TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ

3 CİSİM PROBLEMİ

2 BOYUTLU ORTAMDA SİMÜLASYON

2024-2025 GÜZ DÖNEMİ

ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ

BİL 142 PROJESİ

221201076 / EDA İNAN

Öğr. Gör. OKAY ARIK

1. 3 CİSİM PROBLEMİ

3 Cisim Problemi ile birbirlerinin kütle çekimine maruz kalan üç noktasal cismin rotasını izlenmesi ile ilgilidir. Bu problemin en dikkat çekici özelliği analitik bir çözümünün olmayışıdır. Bu problemin iki cisimli versiyonunda net bir etkileşim hesabı yapılıp rotalar çizilebilirken cisim sayısı arttığında bu mümkün olmamaktadır. Ancak çözülemez olduğu belirtilen Üç Cisim Problemi bazı yaklaşımlar ile çözülebilir, tahmin edilebilir sonuçlar vermektedir. Aşağıda bu problemin çözümü için kullanılan birkaç özel yaklaşımı bulabilirsiniz:

Barrau Konfigürasyonu: Bu yaklaşımda “3-4-5 Pisagor Üçgeni” kullanılır. Üç cisim üçgenin üç noktasına yerleştirilir ve hareketleri bu düzenek üzerinde incelenir.

Lagrange Konfigürasyonu: Burada, eşkenar üçgen kullanılır. Burada kütleler arası mesafeler her zaman eşittir, üç cisimin kütleleri belli bir orana sahip olmasına gerek yoktur ancak üç cismin de açısal hızları aynı olmalıdır.

Euler (Kollinear) Konfigürasyonu: Bu konfigürasyonda, üç cisim aynı doğruya yerleştirilir ve bu üç cisimlerin mesafesi belirli oranlarda tutulur.

Hill Çözümü: Bu çözümde, iki cismin kütlesi üçüncü cisme göre çok büyük seçilir. Böylelikle üçüncü cismin diğer iki cisme bağlı olarak hareket etmesi sağlanır.

Projede, Üç Cisim Problemi 2 boyutlu bir ortamda Newton'un Kütle Çekim Yasası ile Newton'un İkinci Hareket Yasası kullanılarak simüle edilmiştir. Her bir cismin kütlesi, hızı ve konumu hesaplanarak, bu verilerle cisimlerin hareketlerinin zamanla izlenmesi sağlanır. Cisimler arasındaki kütle çekim kuvvetlerinin hesaplanmasının ardından, her bir cismin hız ve konumları güncellenir ve yörüngeleri çizilir. Ayrıca, projede yer alan roketlerin ek itki kuvvetleri de dikkate alınarak roketlerin hareketleri ve etkileşimleri hesaplanır.

Sonuç olarak, bu proje 2 boyutlu ortamda bulunan üç cismin etkileşimlerini ve yörüngelerini C++ dili kullanarak simüle etmektedir.

2. PROJE İÇİNDEKİ DOSYALAR

2.1. Sınıf Tanıtım Dosyaları (.h)

2.1.1. “vektor.h”

```
class vektor{
public:
    // constructors
    vektor(double,double);
    vektor();

    // ozel uyelere erisim icin get fonk.lari
    double getX() const;
    double getY() const;

    // ozel uyelere deger atabilmek icin set fonk.lari
    void setX(double);
    void setY(double);

    // vektorlerin toplanma operatoru
    vektor operator+(const vektor& diger) const;

    // vektorlerin cikarilma operatoru
    vektor operator-(const vektor& diger) const;

    // vektorun bir skalarla carpilma operatoru
    vektor operator*(double skalar) const;

    // vektorun bir skalara bolunme operatoru
    vektor operator/(double skalar) const;

    // vektor buyuklugu hesaplama fonk.u
    double vektor_buyuklugu() const;

    // normalize etmek icin fonk.
    vektor normalize() const;

    // nokta carpimi fonk.u
    double dot(const vektor& other) const;

    // nesneyi yazdirmak icin fonk.
    void print() const;

private:
    double x, y;
};
```

2.1.2. “cisim.h”

```
#include <cmath>
#include "vektor.h"

class cisim{
public:
    // constructor
    cisim(double kutle, const vektor& konum , const vektor& hiz);

    // sanal destructor
    virtual ~cisim() {}

    // ozel uyelere erisebilmek icin
    double getKutle() const;
    vektor getKonum() const;
    vektor getHiz() const;

    // kuvvet eklemek icin fonk.
    void kuvvetEkleme(const vektor& kuvvet);

    // zaman adimi icin sabit kucuk degerli deltaT
    static constexpr double deltaT = 0.01;

    // hiz, ivme, konum, kuvvet icin guncellemeler; sanal
    virtual void guncelleme();

    // yazdirmak icin
    virtual void print() const;

protected:
    double kutle;
    vektor konum;
    vektor hiz;
    vektor netKuvvet;
};
```

