическая часть

Программа была реализована на языке C++, в среде разработки Microsoft Visual Studio. Выбор стал именно таким, так как он обеспечивает нужное быстродействие вычислений и кроссплатформенность. Все графики были сформированы в Gnuplot 6.0. В данный момент в программе реализовано две динамические модели – материальная точка и коптер.

2.1 Вспомогательные классы и методы

Для данной работы были написаны вспомогательные классы и методы для работы с ними такие как:

- `Vector3` - класс для удобной работы с трехмерными векторами. По сути своей класс обладает тремя полями x,y,z для определения координат. В нем были реализованы базовые операции над векторами:
  - $+$ – сложения и вычитание векторов
  - $*$ – умножения вектора на скаляр
  - $/$ – деления вектора на скаляр
  - $*$ – по-координатное унижение векторов
  - `double scalarProduct(Vector3 v1, Vector3 v2)` – скалярное произведение
  - `Vector3 crossProduct(Vector3 v1, Vectro3 v2)` – векторное произведение
  - `void rotate(Vector3 axis, Vector3 axisPoint, double angle)` – поворот вектора вокруг заданной оси и точки, через которую проходит это ось, на заданный угол
  - `double norm()` – подсчет длины вектора
  - `Vector3 getNormVector()` – получение нормированного вектора из данного
- `Point` - структура для описание точки пути, который задает пользователь. Он состоит из полей:
  - `Vector3 position` – поле для хранения координат позиции точки
  - `double time` – момент времени, в который должен оказаться ЛА в заданной позиции
- `TurnData` - структура для хранения данных, которые нужно для поворота. Имеет следующею поля:
  - `double angularVelocity` – поле для хранения предвычисленного углового ускорения, используется для поворота позиции по окружности
  - `Vector3 axisPoint` – поле для хранения точки оси поворота
  - `double angularVelocity_0` – поле для хранения предвычисленнго углового ускорения, используется для вычисления скорости при повороте
  - `double angularAcceleration` – поле для хранения углового ускорения
  - `Vector3 axis` – поле для хранения оси поворота
- `PathPointType` – класс перечислений, для пометки точек траектории прокладки. Содержит:

  - `DEFAULT` - тип для обозначения точки, после которой двжиение прямолиненое, равномерное.
  - `START_TURN` - тип для обозначения точки, после которой начинается маневр поворота

- `PathPoint` – структура для описания точки пути прокладки, является наследником класса `Point`. Состоит из:
  - `Vector3 position` – поле для хранения координат позиции точки
  - `double time` – момент времени, в который должен оказаться ЛА в заданной позиции
  - `PointType type` - поле для типа точки
  - `TurnData turnData` - поле для данные поворота.

- `Path` - класс для описания пути прокладки. Имеет следующие методы и поля:
  - `vector<PathPoint> path` - приватное поле для хранения вектора, состоящего из точек типа. `PathPoint`.
  - `PathPoint getPointForIndex(int ind)` - метод для получения точки пути по индексу.
  - `void addNewPoint(PathPoint p)` - метод для добавления точки в траекторию прокладки.

2.2 Реализация динамических моделей

Для реализации динамических моделей был создан базовый абстрактный класс FV. Он определяет базовые свойства присущие каждой модели движения, такие как:

- `virtual void next(double h, double end_time) abstract` - виртуальная функция, которая служит для вычисления следующего состояния БПЛА с шагом h до момента end_time.
- `virtual void setPath(vector<Point> path)` - виртуальная функция, которая служит для вычисления прокладки пути. По умолчанию вычисляется по схеме прокладки с упреждением поворота.
- `vector<Point> basePath` - полея для хранения базового пути, который вводит пользваотель.
- `Path dynamicPath` - поле для хранения, пути прокладки, заданной с помощью метода `setPath`.
- `double maxAcceleration` - поле для хранения максимального бокового усокрения.
- `double time` - поле для хранения времени текузего состояния БПЛА

Далее рассмотрим подробнее конкретные реализации динмаических моделей – материальной точки и коптера, которые являются наследниками класса `FV`

2.2.1 Материальная точка

Класс материальной точки `MaterialPoint` является публичным наследником класса `FV`. А значит обладает всеми уже ранее описанными методами и полями, но также имеет и собственные. А именно:

