

GEBZE TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

**BİL396 Proje Dersi
BisikleSim Projesi Raporu**

141044012 - M. Yusuf Koçer

141044083 - Esra Sert

151044065 - Zeliha Erim

151044070 - Pınar Erdem

161044083 - Galip Tayfun Saygılı

171044003 - Elif Goral

171044010 - Dilara Karakaş

171044057 - Doruk Ergün

171044070 - Haktan Bilgehan Dilber

171044098 - Akif Kartal

1801042251 - Elif Akgün

1801042628 - Furkan Çelen

1801042663 - Mert Can Beşirli

Grup: 12

Danışman: Prof. Dr. Erkan ZERGEROĞLU

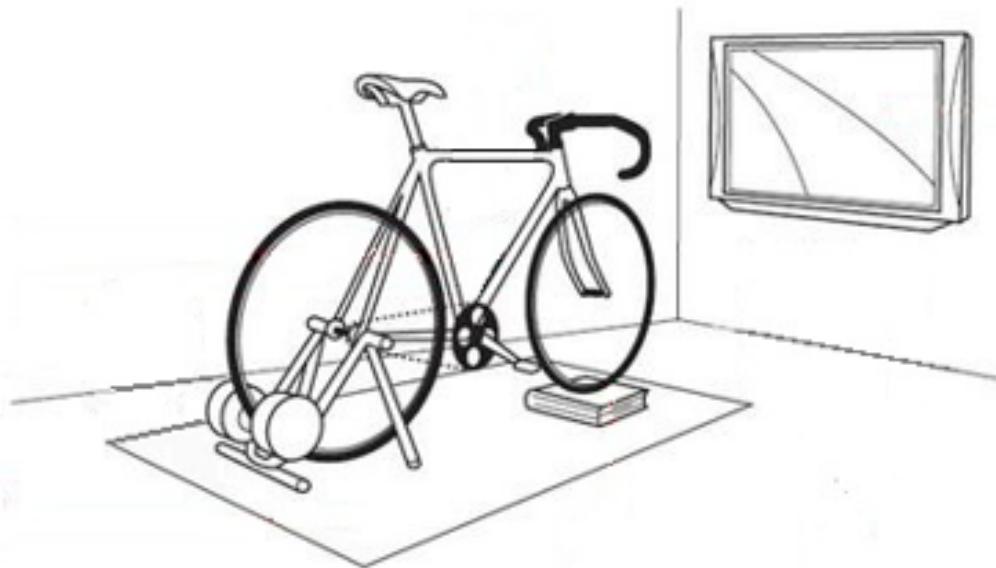
Mayıs 2021

1. Projenin Tanımı ve Amacı

Bu projede belirli bir haritada herhangi bir bisiklet kullanarak, bisiklet kullanma deneyeminin simüle edilmesi hedeflenmiştir. Bu simülasyonda kullanıcı, bisikletin hızını ve yönünü kontrol edebilecektir. Ayrıca kullanıcı simülasyonda nabız değerini görebilicektir. Projenin donanım kısmında mikro işlemci, nabız sensörü, hız sensörü, yön sensörü kullanılacak ve bu bilgiler işlenmesi için projenin yazılım kısmına aktarılacak. Projenin yazılım kısmında 3 boyutlu bir simülasyon ortamında bisikletimiz, bir harita ve çevre simüle edilecek, kullanıcıya ev ortamında bisiklet sürme deneyimi sunulacak.