2.1.3. “roket.h”

```
#include "vektor.h"
#include "cisim.h"

class roket : public cisim{
public:
    // constructor hem cisim hem roket uyeleri icin
    roket(double kutle, double yakitKG, double Vp, double wp, const vektor&
konum, const vektor& hiz);
```

```
    // yakit kontrolu, yakit kutlesi, toplam kutle,
    // ivme, hiz, konum, net kuvvet guncellemeleri
    void guncelleme(); // override

    // yazdirma
    void print() const; // override

private:
    double Vp; // puskurtme hizi
    double wp; // birim zamanda puskurtulen gazin kutlesi
    double yakitKG; // roketteki yakit miktari
};
```

2.1.4. "etkilesim.h"

```
#include <vector>
#include "cisim.h"
#include "roket.h"

class etkilesim {
public:
    // constructor
    etkilesim();
    // destructor
    ~etkilesim();

    // cisim ekleme
    void cisimEkleme(int indeks, cisim* yeniCisim);

    // kuvvet hesaplaari ve cisim guncellemeleri
    void guncelleme();

    // konum bilgisi alma
    vektor getKonum(int indeks) const;

    // yazdirma
    void print() const;

private:
    static const int cisimSayisi = 3; // 3 cisim problemi
    cisim* cisimler[cisimSayisi];
};
```

2.1.5. "canvas.h"

Bize verilen "canvas.cpp" dosyası header ve cpp dosyası olarak ikiye bölünmüştür.

2.2. Sınıf Üyelerinin Tanımlanma Dosyaları:

2.2.1. “vektor.cpp”

```cpp
#include <iostream>
#include "vektor.h"
// constructors
vektor::vektor()
    : x(0.0), y(0.0) {}

vektor::vektor(double xValue = 0.0, double yValue = 0.0)
    : x(xValue), y(yValue) {}

// get
double vektor::getX() const
    { return x; }
double vektor::getY() const
    { return y; }

// set
void vektor::setX(double xValue)
    { this->x = xValue; }
void vektor::setY(double yValue)
    { this->y = yValue; }

// operators
vektor vektor::operator+(const vektor& toplanacak) const
    { return vektor(x + toplanacak.x, y + toplanacak.y); }

vektor vektor::operator-(const vektor& cikarilacak) const
    { return vektor(x - cikarilacak.x, y - cikarilacak.y); }

vektor vektor::operator*(double skalarSayi) const
    { return vektor(x * skalarSayi, y * skalarSayi); }

vektor vektor::operator/(double skalarSayi) const
    { return vektor(x / skalarSayi, y / skalarSayi); }

// func.s
double vektor::vektor_buyuklugu() const
    { return std::sqrt(x * x + y * y); }

vektor vektor::normalize() const{
    double buyukluk = vektor_buyuklugu();
    if (buyukluk > 0) {
        return vektor(x / buyukluk, y / buyukluk);
    } else {
        return vektor(0.0, 0.0);
    }
```

```
    }

double vektor::dot(const vektor& digeri) const
    { return x * digeri.x + y * digeri.y; }

void vektor::print() const
    {  std::cout << "(" << x << ", " << y << ")\n"; }
```

2.2.2. “cisim.cpp”

```
#include <iostream>
#include "cisim.h"
#include "vektor.h"

// constructor
cisim::cisim(double kutle, const vektor& konum , const vektor& hiz)
    : kutle(kutle), konum(konum), hiz(hiz), netKuvvet(0.0, 0.0) {}

// get func.
double cisim::getKutle() const
    { return kutle; }

vektor cisim::getKonum() const
    { return konum; }

vektor cisim::getHiz() const
    { return hiz; }

// func.s
void cisim::kuvvetEkleme(const vektor& kuvvet)
    { netKuvvet = netKuvvet + kuvvet; }


void cisim::guncelleme(){
    vektor ivme = netKuvvet / kutle;

    // v(t+1) = v(t) + a.dt
    hiz = hiz + ivme * deltaT;

    // x(t+1) = x(t) + v.dt
    konum = konum + hiz * deltaT;

    // sifirlama
    netKuvvet = vektor(0.0, 0.0);
}

void cisim::print() const{
    std::cout << "Kutle: " << kutle << ", Konum: ";
```

```
    konum.print();
    std::cout << "Hiz: ";
    hiz.print();
    std::cout << "\n";
}
```

