T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ

BLUETOOTH TABANLI HARİCİ AKILLI GÖZLÜK APARATI

ÖMER EMRE POLAT
BERAT KIZILARMUT
LİSANS BİTİRME-II PROJESİ RAPORU
ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ

DANIŞMANI
DR. TUBA GÖZEL

GEBZE
2022

T.R.

GEBZE TECHNICAL UNIVERSITY

ENGINEERING FACULTY

BLUETOOTH BASED EXTERNAL SMART GLASS APPARATUS

ÖMER EMRE POLAT

BERAT KIZILARMUT

UNDERGRADUATE THESIS

GRADUATION-II PROJECT REPORT

ELECTRONICS ENGINEERING DEPARTMENT

THESIS SUPERVISOR

DR. TUBA GÖZEL

GEBZE

2022

GEBZE TEKNİK ÜNİVERSİTESİ	LİSANS JÜRİ ONAY FORMU

GTÜ Mühendislik Fakültesi Yönetim Kurulu'nun …..…/…..…/..…..… tarih ve ..……/……..… sayılı kararıyla oluşturulan jüri tarafından …..…/…..…/..…..… tarihinde tez savunma sınavı yapılan ……'ın tez çalışması ……Anabilim Dalında LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE
(TEZ DANIŞMANI) :

ÜYE :

ÜYE :

ONAY

Gebze Teknik Üniversitesi …… Enstitüsü Yönetim Kurulu'nun
..…/…../…… tarih ve ……/…..... sayılı kararı.

ÖZET

Günlük hayatta kullanıcılar giyilebilir teknoloji ürünlerini hayatlarına günden güne daha fazla entegre etmektedirler. Akıllı telefonların günlük kullanımı ile başlayan bu teknolojik devrim, akıllı saat ve akıllı bileklik gibi ürünler ile markette ivmelenmeye devam etmektedir. Google ve Microsoft gibi firmaların da geçtiğimiz yıllarda artırılmış gerçeklik barındıran yeni tür giyilebilir teknoloji ürünleri olan akıllı gözlükler ise henüz markette büyük bir kabul görmemiştir. Kullanıcıların gerçeklik ile bilgi teknolojileri arasındaki sınırı birbirine yakınlaştıran bu giyilebilir teknoloji ürünlerinin markette henüz bir yer edememesinin sebebi akıllı gözlük ve türevlerinin tasarım olarak kullanıcılara çekici gelmemesi ve maliyetlerinin yüksek olmasıdır. Bu sebeple yapılan proje daha düşük maliyette ve kullanıcıların istediği işleve göre yazılımını değiştirebileceği bir harici akıllı gözlük aparatı tasarlanması üzerine yapılmıştır.

Anahtar Kelimeler: Giyilebilir Teknoloji, Akıllı Gözlük, Gömülü Sistemler, Optik

SUMMARY

In daily life, users are integrating wearable technology products into their lives more and more day by day. This technological revolution, which started with the daily use of smart phones, continues to accelerate in the market with products such as smart watches and smart wristbands. Smart glasses, which are new types of wearable technology products that include augmented reality in the past years, have not received a great acceptance in the market yet. The reason why these wearable technology products, which bring the border between reality and information technologies closer together, have not yet found a place in the market, is that smart glasses and their derivatives are not attractive to users in terms of their design and their costs are relatively high. For this reason, the project was made on the design of an external smart glasses apparatus with a lower cost and where the users can change the onboard software according to their needs and their desired functionality.

Key Words: Wearable Tech, Smart Eyeglasses, Embedded Systems, Optics

TEŞEKKÜR

Başta, lisans eğitimimizde ve akademik hayatımızda desteğini ve yardımlarını hiçbir zaman esirgemeyip bilgisi ile bu çalışmanın oluşmasının yolunu açan danışmanımız Dr. Tuba Gözel'e ve atölyelerini tarafımız kullanımına açan Gebze Teknik Üniversitesi Robotik ve Otomasyon Kulübü'ne teşekkürlerimizi sunarız.