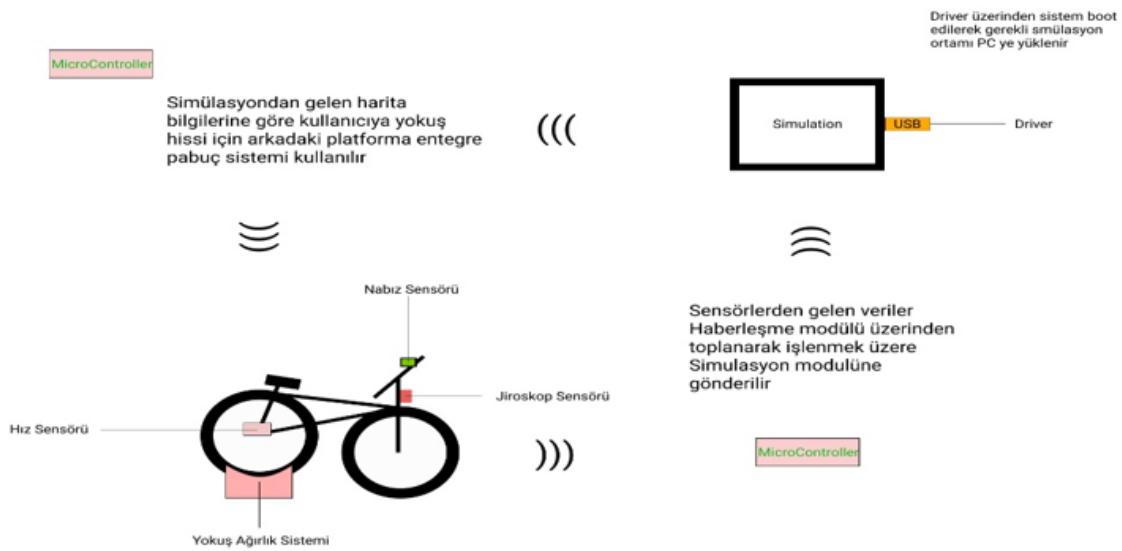
1.1 Konsept Tasarımı

Figure 1: Konsept Tasarımı



1.2 Çalışma Akış Diyagramı

Figure 2: Konsept Tasarımı



2. Modüller

2.1 Donanım Modülü

Figure 3: Bisiklet



*Sensör testlerini izlemek için buraya tıklayın.
BisikleSim tanıtım videosu için buraya tıklayın.*

2.1.1 SERVO MOTOR İLE SÜRÜŞ AĞIRLAŞTIRMA SİSTEMİ

Bisiklet simülasyonu projemizde sürüs hissinin maksimum seviyede yaşanabilmesi için sisteme entegre ettiğimiz bir frenleme sistemi mevcuttur. Bu sistem bir adet servo motor ve bir adet bisiklet fren pabuçundan oluşmaktadır. STM32F4 microcontroller üzerinden PWM sinyalleri ile kontrol ettiğimiz servo motor, Simülasyon tarafında tasarlanan haritada eğimli bir yola girildiğinde USART haberleşme üzerinden aldığı açıya göre fren pabuçlarının tekerleğin jantlarına dokunmasını veya ayrılmamasını sağlayarak kullanıcıya bisikletin yokuş çıkarken ağırlaşması hissini yaşatmak için tasarlanmıştır.

Figure 4

```
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
_HAL_UART_ENABLE_IT(&huart, UART_IT_RXNE);
```

Microcontroller üzerinde gerekli ayarları yapıp PWM için bir timer atadık ve Usart haberleşme için interrupt ayarlarını aktif ettik.

Figure 5

```
void USART1_IRQHandler(void)
{
    /* USER CODE BEGIN USART1_IRQn 0 */
    HAL_UART_Receive(&huart1, (uint8_t*)RxByte, 1, 100);
    /* USER CODE END USART1_IRQn 0 */
    HAL_UART_IRQHandler(&huart1);
    /* USER CODE BEGIN USART1_IRQn 1 */
    RxData[RxDataCnt++]=RxByte[0];
    if(RxDataCnt>=3){
        RxFlag=1;
        RxDataCnt=0;
    }
    /* USER CODE END USART1_IRQn 1 */
}
```

Simülasyondan veri geldiğinde sisteme kesme sinyali olarak düşmekte ve gelen veri RxFlag 1 olduğunda değişken içerisinde saklanıp frenleme fonksiyonuna gönderilmemektedir. Gelen açıya göre PWM sinyali ile servo kontrol edilmektedir.

Figure 6

```
void frenServo(int angle){

    if(angle>90){
        __HAL_TIM_SetCompare(&htim1,TIM_CHANNEL_1,135);
    }
    else{
        __HAL_TIM_SetCompare(&htim1,TIM_CHANNEL_1,45);
    }

}
```

2.1.2 POTANSİYOMETRE SENSÖRÜ

Bisikletin dönüşlerini algılamak için direksiyonun dönüşüne bağlı olarak hareket eden bir potansiyometre kullanılmıştır.