2.2.3. “roket.cpp”

```
#include <iostream>

#include "roket.h"
#include "vektor.h"
#include "cisim.h"

// constructor
roket::roket(double kutle, double yakitKG, double Vp, double wp, const vektor&
konum, const vektor& hiz)
    : cisim(kutle, konum, hiz),
      Vp(Vp), wp(wp), yakitKG(yakitKG) {}

// func.s
void roket::guncelleme(){
    // itme kuvveti
    vektor itmeK(0.0, 0.0);
    if(yakitKG > 0){
        double itmeKBuyuklugu = Vp * wp;
        itmeK = vektor (itmeKBuyuklugu, 0.0);

        yakitKG -= wp * deltaT;
        if(yakitKG < 0) { yakitKG = 0; }
    }
    // itme kuvvetinin eklenmis hali
    netKuvvet = netKuvvet + itmeK;

    // kalan yakit kutlesinin eklenmis hali
    double toplamKutle = kutle + yakitKG;

    vektor ivme = netKuvvet / toplamKutle;

    // v(t+1) = v(t) + a.dt
    hiz = hiz + ivme * deltaT;

    // x(t+1) = x(t) + v.dt
    konum = konum + hiz * deltaT;

    // sifirlama
    netKuvvet = vektor(0.0, 0.0);
}
```

```
void roket::print() const{
    std::cout << "Kutle + Kalan Yakit Kutlesi: " << (kutle + yakitKG)
              << ", Konum: ";
    konum.print();
    std::cout << "Hiz: ";
    hiz.print();
    std::cout << "Kalan Yakit Miktari:" << yakitKG << "\n";
    std::cout << "\n";
}
```

2.2.4. "etkilesim.cpp"

```
#include <iostream>
#include "etkilesim.h"
#include "roket.h"
#include "vektor.h"
#include "cisim.h"

const double G = 1.0;

etkilesim::etkilesim()
{
    for(int i=0; i<cisimSayisi; i++){
        cisimler[i] = nullptr;
    }
}

etkilesim::~etkilesim(){
    for(int i=0; i<cisimSayisi; i++){
        delete cisimler[i];
    }
}

void etkilesim::cisimEkleme(int indeks, cisim* yeniCisim){
        if (indeks >= 0 && indeks < cisimSayisi) {
        delete cisimler[indeks]; // eski cisim
        cisimler[indeks] = yeniCisim; // yeni cisim ekleme
    } else {
        std::cout << "3 cisim eklendi, daha fazla cisim eklenemez.\n";
    }
}

void etkilesim::guncelleme(){
    // her cisim icin net kuvvet hesaplama
    for (int i = 0; i < cisimSayisi; ++i) {
        for (int j = 0; j < cisimSayisi; ++j) {
            if (i != j) {
                // cisim i'nin cisim j'ye uyguladigi
                // kutle cekim kuvveti (kck) hesabi
```

```
                vektor kck = cisimler[j]->getKonum() - cisimler[i]-
>getKonum();
                double uzaklik = kck.vektor_buyuklugu();
                if(uzaklik > 0.000001){ // cakisma onleyici
                    double kuvvetBuyuklugu = G * (cisimler[i]->getKutle() *
cisimler[j]->getKutle()) / (uzaklik * uzaklik); // yonsuz
                    vektor kuvvet = kck.normalize() * kuvvetBuyuklugu; //
yonlu
                    cisimler[i]->kuvvetEkleme(kuvvet);
                }
            }
        }
    }
    // roketlerin ek itme kuvveti
    for (int i = 0; i < cisimSayisi; ++i) {
        roket* r = dynamic_cast<roket*>(cisimler[i]); // eger cisim bir
roketse
        if (r) {
            vektor itmeKuvveti = r->itmeKuvveti();
            cisimler[i]->kuvvetEkleme(itmeKuvveti);
        }
    }

    // tum cisimlerin guncellenmesi
    for(int i=0; i<cisimSayisi; i++){
        cisimler[i]->guncelleme();
    }
}

vektor etkilesim::getKonum(int indeks) const{
    if (indeks >= 0 && indeks < cisimSayisi && cisimler[indeks]) {
            return cisimler[indeks]->getKonum();
    }else
        return vektor(0.0, 0.0); // hatali indeks icin varsayim
}

void etkilesim::print() const{
    for(int i=0; i<cisimSayisi; ++i){
        cisimler[i]->print();
    }
}
```

2.2.5. “canvas.cpp”

Bize verilen “canvas.cpp” dosyası header ve cpp dosyası olarak ikiye bölünmüştür.