İÇİNDEKİLER

SİMGELER ve KISALTMALAR DİZİNİ

Simgeler ve Kısaltmalar		**Açıklamalar**
RISC	:	Reduced Instruction Set Architecture (İndirgenmiş Komut Takımlı Mimari)
SPI	:	Serial Peripheral Interface (Seri Çevre Birimi Arayüzü)
OLED	:	Organic Light Emitting Diode (Organik Işık Saçılımlı Diyot)
HUD	:	Head-Up Display (Baş Üstü Göstergesi)
Wi-Fi	:	Wireless Field (Kablosuz Ağ)

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

1. GİRİŞ

Günlük hayatta kullanıcılar giyilebilir teknoloji ürünlerini hayatlarına günden güne daha fazla entegre etmektedirler. Akıllı telefonların günlük kullanımı ile başlayan bu teknolojik devrim, akıllı saat ve bileklik gibi ürünler ile markette ivmelenmeye devam etmektedir. Google ve Microsoft gibi firmaların da geçtiğimiz yıllarda artırılmış gerçeklik barındıran yeni tür giyilebilir teknoloji ürünleri olan akıllı gözlükler ise henüz markette büyük bir kabul görmemiştir. Kullanıcıların gerçeklik ile bilgi teknolojileri arasındaki sınırı birbirine yakınlaştıran bu giyilebilir teknoloji ürünlerinin markette henüz bir yer edememesinin sebebi akıllı gözlük ve türevlerinin tasarım olarak kullanıcılara çekici gelmemesi ve maliyetlerinin yüksek olmasıdır.

Şekil 1.1: Bluetooth Tabanlı Harici Akıllı Gözlük Aparatı

1.1. Projenin Amacı, Katkısı ve İçeriği

Her telefona ve her gözlüğe uyum sağlayabilecek, kullanımı kolay ve gündelik hayata negatif etki etmeyecek bir harici akıllı gözlük aparatı çözümü üretmek. Bu gözlük aparatı kullanıcının gözüne polarize optik çözümleri ile sahip olunan bilgileri yansıtacak. Gözlüğün işlem merkezi olarak 'RISC' temelli bir mikro denetleyici ünitesi kullanılacak. Mikro denetleyici ünitesinde gömülü durumda bulunan 'Bluetooth' modülü aracılığı ile gözlüğün, 'Bluetooth' uyumlu telefon arasında iletişimi sağlanacak. Gözlüğün konfigürasyonları telefonda bulunan uygulama aracılığı ile yapılacak. Modüler teknolojisi ile kullanıcının tercihindeki gözlüğe adapte edilebilen bir ürün olacak.

1.2. Projenin Kapsamı

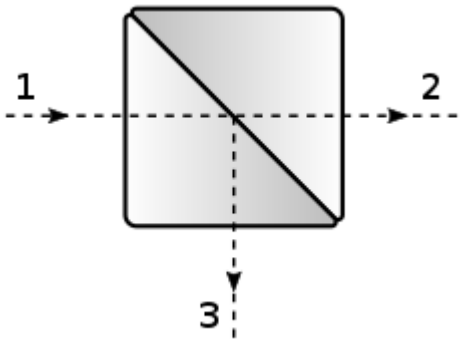
Giyilebilir teknoloji marketindeki bir sonraki adım olan akıllı gözlük çözümlerinin kullanıcılara daha uygun fiyatlarla ve daha geniş bir gözlük yelpazesine uygulanabilir hale getirilmesi.
Proje sayesinde:
-Daha ulaşılabilir fiyatlarda bir akıllı gözlük çözümü.
-Kullanıcıların kendi gözlüklerine uyum sağlayan modüler bir yapı.
-Modüler yapısından dolayı tamir edilebilir olması.
-Açık kaynak temelli olması.
-Yazılımsal olarak modüler olması, modifikasyonlara ve kişiselleştirmelere açık olması.

1.3. Literatür Taraması

Bluetooth Tabanlı Harici Akıllı Gözlük Aparatı Projesinin geliştirme aşamasında araştırılan ve projede kullanılan teknolojilerin özet bilgileri verilmiştir.

1.3.1. Işın Ayırıcı Prizma

Işın ayırıcı [1], bir ışık hüzmesini ikiye ayıran bir optik araçtır. Genellikle yapı olarak iki adet yarım küp prizmanın araya çeşitli kaplamalar eklenip yapıştırılması ile üretilir. İki yarım prizma arasındaki kaplama prizmanın ışın bölme oranını (çoğunlukla %50 ye %50) ve etkili bulunduğu dalga boyu gibi parametreleri ayarlamak için kullanılır. Işın ayırıcı prizmalar hem gelen ışık hüzmesini iki ayrı hüzmeye bölebilir, **Şekil 1.3.1.1**'de gösterildiği gibi, hem de ayrı yönlerden gelen iki ışık hüzmelerini birleştirip tek bir yönden iletebilirler.