Figure 7

```
while(TM_MPU6050_Init(&MPU6050_Sensor, TM_MPU6050_Device_0, TM_MPU6050_Accelerometer_8G, TM_MPU6050_Gyroscope_250s)
```

Usart üzerinden simülasyon ortamına aktarılıp 3 boyutlu bisikletin harita dönüş hareketleri yapması sağlanmaktadır.

Figure 8

```
TM_MPU6050_ReadAll(&MPU6050_Sensor);
ax=MPU6050_Sensor.Accelerometer_X;
ay=MPU6050_Sensor.Accelerometer_X;
az=MPU6050_Sensor.Accelerometer_X;
gx=MPU6050_Sensor.Gyroscope_X;
gy=MPU6050_Sensor.Gyroscope_Y;
gz=MPU6050_Sensor.Gyroscope_Z;
temperature=MPU6050_Sensor.Temperature;
```

2.1.3 NABİZ ÖLÇER SENSÖRÜ

Bisiklet simülasyonumuzda kullanıcıya daha çeşitli deneyim sunabilmek için sisteme bir nabız ölçer sensör ekledik. Parmağa bağlanan sensör kalp atışını algılayarak analog sinyale dönüştürmekte. Bu sinyaller STM32 üzerinden analog okuma yapılarak dijital

veriye dönüştürilmekte. Elde edilen veriler anlamlandırılarak Usart haberleşme ile simülasyon tarafına iletilerek kullanıcıya ekranda nabız bilgileri gösterilmektedir.

Figure 9

```
void readPulse(){
    HAL_ADC_Start(&hadc1);

    if(HAL_ADC_PollForConversion(&hadc1, 1000)==HAL_OK){
        adc_value=HAL_ADC_GetValue(&hadc1);
    }

    HAL_ADC_Stop(&hadc1);
}
```

2.1.4 HIZ ÖLÇER

Bisiklet simülasyonunda hareket edebilmek için, gerçek ortamındaki bisikletimizin tekerlek hızını simülasyona aktarabilmemiz lazım. Bunun için kapı alarmlarına benzer mıknatıslı bir sistem tasarladık. Bunun için bir adet 14 mm reed rôle ile bir adet mıknatısa ihtiyacımız var. Bisikletin jantlarına yerleştirilen mıknatıs her turda röleye denk gelerek sinyal üretmekte, sinyal süresine göre de STM32 üzerinde hesaplamalar yapılarak tur süresi üzerinden ortalama bisiklet hızı tespit edilip Usart haberleşme ile simülasyon tarafına iletilerek bisiklet hareketi sağlanmaktadır.