2.3. Proje Tanımına Ek Olarak Eklenen Özelliklerin Bulunduğu “int main()” İçeren Dosya

2.3.1. “ekozellikler.cpp”

```
#include <iostream>
#include <cmath>
#include <algorithm>

#include "canvas.h"
#include "cisim.h"
#include "etkilesim.h"
#include "roket.h"
#include "vektor.h"

// mesafe hesabi icin fonk.
double mesafe(const vektor& p1, const vektor& p2) {
    // d = sqrt((x2 - x1)^2 + (y2 - y1)^2)
    return std::sqrt(std::pow(p2.getX() - p1.getX(), 2) + std::pow(p2.getY() -
p1.getY(), 2));
}

// hangi konfig.
std::string konfigBelirle(const vektor& p1, const vektor& p2, const vektor&
p3) {
    double d1 = mesafe(p1, p2); // Kenar uzunluğu d1
    double d2 = mesafe(p2, p3); // Kenar uzunluğu d2
    double d3 = mesafe(p3, p1); // Kenar uzunluğu d3

    double kenarlar[] = {d1, d2, d3};
    std::sort(kenarlar, kenarlar + 3);

    // 3-4-5 ucgeni kontrolu -> Barrau
    // kenar uzunluklari orani: d1/d2 = 3/4 ve d2/d3 = 4/5
    if (std::abs(kenarlar[0] / kenarlar[1] - 3.0 / 4.0) < 1e-5
    && std::abs(kenarlar[1] / kenarlar[2] - 4.0 / 5.0) < 1e-5){
        return "Barrau konfigurasyonu ile uyumlu.";
    }

    // dogrsallik kontrol -> Euler
    // (y2-y1)/(x2-x1) = (y3-y2)/(x3-x2)
    if (std::abs((p2.getY() - p1.getY()) * (p3.getX() - p2.getX()) -
                 (p3.getY() - p2.getY()) * (p2.getX() - p1.getX())) < 1e-6) {
        return "Euler konfigurasyonu ile uyumlu.";
    }

    return "uygun bir konfigurasyon bulunamadi.";
}

int main() {
```

```
    canvas graphic("ekOzellikli3CisimProblemi");
    graphic.startDoc();
    graphic.drawFrame();

    // kullanicidan giriş almak icin
    double mass1, mass2, mass3;
    double x1, y1, vx1, vy1;
    double x2, y2, vx2, vy2;
    double x3, y3, vx3, vy3;

    std::cout << "Birinci cisim icin (kutle, konum_x, konum_y, hiz_vx,
hiz_vy): ";
    std::cin >> mass1 >> x1 >> y1 >> vx1 >> vy1;

    std::cout << "Ikinci cisim icin (kutle, konum_x, konum_y, hiz_vx, hiz_vy):
";
    std::cin >> mass2 >> x2 >> y2 >> vx2 >> vy2;

    std::cout << "Ucuncu cisim icin (kutle, konum_x, konum_y, hiz_vx, hiz_vy):
";
    std::cin >> mass3 >> x3 >> y3 >> vx3 >> vy3;

    etkilesim uzay;
    uzay.cisimEkleme(0, new cisim(mass1, vektor(x1, y1), vektor(vx1, vy1)));
    uzay.cisimEkleme(1, new cisim(mass2, vektor(x2, y2), vektor(vx2, vy2)));
    uzay.cisimEkleme(2, new cisim(mass3, vektor(x3, y3), vektor(vx3, vy3)));

    // konfig var mi kontrol
    std::string konfig = konfigBelirle(uzay.getKonum(0), uzay.getKonum(1),
uzay.getKonum(2));
    std::cout << "Sectiginiz degerler " << konfig << "\n";


    // simulasyon uzunlugu
    const int simUzunlugu = 10000;

    // her 10 uzunlukta bir cizim yapilsin
    const int cizimAraligi = 10;

    // Simülasyonu başlat ve her adımda cisimlerin pozisyonunu çiz
    for (int t = 0; t < simUzunlugu; ++t) {
        uzay.guncelleme();

        if(t % cizimAraligi == 0){

            for(int i=0; i<3; ++i){
            vektor yer = uzay.getKonum(i);

            if (i == 0) {
```

```
                graphic.drawPoint(yer.getX(), yer.getY(), "red", 2);
            } else if (i == 1) {
                graphic.drawPoint(yer.getX(), yer.getY(), "green", 2);
            } else {
                graphic.drawPoint(yer.getX(), yer.getY(), "blue", 2);
            }

            }
        }
    }

    graphic.finishDoc();
    return 0;
}
```