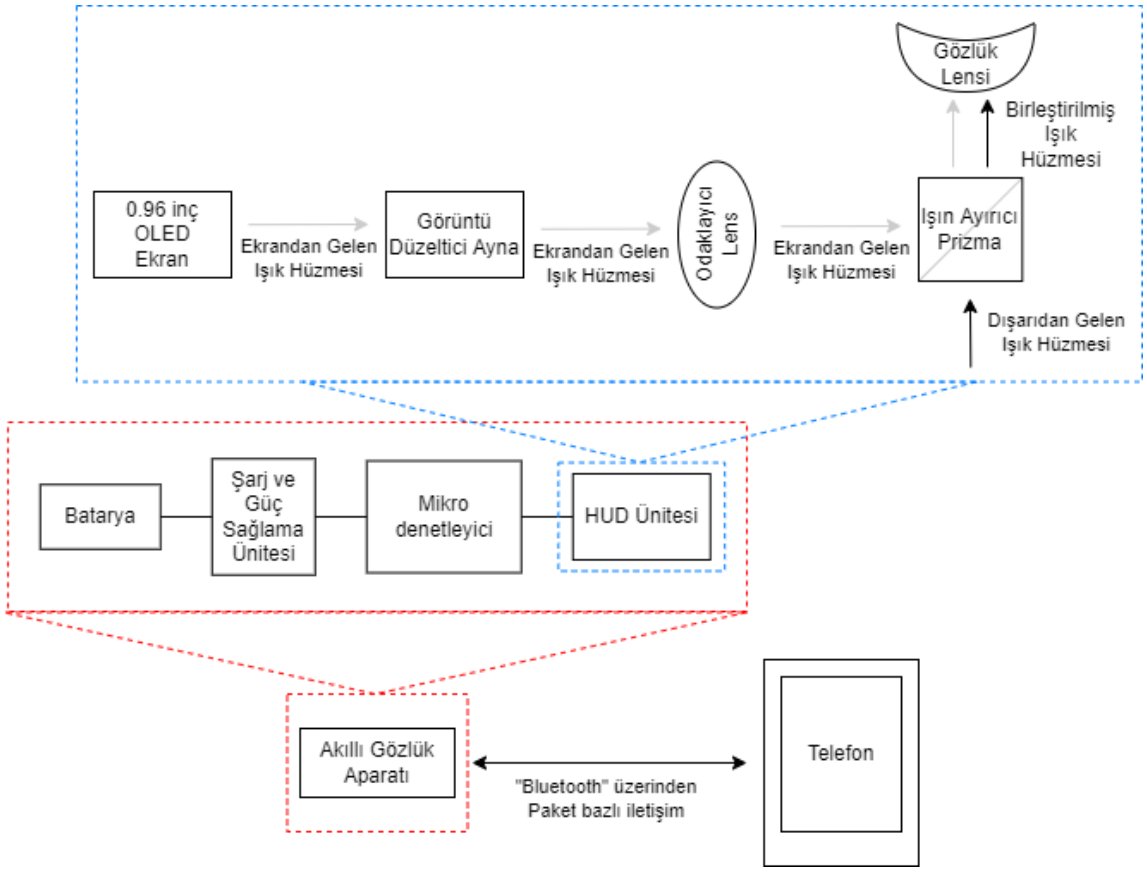


Şekil 1.3.1.1: Işın Ayırıcı Prizma çalışma prensibi diyagramı

1.3.2. Kapasitif Dokunmatik Arayüz

Kapasitif Dokunmatik Arayüzler [2] bir insan dokunuşu veya iletken dokunuşunu tespit edebilen iletişim arayüzleridir. Kapasitif arayüze temas edildiğinde temas noktasında küçük bir miktar elektrik yükü çekilerek işlevsel bir kapasitör oluşturulur. Bu oluşan kapasitör ve kapasitans değeri bir mikro denetleyici tarafından tespit edilebilir ve bir giriş arayüzü olarak kullanılabilir. Bu metot mekanik, hareket eden parçalar bulundurmadığı için uzun ömürlü bir buton tipi olarak sunulmaktadır.