2.1.5 MALZEMELER

Figure 10: Servo Fren Sistemi-1



Figure 11: Servo Fren Sistemi-2



Figure 12: Potansiyometre

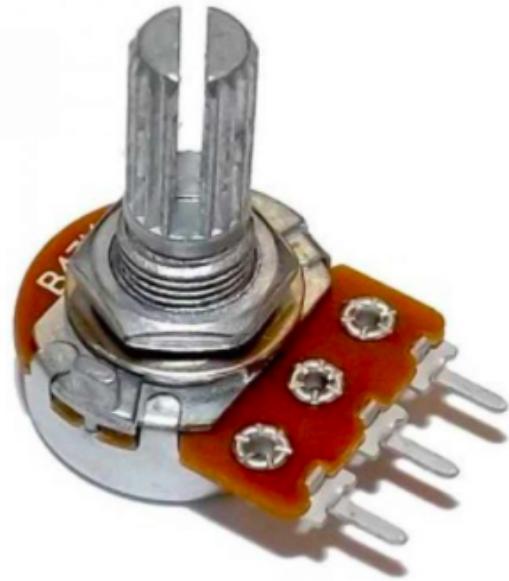


Figure 13: Nabız Ölçer



Figure 14: Hız Ölçer-Röle



Figure 15: Hız Ölçer-Mıknatıs



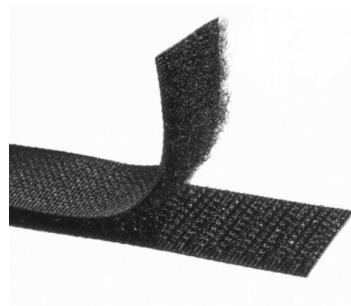
2.2 Mekanik Modülü

Mekanik modülüümüz bir adet bisiklet tutucu Trainer ile sensörleri bisiklete bağlamak için kelepçelerden oluşmaktadır. Arka tekerleği havada tutabilmek için Trainer üzerinde yerlestirdiğimiz bisikletin direksiyonuna bağladığımız potansiyometre sensörü, tekerleğin yanına servo motor, jantın birine mıknatıs ve parmağa çitçitlanacak nabız ölçer sensörü yerleştirilmiştir.

Figure 16



Figure 17



2.3 Simülasyon Modülü

BisikleSim simülasyonunu izlemek için buraya tıklayın.

Öncelikle simülasyon ortamını oluşturmak için hangi programları kullanabileceğimizi araştırdık. Araştırmalarımızın sonucunda Blueprint ve C++ olmak üzere iki tane programlama seçenekleri sunan Unreal Engine 4 kullanmaya karar verdik. Proje gereksinimlerinden dolayı C++ dili ve projeyi geliştirmek için de Visiual Studio IDE'sini kullandık.

2.3.1 MENÜ

BisikleSim iki tane harita içermektedir. Bir haritamızda gerçek dünyayı yansitan bir şehir tasarlarken diğer haritamızda ise yarış pisti konseptini esas aldık. Kullanıcı, haritalar arası geçiş yapabileceği gibi, ESC tuşunu kullanarak bulunduğu haritayı durdurma - devam etme imkanlarına sahiptir.

Figure 18: Menü



Figure 19: Pause



2.3.2 HARITA -1

Harita-1 simülasyonunu izlemek için buraya tıklayın.

Bu haritada, Unreal Engine 4 objelerini ve dışarıdan ek olarak aldığımız fbx formatındaki objeleri kullandık. Bu haritamızı yarış pisti şeklinde tasarladık. Yarış pistinin yollarını Unreal Engine 4 materyalleri ile, haritada kalan diğer alanları ağaç ve taş objeleri ile detaylandırdık. Simülasyon deneyimi esnasında sürücüye gerçek yol deneyimini yaşatabilmek için parkura bazı tümsek ve yokuşlar ekledik. Amacımız birçok viraj ve engelin kullanıldığı bu haritada sürücüye gerçekçi bir deneyim kazandırmak.