2.4. Proje Tanımından Başka Özelliği Olmayan “int main()” İçeren Dosya

2.4.1. “main.cpp”

```
#include <iostream>

#include "canvas.h"
#include "cisim.h"
#include "etkilesim.h"
#include "roket.h"
#include "vektor.h"

int main() {
    std::cout << "Simulasyon basladi.\n";

    // canvas'ta tanimlanmis                centerX = centerY = 500
    canvas graphic("3CisimProblemi", 1, 1, centerX, centerY);
    graphic.startDoc();
    graphic.drawFrame();

    // cisim cisim1(1000, vektor(-200, 0), vektor(0, 2));
    // cisim cisim2(1500, vektor(200, 0), vektor(0, -2));
    // cisim cisim3(2000, vektor(0, 300), vektor(-1.5, 0));

    // etkilesim uzay;
    // uzay.cisimEkleme(0, &cisim1);
    // uzay.cisimEkleme(1, &cisim2);
    // uzay.cisimEkleme(2, &cisim3);

    etkilesim uzay;
    uzay.cisimEkleme(0, new cisim(1000, vektor(-200, 0), vektor(0, 2)));
    uzay.cisimEkleme(1, new cisim(1500, vektor(200, 0), vektor(0, -2)));
    uzay.cisimEkleme(2, new cisim(2000, vektor(0, 300), vektor(-1.5, 0)));
    // uzay.cisimEkleme(2, new roket(500, 100, 3000, 1, vektor(0, 200),
vektor(0.05, -0.05)));

    // simulasyon uzunlugu
    const int simUzunlugu = 2000;

    // her 10 uzunlukta bir cizim yapilsin
    const int cizimAraligi = 10;

    // Simülasyonu başlat ve her adımda cisimlerin pozisyonunu çiz
    for (int t = 0; t < simUzunlugu; ++t) {
        uzay.guncelleme();

        if(t % cizimAraligi == 0){

            for(int i=0; i<3; ++i){
```

```
            vektor yer = uzay.getKonum(i);

            if (i == 0) {
                graphic.drawPoint(yer.getX(), yer.getY(), "red", 2);
            } else if (i == 1) {
                graphic.drawPoint(yer.getX(), yer.getY(), "green", 2);
            } else {
                graphic.drawPoint(yer.getX(), yer.getY(), "blue", 2);
            }

            }
        }
    }

    graphic.finishDoc();
    return 0;
}
```

2.5. Her Sınıfın Kontrolü İçin Yazılan “int main()” İçeren Dosyalar

2.5.1. “cisimKontrol.cpp”

```
#include <iostream>
#include "canvas.h"
#include "cisim.h"
#include "etkilesim.h"
#include "roket.h"
#include "vektor.h"

/*
// cisim sinifi kontrol
int main() {
    cisim earth(1000, vektor(-200, 0), vektor(0, 2));

    earth.kuvvetEkleme(vektor(10.0, 20.0));

    earth.guncelleme();

    earth.print();
    // terminal:
    // Kutle: 1000, Konum: (-200, 0.020002)
    // Hiz: (0.0001, 2.0002)

    return 0;
}
*/
```