2. PROJE YÖNTEMİ



Şekil 2.1: Akıllı Gözlük Aparatı Bağlantı ve İletişim Diyagramı

Bluetooth Tabanlı Harici Akıllı Gözlük Aparatı Projesi, diyagramı yukarıda **Şekil 2.1**'de verilmiş bulunan, çalışma yöntemi olarak akıllı telefonda bulunan mobil uygulama sayesinde 'Bluetooth' aracılığı ile mikro denetleyici ünitesi ile iletişim kuracaktır. Mobil uygulama tarafından alınan veriler mikro denetleyici tarafından işlenecek ve optik işlemler sonucunda kullanıcının gözüne yansıtılacaktır. Bluetooth Tabanlı Harici Akıllı Gözlük Aparatı Projesi üç alana bölünüp incelenmiştir. Raporda malzeme listesinin ardından öncelikle optik çözüm, kullanıcı ara yüzü ve elektronik donanımı barındıran fiziksel donanım tasarımı incelenecektir. Bu kısım mikro denetleyici mobil uygulama aracılığı ile gönderilen verileri kullanarak gözlük görüntü işlevselliklerini düzenleyecek ve çıkış verecektir. Dahilinde bulunan batarya ve güç üniteleri ise hem mikro denetleyiciyi hem de görüntü işlevlerine gerekli enerjiyi sağlayacaktır. Fiziksel yapının ardından mikro denetleyici ünitesinde çalışmakta olan 'Bluetooth' iletişim işlevselliği, görüntü çıktı fonksiyonları ve kullanıcı ara yüz

fonksiyonları gibi işlevsellikleri barındıran mikro denetleyici yazılımı incelenecektir. Ardından bu kısımlardan sonuncusu olan akıllı telefon ile mikro denetleyici ünitesi arasında iletişimi sağlayan, telefondan işlevsellikler için gerekli olan bilgilerin alınmasını sağlayan ve kullanıcı ayarlarının yapılmasına olanak sağlayan mobil uygulama incelenecektir.

2.1. Malzeme Listesi

Bluetooth Tabanlı Harici Akıllı Gözlük Aparatı Projesinde kullanılmak üzere seçilen parçalar ve seçilme nedenleri.

2.1.1. Mikro Denetleyici Ünitesi

Mikro denetleyici ünitesi olarak, kendi 'Bluetooth' ve 'Wi-Fi' iletişim ünitesi içinde olması, kapasitif sezi özelliğine sahip olması, boyut faktörünün kullanımımıza uygun olması, performansına göre güç harcaması az olması, fiyatının uygun olması ve bulunabilirliği kolay olması sebeplerinden dolayı ESP32-WROOM-32D [3] modeli seçilmiştir.

2.1.2. Ekran Ünitesi

Görüntü çıkışı alınması için ekran ünitesi olarak küçük boyutu, ‘SPI’ iletişim protokolü, yüksek parlaklığı ve uygun fiyatından dolayı markasız 0,96 inç OLED 80x160 SPI Ekran [4] seçilmiştir.

2.1.3. Batarya

Sisteme güç sağlaması için batarya elimizde bulunduğu için 3.7V 2000mAh Lityum-İyon bir batarya kullanılmıştır. Batarya tercihi olarak Lityum-İyon batarya seçilmesinin sebebi diğer batarya teknolojilerine göre birim boyuta kıyaslandığında enerji yoğunluğunun yüksek olmasıdır.

2.1.4. Batarya Şarj ve Yükseltici Ünitesi

Lityum-İyon tip batarya kullanıldığı için, lityum-iyon bataryaların özel gereksinimlerini sağlayan ve kendi içerisinde 3.7V gerilimi mikro denetleyicinin kullandığı 5V gerilime yükselten Adafruit Powerboost 1000C Batarya Şarj ve Yükseltici Ünitesi elimizde bulunduğu için ve gereksinimleri sağladığı için kullanılmıştır.

2.1.5. Işın Ayırıcı Prizma

Optik işlemlerde 18 mm en, 18 mm boy ve 18 mm yüksekliğe sahip, 50:50 bölme oranına sahip ve görünür ışık dalga boyunda işlevselliğe sahip bir ışın ayırıcı prizma kullanılmıştır. Geçirgenlik oranı olarak 50:50 seçilmesinin sebebi ekran parlaklığı ile ortam ışığı seviyesinin denk tutulmasının amaçlanmasıdır.

2.1.6. Lens

Optik işlemlerde kullanıcının gözünün yorulmaması için görüntünün odak noktasını kaydıran 45 mm odak uzaklığına sahip bir lens kullanılmıştır. Özel tasarım lens üretilmesinin masraflı olmasından dolayı projedeki optik çözümlere uyum sağlayabilen özelliklere sahip ‘Google Cardboard’ yapay gerçeklik gözlüğünün lensleri tercih edilmiştir.