Figure 20: Birinci harita-1

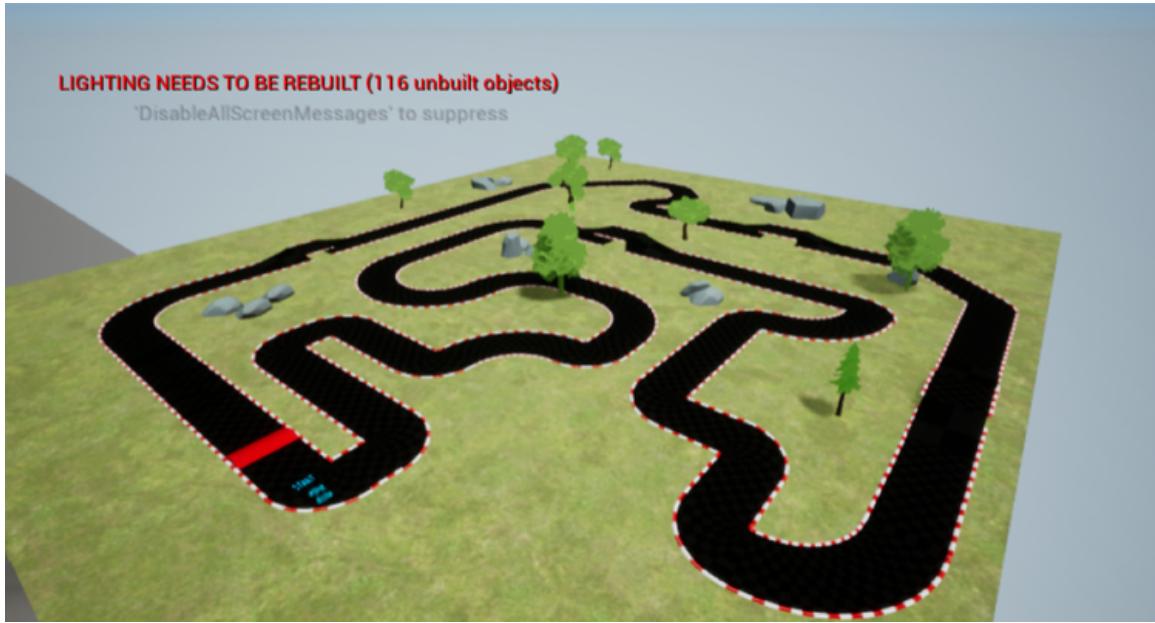
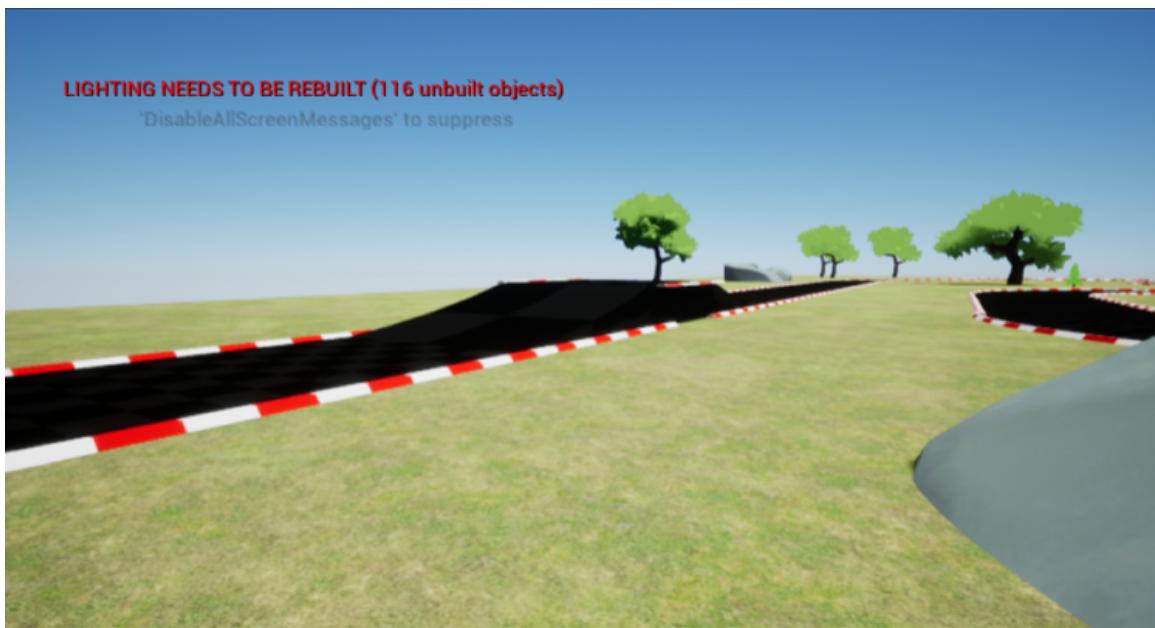


Figure 21: Birinci harita-2



2.3.3 HARITA -2

Harita-2 simülasyonunu izlemek için buraya tıklayın.

Bu haritada kullanılan objeler için Unreal Engine tarafından sağlanan Modular House in Environments paketini kullandık. İlk haritamız yarış pisti şeklinde olduğu için bu haritamızı gerçek dünyayı yansitan bir şehir konsepti olarak tasarladık. Ağaç, ev, araba, yol ve kaldırım objeleri kullanarak gerçek bir dünya görüntüsü oluşturmayı hedefledik. Haritada, bisikletin sınırları aşmaması için ve evlerin, arabaların içine girmesini önlemek için çeşitli sınırlandırmalar kullandık. Ayrıca bisikletin eğimli yol üzerinde tepkisini ölçebilmek için, haritaya eğimli sahip olan yollar ekledik.