2.5.2. “roketKontrol.cpp”

```
#include <iostream>
#include "canvas.h"
#include "cisim.h"
#include "etkilesim.h"
#include "roket.h"
#include "vektor.h"
/*
// roket sinifi kontrol
int main() {

    roket roket(500, 100, 3000, 100, vektor(0.0, 0.0), vektor(0.0, 0.0));

    roket.kuvvetEkleme(vektor(0.0, -10 * 500));

    for (int i = 0; i < 10; ++i) {
        roket.guncelleme();
        roket.print();
    }

// terminal:
```

```
// Kutle + Kalan Yakit Kutlesi: 599, Konum: (0.0500835, -0.000834725)
// Hiz: (5.00835, -0.0834725)
// Kalan Yakit Miktari:99

// Kutle + Kalan Yakit Kutlesi: 598, Konum: (0.150334, -0.00166945)
// Hiz: (10.0251, -0.0834725)
// Kalan Yakit Miktari:98

// Kutle + Kalan Yakit Kutlesi: 597, Konum: (0.300836, -0.00250417)
// Hiz: (15.0502, -0.0834725)
// Kalan Yakit Miktari:97

// Kutle + Kalan Yakit Kutlesi: 596, Konum: (0.501674, -0.0033389)
// Hiz: (20.0838, -0.0834725)
// Kalan Yakit Miktari:96

// Kutle + Kalan Yakit Kutlesi: 595, Konum: (0.752931, -0.00417362)
// Hiz: (25.1258, -0.0834725)
// Kalan Yakit Miktari:95

// Kutle + Kalan Yakit Kutlesi: 594, Konum: (1.05469, -0.00500835)
// Hiz: (30.1763, -0.0834725)
// Kalan Yakit Miktari:94

// Kutle + Kalan Yakit Kutlesi: 593, Konum: (1.40705, -0.00584307)
// Hiz: (35.2353, -0.0834725)
// Kalan Yakit Miktari:93

// Kutle + Kalan Yakit Kutlesi: 592, Konum: (1.81008, -0.0066778)
// Hiz: (40.3029, -0.0834725)
// Kalan Yakit Miktari:92

// Kutle + Kalan Yakit Kutlesi: 591, Konum: (2.26387, -0.00751252)
// Hiz: (45.379, -0.0834725)
// Kalan Yakit Miktari:91

// Kutle + Kalan Yakit Kutlesi: 590, Konum: (2.7685, -0.00834725)
// Hiz: (50.4638, -0.0834725)
// Kalan Yakit Miktari:90
    return 0;
}
*/
```

3. SINIFLARIN GÖREVLERİ

3.1. Vektör Sınıfı

Projede kullanılan ‘vektor’ sınıfı, iki boyutlu bir vektörün matematiksel modellemesini sağlar. Bu sınıf, fiziksel simülasyonlarda cisimlerin konum ve hız bilgilerini temsil etmek için tasarlanmıştır. ‘vektor’ sınıfı, temel matematiksel işlemleri destekleyen çeşitli operatör aşırı yüklemeleri bulundurur. Bunlar arasında vektörlerin toplanması, çıkarılması, bir skalar ile çarpılması veya bölünmesi işlemleri yer alır. Ayrıca, vektör büyüklüğünün hesaplanması, birim vektör elde edilmesi (normalize) ve iki vektör arasındaki nokta çarpımının hesaplanması gibi fonksiyonlar da mevcuttur. Sınıfın kapsülleme ilkesi doğrultusunda, vektörün x ve y bileşenlerine doğrudan erişim yerine get ve set fonksiyonları kullanılmıştır. Bu yaklaşım, hem güvenli bir veri yönetimi sağlar hem de sınıfın dışarıdan kullanımını kolaylaştırır. ‘vektor’ sınıfı, proje boyunca kullanılan diğer sınıflar için temel yapı taşı oluşturur.

3.2. Cisim Sınıfı

‘cisim’ sınıfı, üç cisim problemi simülasyonunda nesneleri temsil eden temel sınıftır. Bu sınıf, bir cismin kütlesini, konumunu, hızını ve kendisine etki eden toplam kuvveti modellemek amacıyla tasarlanmıştır. Kütle, cismin fiziksel özelliklerini belirlerken; konum ve hız vektörleri cismin uzaydaki hareketini ve dinamiklerini ifade eder. Kuvvet vektörü ise, cismin diğer cisimlerle olan etkileşimleri sonucunda üzerine etki eden toplam kuvveti temsil eder. Sınıfın işlevselliği, kullanıcıya cisimlerin durumunu güncelleme, fiziksel parametrelerini değiştirme ve dış etkenlere göre tepki verme gibi imkanlar verir. Bu sayede, üç cisim probleminin sürekli olan yapısına uygun bir şekilde simülasyon gerçekleştirilmesi sağlanır.

3.3. Roket Sınıfı

‘roket’ sınıfı, üç cisim problemi simülasyonunda, roketin kütlesini, konumunu, hızını ve itme kuvvetini modellemek için tasarlanmıştır. Roketler, belirli bir başlangıç pozisyonundan hareket ederken, etkileşimde bulunduğu cisimlerin kütleçekim etkilerini dikkate alır ve bu doğrultuda hareketlerini günceller. İtme kuvveti, roketin hareketini yönlendiren temel bir parametredir.