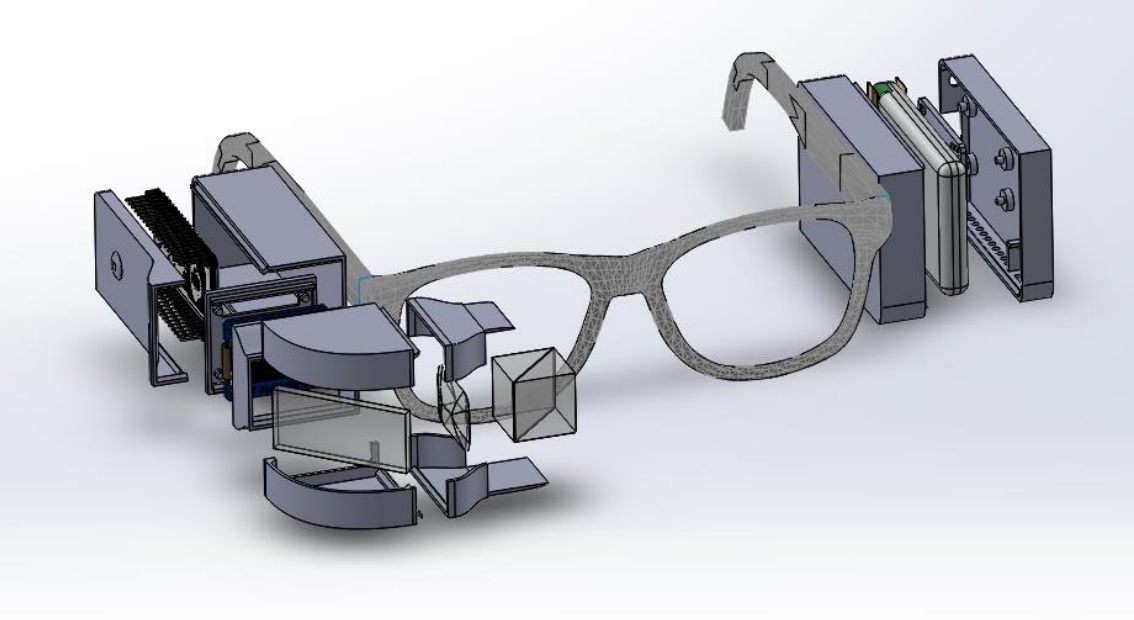
2.1.7. Bütçe

Bluetooth Tabanlı Harici Akıllı Gözlük Aparatı Projesinde kullanılan malzemeler ve fiyatları **Tablo 2.1.7.1**’de verilmiştir.

Tablo 2.1.7.1 Bluetooth Tabanlı Harici Akıllı Gözlük Aparatı Malzeme Listesi ve Bütçesi