Figure 22: İkinci harita-1



Figure 23: İkinci harita-2



2.3.4 BİSİKLET

Bisikleti simüle ederken Unreal Engine'in 4 tekerlekli araç modülüni bisikletimiz için bir destek mekanizması olarak kullandık. Bisiklet arka planda çalışan dört tekerlekli bir aracın boyut ve öznitelikler olarak bisiklete yakınsanmış halini hareket ve çarpışma kontrollerinde kullanmaktadır. Önceki raporumuzda bisikletin statik bir model üzerinde çalıştığını söylemiştim. Yaptığımız geliştirmeler sonucu artık bisiklet kinetik bir model üzerinden çalışmaktadır. Sensörlerden aldığı veriye göre direksiyonu yön değiştirebilmektedir. Ayrıca bisiklet tekerlekleri de gidilen hıza uygun olarak dönmektedir. Bisikletin anlık hızı ekranın üst köşesinden görüntülenebilmektedir. Bisiklet sürüş sırasında iki farklı görüş açısına sahip iki farklı kamera tarafından izlenebilmektedir. Bu kameralardan biri bisikleti süren bir kişinin görüş açısını yansıtırken, ikinci bakış açısı ise bisikletin tamamen görüntülenebildiği bir üçüncü şahıs açısını yansıtmaktadır. Önceki raporumuzda klavye üzerinden yönetilen bisiklet, haberleşme entegrasyonu ile beraber tamamen sensörlerden gelen veri ile çalışmaktadır.

Figure 24: Birinci harita bisiklet ile

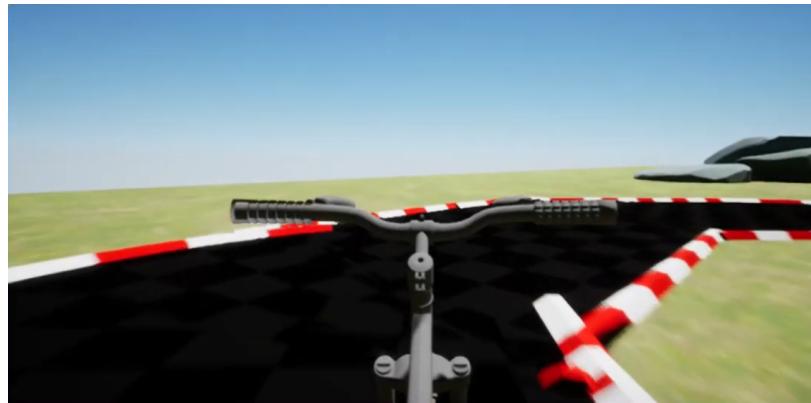


Figure 25: İkinci harita bisiklet ile



2.4 Haberleşme Modülü

Haberleşme modülünü testini izlemek için buraya tıklayın.

Bu modülde donanım tarafında mikroişlemci kartından(STM32f4) gelen verilerin okunup işlenmesi amaçlanmıştır. Ayrıca projemizde yokuş sistemi bulunduğu için haberleşme modülüümüz 2 yönlü çalışmaktadır. Gerçekleştirilmesi amaçlanan haberleşmeler; 1) Donanım tarafından gelen verilerin her saniyede anlık olarak okunup ayırtılması. 2) Simülasyondaki bisiklet haritalardaki yokuşlara geldiğinde donanım kartımıza veri gönderilmesi ve yokuş sisteminin aktifleşerek bisikletin yavaşlatılması.

2.4.1 VERILERIN OKUNMASI VE YAZILMASI

Verileri okumak ve yazmak için UART haberleşme protokolü kullanarak mikroişlemci kart ile haberleşme sağlanmıştır. Bunun için simülasyonda oyun başladığında bilgisayarın COMX portundaki karta bağlanarak veriler okunmaya başlanmaktadır. Daha sonra simülasyon devam ederken eğer herhangi bir yokuşa giriş yapılrsa mikroişlemci kartımıza mesaj gönderilir.