- `string getName()` – метод возвращающий имя БПЛА.
- `int32_t getType()` – метод метод возвращающий тип БПЛА, у материальной точки 0.
- `vector<Point> getBasePath()` – возвращает базовый путь, введенный пользователем.
- `Path getDynamicPath()` – возвращает построенный путь прокладки при задании пути.
- `Vector3 acceleration` – поле которое хранит ускорение в текущем состоянии.
- `Vector3 wishPosition` – поле хранящее желаемую позицию, вычисленную в методе `computeWishData`.
- `Vector3 wishVelocity` – поле хранящее желаемую скорость, вычисленную в методе `computeWishData`.
- `string name` – поле, содержащее имя БПЛА.
- `int32_t type` – поле, хранящее тип БПЛА.
- `Vector3* curPosition` – поле хранящее указатель на текущую позицию БПЛА.
- `Vector3* newPosition` – поле хранящее указатель на новую позицию, вычисленную в методе `next`.
- `Vector3* curVelocity` – поле хранящее указатель на текущую скорость БПЛА.
- `Vector3* newVelocity` поле хранящее указатель на новую скорость, вычисленную в методе `next`
- `double k_v` – поле хранящее значение коэффициента вклада позиции на управление.
- `double k_x` – поле хранящее значение коэффициента вклада скорости на управление.
- `void computeWishData(double time_solve)` – метод предназначенный для вычисления желаемой позиции и желаемой скорости в момент тврмени `time_solve`

2.2.2 Коптер

Класс коптер `Copter`, также как и материальная точка является публичным наследником класса `FV`. А значит обладает всеми уже ранее описанными методами и полями, но также имеет и собственные, почти во всем перекликается с материальной точкой, но также имеет собственные особенности:

- `double inertialXZ` – поле для хранения коэффициента инерции в плоскости OXZ.
- `double inertialY` – поле для хранения коэффициента инерции по оси OY.

2.3 Вычисление «штурманской» прокладки

2.3.1 Разметка траектории

Разберем весь процесс вычисления, того как должен двигаться БПЛА. Начнем с метода `setPath` прописанного в базовом классе `FV`. Данный метод, как упоминалось ранее, принимает вектор из точек траектории пользователя, а затем пробегает по всем его точкам, проверяя отрезки на наличии поворотов. Если поворот нужен, то вместо точек стыка двух отрезок заменяют две точки, к которые являются касательными. Первая точка касания помечается типом `PointPathType::START_TURN` и для нее вычисляются все данные для поворота, указанные в структуре `TurnData`. После данного процесса полученный путь сохраняется в поле `Path dynamicPath`.

???? Нужна ли вставка с кодом ????

2.3.2 Вычисление прокладки

Далее перейдем к методу `computeWishData`. Данный метод вычисляет `wishPosition, wishVelocity` желаемые позицию и скорость соответственно по моменту времени. Данный метод идентичен как в классе `MeterialPoint`, так и в классе `Copter`. В самом начале бинарным поиском происходит нахождение соответствующего отрезка в `Path dynamicPath`. Затем, относительного типа точки пути вычисляется желаемая скорость, а после желаемая позиция. Если точка имеет тип `DEFAULT`, то это значит что нужно двигаться равномерно, прямолинейно до следующей точки, если же тип `START_TURN`, то используя данные для поворота из соответствующего поля структуры, вычисляем скорость, находя касательный к окружности вектор, с помощью векторого произведения вектора угловой скорости и радиус вектора с началом в точке в центре окружности и концом в точке позиции БПЛА. Для вычисления новой позиции на окружности, последняя позиция двигается с помощью метода `rotate`, в котором реализована формула поворота.

2.3.3 Вычисление движения

Теперь перейдем к методу `next` для материальной точки. В данном методе происходит вычисление действительной позиции и скорости БПЛА посредством численного интегрирования. Для интегрирования используется метод Эйлера на левый край, где за производную управление берется регулятор, построенный в соответствующем пункте теоретической части. Регулятор используется один и тот же как для материальной точки:

```
for (int i = 0; i < 3; i++) {
    double x = (*curPosition)[i];
    double v = (*curVelocity)[i];
    (*newPosition)[i] = x + h * v;
    (*newVelocity)[i] = v + h * k_x * (x -  wishPosition[i]
                              + k_v * (v - wishVelocity[i]));
    acceleration[i] = k_x * (x - wishPosition[i])
                    + k_v * (v - wishVelocity[i]);
}
```

так и для коптера:

```
double x = (*curPosition)[0];
double v = (*curVelocity)[0];
```

```
(*newPosition)[0] = x + h * v;
(*newVelocity)[0] = v + h *
            ((k_v * wishVelocity.x + k_x * wishPosition.x) - v) / inertialXZ;
x = (*curPosition)[1];
v = (*curVelocity)[1];

(*newPosition)[1] = x + h * v;
(*newVelocity)[1] = v + h *
            ((k_v * wishVelocity.y + k_x * wishPosition.x) - v) / inertialY;

x = (*curPosition)[2];
v = (*curVelocity)[2];

(*newPosition)[2] = x + h * v;
(*newVelocity)[2] = v + h *
            ((k_v * wishVelocity.z + k_x * wishPosition.x) - v) / inertialXZ;
```