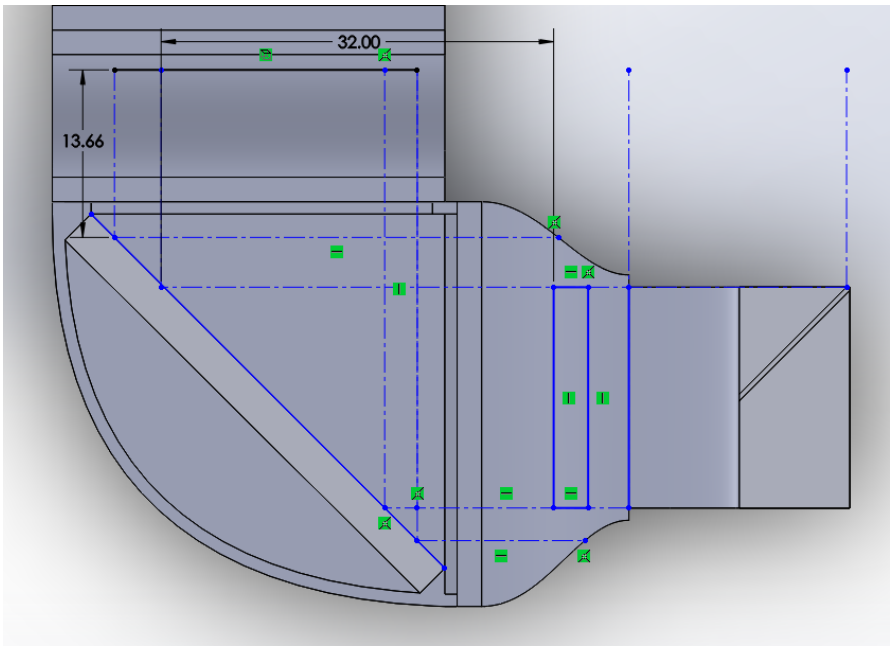
Ürün ismi	Adet	Fiyat	Link
ESP32-WROOM-32D	1	131,27 TL	https://www.direnc.net/esp32-wroom-32d-wifi-bluetooth-gelistirme-modulu
PowerBoost 1000C	1	327,26 TL	https://www.adafruit.com/product/2465
3.7V 2000 mAh Li-Ion Batarya	1	108,59 TL	https://www.direnc.net/37v-2000-mah-li-polymer-pil
tinylab 3D 1.75 mm Siyah PLA Filament	1	239,99 TL	https://www.amazon.com.tr/tinylab-1-75-Siyah-PLA-Filament/dp/B08DJ3XC3W/
0.96 inch SPI OLED Ekran	1	104,37 TL	https://www.direnc.net/096-inch-oled-arduino-spi-tft-ekran
18*18*18 Işın Ayırıcı Prizma	1	156,41 TL	https://tr.aliexpress.com/item/1005001279808206.html
Google Cardboard 25mm Lens	1	9,90 TL	https://urun.n11.com/giyilebilir-teknoloji-aksesuarlari/toptan-25mm-sanal-gerceklik-gozlugu-google-cardboard-lens-25-mm-P377565747
5*5 Pleksi Ayna	1	20,40 TL	https://urun.n11.com/diger/3mm-gumus-renk-ayna-pleksi-istediginiz-olcude-lazer-kesim-P441922073
TOPLAM		1097,80 TL	

2.2. Donanım Tasarımı



Şekil 2.2.1: Donanım Tasarımı Üç Boyutlu Modeli

Bluetooth Tabanlı Harici Akıllı Gözlük Aparatı Projesinin işlevselliği için kullanımına karar verilen elektronik ve optik malzemelerin bir araya getirilebilmesi için bir donanım tasarımı yapılmıştır. Tasarım için Bilgisayar Destekli Tasarım Programı olan 'Solidworks' kullanılmış, tasarımlar üç boyutlu yazıcı aracılığı ile PLA malzeme kullanılarak üretilmiştir. Donanım üç parçaya ayrılarak incelenmiştir. İlk parça olarak mikro denetleyici ünitesi ve kapasitif dokunmatik ara yüzü barındıran bölme oluşturulmuştur. İkinci parça olarak Lityum-İyon batarya, Batarya Şarj ve Yükseltici Ünitesi ve açma kapama anahtarı bulunan güç ünitesi bölmesi oluşturulmuştur. Kullanılmakta olan güç ünitesi teorik olarak toplamda 450mW güç kullanan mikro denetleyici ünitesi ve ekran ünitesini 16 saat olarak besleyebilecek kapasiteye sahiptir. Üçüncü parça olarak 0,96 inç OLED ekranı, görüntü düzeltici aynayı, görüntü odaklayıcı lensi ve ışın ayırıcı prizmayı barındıran bir bölme oluşturulmuştur. Bu bölmenin boyutları istenilen optik etkinin oluşturulabilmesi için, geometrisi **Şekil 2.1.2'** de verilmiş olan, matematiksel hesaplamalar sonucu elde edilmiştir. Ekran, ayna ve ışın ayırıcı prizma arasındaki mesafe yaklaşık 45mm olduğu için ayna ile ışın ayırıcı prizma arasına 45 mm odak mesafeli lens yerleştirilmiştir. Bu lens sayesinde kullanıcı görüntüyü odaklanma sorunu yaşamadan daha rahat bir şekilde görebilmektedir.