Figure 26: Haberleşme için kullanılan C++ sınıfı

```
#include <windows.h>
#include <iostream>

class SerialPort
{
private:
    HANDLE handler;
    bool connected;
    COMSTAT status;
    DWORD errors;
public:
    explicit SerialPort(const char* portName);
    ~SerialPort();

    int readSerialPort(const char* buffer, unsigned int buf_size);
    bool writeSerialPort(const char* buffer, unsigned int buf_size);
    bool isConnected();
    void closeSerial();
};


```

Yukarıdaki class'da readSerialPort and writeSerialPort metodları kullanılarak mikroişlemci ile haberleşme sağlanmaktadır.

2.4.2 YOKUŞ SİSTEMİNİN AKTİFLEŞTİRİLMESİ

Mekanik modülüümüzdeki yokuş sistemini aktifleştirmek için mikroişlemci kartımıza simülasyon içerisinde sinyal göndermemiz gereklidir. Bunu yapabilmek için Unreal Engine oyun motorunda “TriggerBox” nesnesi kullanılarak yokuş girişleri tespit edilerek sinyal gönderilmektedir.

Figure 27: Trigger Box C++ Class

```
#include "CoreMinimal.h"
#include "UObject.h"
#include "Engine/TriggerBox.h"
#include "MyTriggerBox.generated.h"

UCLASS()
class MAP2_API AMyTriggerBox : public ATriggerBox
{
    GENERATED_BODY()

protected:

    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // constructor sets default values for this actor's properties
    AMyTriggerBox();

    // overlap begin function
    //send a signal to the stm32 card
    UFUNCTION()
    void OnOverlapBegin(class AActor* OverlappedActor, class AActor* OtherActor);

    // overlap end function
    //send a signal again to the stm32 card
    UFUNCTION()
    void OnOverlapEnd(class AActor* OverlappedActor, class AActor* OtherActor);

    // specific actor for overlap
    //bicycle in simulation
    UPROPERTY(EditAnywhere)
    class AActor* SpecificActor;

    //this needs to be public for now
    SerialPort* test; //stm32 card
};

};
```

Bu kod ile yaratılan obje(görünmez küp) simülasyonda yokuşların önüne eklenerek yokuş girişleri tespit edilip mikroişlemci karta mesaj gönderilmektedir.

Figure 28: Mesaj Gönderim Kodları

```
void AMyTriggerBox::OnOverlapBegin(class AActor* OverlappedActor, class AActor* OtherActor)
{
    //if the overlapping actor is the specific actor we identified in the editor
    if (OtherActor && OtherActor != this && OtherActor == SpecificActor)
    {
        if (GEngine)
        {
            //char ledON[] = "Yokus\n";
            //char ledOFF[] = "Normal\n";
            if (!isIn)
            {
                test->writeSerialPort(ledON, strlen(ledON));
                isIn = 1;
            }
            else {
                test->writeSerialPort(ledOFF, strlen(ledOFF));
                isIn = 0;
            }
        }
    }
}
```

2.4.3 VERILERIN OKUNMASI VE KULLANILMASI

Mikroişlemci kartından gelen verilerin okunup değerlendirilmesi için simülasyon kısmında haberleşme kısmından readSerialPort metodu kullanılarak simülasyonda her saniye okuma yapılarak karttan gelen güncel bilgiler programda geçici bir arrayde tutulur daha sonra bu bilgiler delimiter(ayırıcı) karakterler kullanılarak ayırtılı olarak gerekli işlemlerde kullanılırlar. Örneğin aşağıdaki kodda karttan gelen hızlan verisi ile bisikletin hareket etmesi amaçlanmıştır

Figure 29: Mesaj Gönderim Kodları

```
void AMap2Pawn::exampleReceiveData(void)
{
    char signal[] = "hizlan";
    int readResult = stm32->readSerialPort(incomingData, MAX_DATA_LENGTH);
    if (strstr(incomingData, signal))
    {
        MoveForward(1);
    }
    //FString fs(incomingData);
    FString Fs = FString(ANSI_TO_TCHAR(incomingData));
    UE_LOG(LogTemp, Log, TEXT("%s"), *Fs);
}
```

2.4.4 TEST EDİLMESİ

Yukarıda uyguladığımız çözümleri test edebilmek için bir mikroişlemci kartına ihtiyacımız var ancak pandeminden dolayı mikro işlemci kart diğer bir grup arkadaşımızda bulunsa dahi bizler veri alışverişini test etmek bilgisayarımızda sanal bir bluetooth COM portu oluşturarak bu çözümleri telefon üzerinde test ettik.