2.4 Результаты

В данном пункте пошагово пройдемся по этапам вычисления. Пусть пользователь задал такой маршрут состоящий из 4 точек:

- $(r_1,t_1) = ((0,0,0),0)$ - точка для нулевого момента времени,
- $(r_2,t_2) = ((10,0,0),10)$ - точка для 10 секунды,
- $(r_3,t_3) = ((10,0,-5),20)$ - точка для 20 секунды,
- $(r_3,t_3) = ((10,0,-15),30)$ - точка для 30 секунды

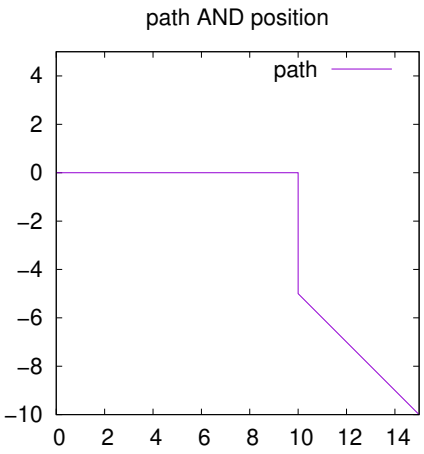

Рис. 4. Путь пользователя

Продемонстрирую поведение материальной точки стоящей в нулевой момент времени в точке(0,0,0), с начальной скорость (0,0,0) если использовать «Наивную» прокладку. Пусть коэффициенты вклада в управление для позиции и скорости соответственно $k_x = -1 =$, $k_v = -1$,

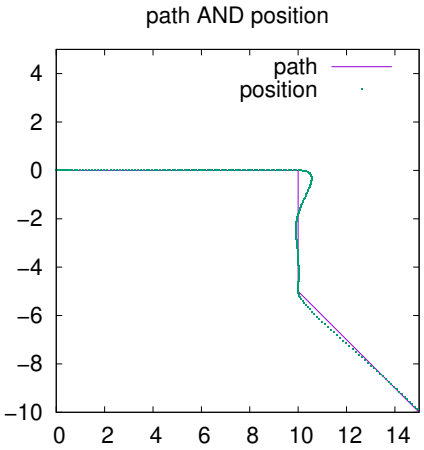

Рис. 5. Путь пользователя

Как видно по зеленой линии, летательный аппарат не успевает изменить скорость на повороте. Теперь построим прокладку с упреждение поворота:

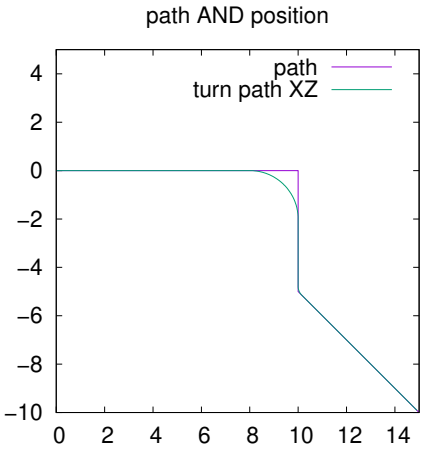

Рис. 6. Путь пользователя и прокладка

Посмотрим как будет себя вести БПЛА, нацелившийся на прокладку:

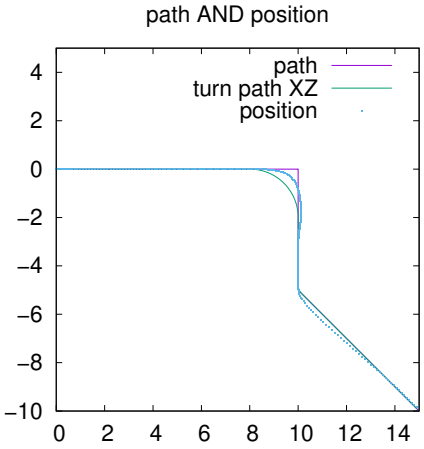

Рис. 7. Путь пользователя

Как видно, результат стал намного лучше. Более лучше результата можно добиться подбором коэффициентов k_x и k_v, уменьшая или увеличивая вклад желаемых позиции и скорости.

Список литературы

[1] Шилдт Герберт C++ базовый курс [Электронный ресурс]: Белорусский государственный университет информатики и радиоэлектроники – Режим доступа: https://www.bsuir.by/m/12_119786_1_98220.pdf (дата обращения: 01.12.2023)

[2] Документация по Microsoft C++, C и ассемблеру [Электронный ресурс]:– Режим доступа: https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170 (дата обращения: 01.12.2023).