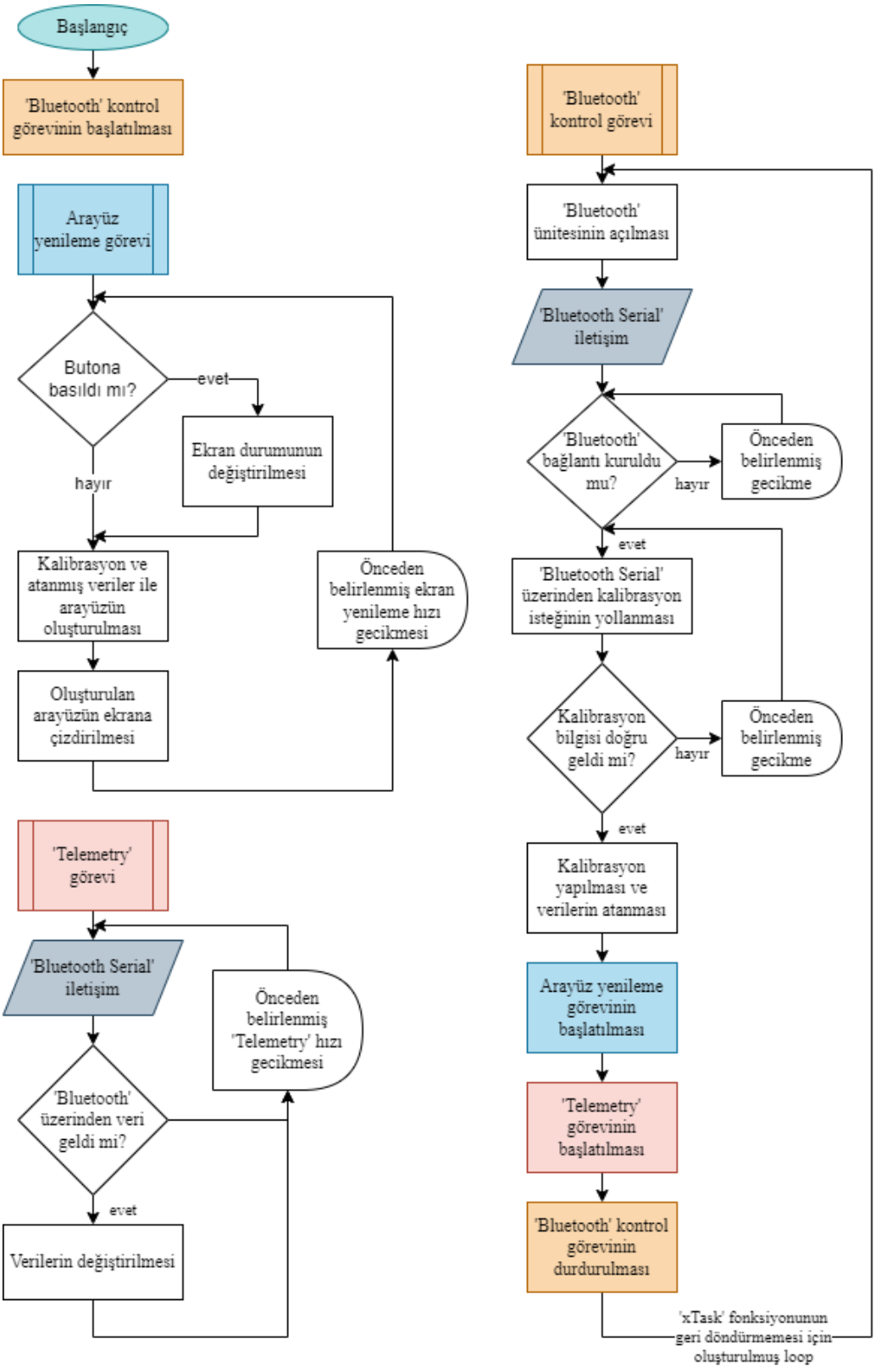


Şekil 2.2.2: Ekran, ayna, lens ve ışın ayırıcı prizma geometrik diyagramı

2.3. Mikro Denetleyici Yazılımı

Mikro denetleyici, **Şekil 2.2.1**, üzerinde bulunan yazılım düşük güç tüketimini amaçlayarak verim odaklı şekilde yazılmaktadır. Mikro denetleyici geliştirme aşamasında yazılım sürecinde ‘C++’ yazılım dili ile 'Visual Studio Code' programında 'PlatformIO' eklentisi aracılığı ile çalışılıp, 'Arduino Framework' yardımıyla hazırlanmıştır. Yazılımda ayrıca 'FreeRTOS' açık kaynak işletim sistemi kullanılmıştır. Bu işletim sisteminin avantajı birden fazla işlemin aynı anda yapılabilmesi, çeşitli işlemlerin seçilen çekirdeklerde çalıştırılması ve sürekli yapılan işlemler için görev ataması yapılabilmesidir.

Şekil 2.3.1: ESP32-WROOM-32D Mikro Denetleyici



Şekil 2.3.2: Mikro Denetleyici Yazılım Akış Diyagramı

Mikro denetleyicinin çalışma prensiplerini barındıran akış diyagramı **Şekil 2.2.2**'de verilmiştir. Mikro denetleyici işlevleri olarak tasarlanan fonksiyonlar ve kütüphaneler sıradaki kısımlarda ayrı ayrı incelenmiştir.