3.4. Etkileşim Sınıfı

‘etkilesim’ sınıfı, üç cisim problemi simülasyonunda cisimler ve roketler arasındaki dinamik etkileşimleri yönetmek için tasarlanmıştır. Bu sınıf, evrendeki tüm nesnelerin pozisyonlarını, hızlarını ve birbirlerine uyguladıkları kuvvetleri dikkate alarak hareketlerini günceller. ‘etkilesim’ sınıfı, kütleçekim kuvveti hesaplamaları, roketlerin itme kuvvetine bağlı olarak yönlendirilmesi ve sistemin genel dengesinin sağlanması gibi görevleri yerine getirir. Ayrıca, kullanıcı tarafından sağlanan başlangıç verilerini işleyerek simülasyonun doğru bir şekilde başlatılmasını sağlar. Bu sınıf, sistemin zamana bağlı olarak değişimini hesaplamak için gereken işlemleri içerir. ‘etkilesim’ sınıfı, tüm simülasyonun asıl gerçekleştiği kısımdır.

4. PROJEYİ ÇALIŞTIRMA

Bu proje CMake kullanılarak yapılandırılmıştır. İlk önce proje yapısına aşağıdaki listeden ulaşabilirsiniz, sonra ise projeyi nasıl çalıştırabileceğinizi görebilirsiniz.

4.1. Proje Yapısı

- ThreeBodyProblem/
  - CMakeLists.txt — CMake yapılandırma dosyası
  - src/ — Projeye ait kaynak dosyaları (cpp dosyaları)
    - canvas.cpp — Grafik işlemleri ve ekranda görselleştirme
    - cisim.cpp — Cisim sınıfının implementasyonu
    - cisimKontrol.cpp — 'cisim' sınıfının kontrolü
    - etkilesim.cpp — Cisimler arasındaki etkileşimleri hesabı
    - roket.cpp — Roket alt sınıfının implementasyonu
    - roketKontrol.cpp — 'roket' sınıfının kontrolü
    - vektor.cpp — Vektör sınıfının implementasyonu
    - main.cpp — Ana program dosyası
    - ekozellikler.cpp — Euler ve Barrau konfigürasyonuna uygunluk testinin olduğu ana program dosyası
  - include/ — Header dosyaları
    - canvas.h
    - cisim.h — Cisim sınıfının tanımı
    - etkilesim.h — Etkileşim sınıfının tanımı
    - roket.h — Roket alt sınıfının tanımı
    - vektor.h — Vektör sınıfının tanımı
  - build/ — Derleme çıktılarının bulunduğu klasör
  - README.md — Açıklamalar ve kullanım talimatları

4.2. Projenin Çalıştırılması

4.2.1. Projenin Yoluna Gitme:

İlk olarak, projenin bulunduğu dizine gidilmelidir. Aşağıdaki komutlarla ThreeBodyProblem projesine erişebilirsiniz:

- Linux:

  cd /path/to/ThreeBodyProblem

- Windows (CMD):

  cd C:\path\to\ThreeBodyProblem

4.2.2. Projenin CMake ile Yapılandırılması:

Projenin çalıştırılabilmesi için, proje dizini içerisindeki build klasörünü kullanarak CMake ile yapılandırma yapılmalıdır. şu adımlar takip edilmelidir:

- Linux ve Windows:

  cd build

  cmake ..

4.2.3. Projenin Derlenmesi:

Projeyi derlemek için şu komut kullanılır:

- Linux:

  make

- Windows (CMD):

  cmake--build .

4.2.4. Projenin Çalıştırılması:

Derleme başarıyla tamamlandıktan sonra, projeyi çalıştırmak için aşağıdaki komutlar kullanılır:

- Linux:

./ThreeBodyProblem

- Windows (CMD):

ThreeBodyProblem.exe

5. EKLER

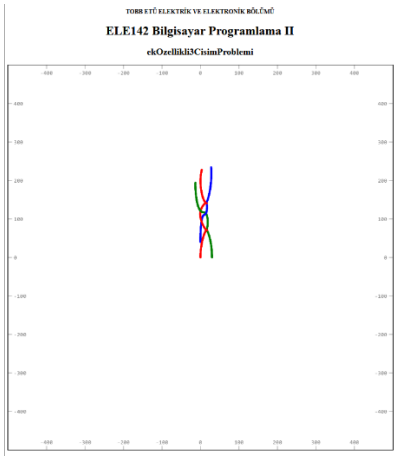
5.1. “ekozellikler.cpp” Dosyası Ana Program Dosyası İken

Bu bölümde ‘ekozellikler.cpp’ dosyasına göre terminalden girilen 3 cisim (roket yok) değerlerine göre bulunan çıktılar verilmektedir.