2.5 Driver Modülü

Driver modülüümüzde işletim sistemi olarak Windows kullandık. Bu seçimimizin iki farklı sebebi var. İlk sebebi Unreal Engine'dan aldığımız export'un Linux sistemlerinde çalıştırılmak istendiğinde Vulkan driverları kaynaklı bir hata almış olmamız. Bu hatayı gidermek için çeşitli paketler kurmuş olmamıza rağmen denemelerimizde farklı hatalarla karşılaştık. İnternet üzerinde yaptığımız araştırmalarda bu sorunun çözümlerinden birinin Linux'ta OpenGL kullanmak olduğunu ancak artık OpenGL'in Unreal Engine tarafından desteklenmediğini öğrendik. Öte yandan Windows üzerinde çalıştırırken DirectX kullandık ve sıkıntı yaşamadık. İkinci bir sebep olarak araştırmalarımızda Unreal Engine'in Linux üzerindeki performansının Windows üzerindeki performansından daha kötü olduğunu gördük. Bu sebeplerden dolayı Windows ile devam ettik.

Bu modülüümüzde bir USB bellek kullandık. USB bellek içerisinde ilk olarak bir işletim sistemi kurulumu gerçekleştirdik. Buradaki amacımız simülasyonumuzun çalıştırılan cihazda herhangi bir hard disk müdahalesına neden olmamasıdır yani işletim sistemi hafıza olarak usb'de kalan kısmı kullanacaktır. İçerisine işletim sistemi kurduğumuz USB bellek içerisinde gerekli driver kurulumları yapılmıştır. Son olarak da simülasyonun .exe kurulumu yapılmıştır. USB bellek takıldırca çalıştırılan bilgisayarın hard diskine herhangi bir yükleme yapılmadan simülasyonumuz çalışmaktadır.

Görevler	Görevliler
Map 1 Harita Çizimi	Esra Sert Elif Goral
Map 2 Harita çizimi	Zeliha Erim Elif Akgün
Simülasyon Menu Ekleme	Elif Goral Zeliha Erim Akif Kartal
Simülasyon Bisiklet Ekleme	Akif Kartal Haktan Bilgehan Dilber Pınar Erdem Furkan Çelen
Windows driver boot usb üzerinde uygulama ve oluşturma	Galip Tayfun Saygılı Mert Can Beşirli
Windows - Linux Boot İşlemi Araştırma	Haktan Bilgehan Dilber Furkan Çelen Elif Akgün
Genel uygulama ve araştırması	Galip Tayfun Saygılı Mert Can Beşirli Furkan Çelen Haktan Bilgehan Dilber Elif Akgün
Windows direkt açılma işlemi araştırması	Furkan Çelen
Mıknatıslı hız sistemi	Muhammed Yusuf Koçer Doruk Ergün Mert Can Beşirli
Potansiyometre ile yön tayini	Akif Kartal Dilara Karakaş Galip Tayfun Saygılı
Servo motor ile force feedback	Dilara Karakaş M. Yusuf Koçer Mert Can Beşirli
Nabız sensörü ile nabzin alınması	Doruk Ergün Galip Tayfun Saygılı
Bluetooth modülünün microcontroller a bağlanması	Yusuf Koçer Mert Can Beşirli Dilara Karakaş
Trainer ile donanım için gerekli bağlantıların yapılması	Doruk Ergün Yusuf Koçer Dilara Karakaş
Simülasyon tarafında haberleşme kodunun implement edilmesi	Akif Kartal
Microcontroller tarafında haberleşme kodlarının implement edilmesi	Yusuf Koçer Dilara Karakaş Doruk Ergün
Web Sitesi yapımı	Elif Akgün Zeliha Erim
Rapor	Herkes