- Barrau Konfigürasyonu ile:

Terminal:

```
edvin@edi:~/workspaces/cpp_workspace/ThreeBodyProblem/build$
./ThreeBodyProblem
Birinci cisim icin (kutle, konum_x, konum_y, hiz_vx, hiz_vy): 10 0 0 0 2
Ikinci cisim icin (kutle, konum_x, konum_y, hiz_vx, hiz_vy): 20 30 0 0 2
Ucuncu cisim icin (kutle, konum_x, konum_y, hiz_vx, hiz_vy): 30 0 40 0 2
Sectiginiz degerler Barrau konfigurasyonu ile uyumlu.
```



Şekil 1 Barrau Konfigürasyonu Örneği Çıktısı

- Euler Konfigürasyonu ile:

Terminal:

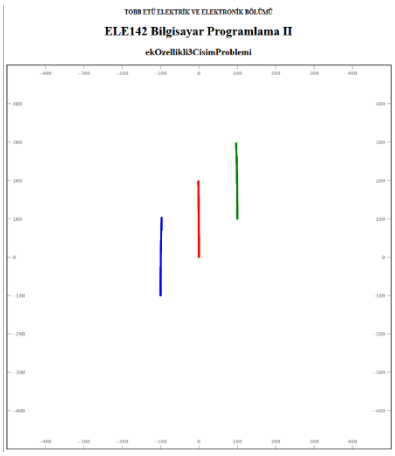
edvin@edi:~/workspaces/cpp_workspace/ThreeBodyProblem/build$ ./ThreeBodyProblem

Birinci cisim icin (kutle, konum_x, konum_y, hiz_vx, hiz_vy): 10 0 0 0 2

Ikinci cisim icin (kutle, konum_x, konum_y, hiz_vx, hiz_vy): 20 100 100 0 2

Ucuncu cisim icin (kutle, konum_x, konum_y, hiz_vx, hiz_vy): 30-100-100 0 2

Sectiginiz degerler icin Euler konfigurasyonu ile uyumlu.



Şekil 2 Euler Konfigürasyonu Örneği Çıktısı

5.2. “main.cpp” Dosyası Ana Program Dosyası İken

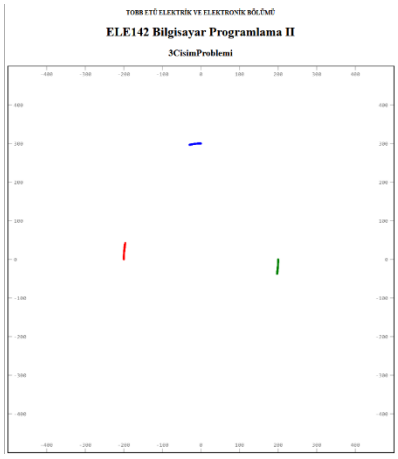
- “main.cpp” dosyasındaki cisimlerin değerleri:

```
uzay.cisimEkleme(0, new cisim(1000, vektor(-200, 0), vektor(0, 2)));
uzay.cisimEkleme(1, new cisim(1500, vektor(200, 0), vektor(0, -2)));
uzay.cisimEkleme(2, new cisim(2000, vektor(0, 300), vektor(-1.5, 0)));
```

Terminal:

edvin@edi:~/workspaces/cpp_workspace/ThreeBodyProblem/build$ ./ThreeBodyProblem

Simulasyon basladi.



Şekil 3 Roketsiz 3 Cisim

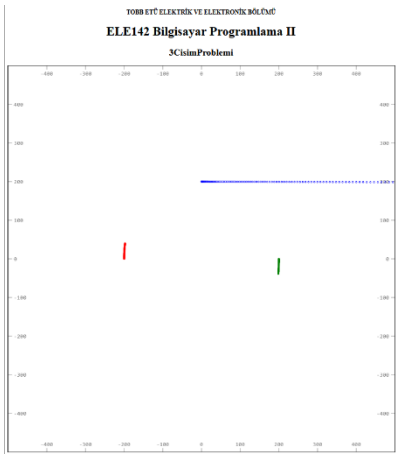
- “main.cpp” dosyasındaki cisimlerin değerleri:

```
uzay.cisimEkleme(0, new cisim(1000, vektor(-200, 0), vektor(0, 2)));
uzay.cisimEkleme(1, new cisim(1500, vektor(200, 0), vektor(0, -2)));
uzay.cisimEkleme(2, new roket(500, 100, 3000, 1, vektor(0, 200), vektor(0.05,
-0.05)));
```

Terminal:

edvin@edi:~/workspaces/cpp_workspace/ThreeBodyProblem/build$ ./ThreeBodyProblem

Simulasyon basladi.



Şekil 4 Biri Roket Olan 3 Cisim