2.3.1. Ana İşlevsellik

Yazılımın temeli olan 'main.cpp' **Ek-A** dosyasında bütün proje kapsamında kullanılan hazır kütüphaneler (Örn:'Arduino.h', 'BluetoothSerial.h', 'Adafruit_SSD1306.h', 'SPI.h' ve 'TFT_eSPI.h'), proje için özel olarak tarafımızdan yazılmış olan kütüphaneler (Örn: 'BluetoothUtils.h', 'DisplayUtils.h' ve 'GlassUtils.h') çağırılmıştır. Bu çağırmaların ardından 'GlassUtils Glass;' aracılığıyla gözlük işlevsellik sınıfı oluşturulmuş, kurulum ayarlarının başlatılmasından itibaren yazılımın mobil cihaz uygulaması ile iletişim protokolleri başlatılmış ve görüntü çıkış operasyonları harekete geçirilmiştir.

2.3.2. Gözlük İşlevselliği Kütüphanesi

Mikro denetleyici yazılımında bütün kütüphanelerin ve işlevselliklerin hiyerarşik olarak en tepesinde bulunan kütüphane 'Gözlük İşlevselliği Kütüphanesi' 'GlassUtils.cpp' **Ek-B** ve 'GlassUtils.h' **Ek-C** dosyalarıdır. Diğer kütüphaneler bu kütüphane dahilinde üye sınıf objesi olarak tutulmaktadır. Bu kütüphane bünyesinde diğer kütüphanelerden (Örn:'IOUtils.h') alınan bilgiler diğer kütüphanelere aktarılmakta ve aralarındaki iletişim sağlanmaktadır. 'FreeRTOS' platformunun sağlamış olduğu sürekli çalışan işlemler işlevselliği bu kütüphanede kullanılmakta olup düzenli işlemler bu kütüphanede başlatılmaktadır.

2.3.3. Görüntü İşlevselliği Kütüphanesi

Mikro denetleyici görüntü çıkışları bu kütüphanede yapılmaktadır. Kütüphane 'DisplayUtils.cpp' **Ek-D** ve 'DisplayUtils.h' **Ek-E** dosyalarından oluşmaktadır. Bu kütüphane 'Bluetooth' aracılığı ile iletilen verilerin işlenilmesi, görüntü çıkışlarının

alınması, görüntü hizalama ayarlarının yapılması, buton ve durum işlevselliklerin çalışması görevlerini üstlenmektedir.

2.3.4. Bluetooth İşlevselliği Kütüphanesi

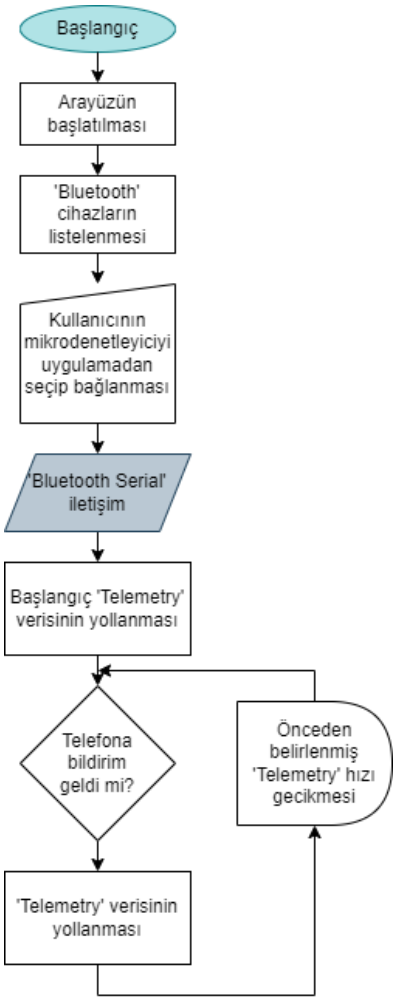
Bluetooth İşlevselliği Kütüphanesi mikro denetleyici ünitesi ile mobil cihaz uygulaması arasındaki 'Bluetooth' iletişim protokolünün başlatılması ve gelen verilerin gerekli kütüphanelere yönlendirilmesi işlevselliklerini gerçekleştirmektedir. Kütüphane 'BluetoothUtils.cpp' **Ek-F** ve 'BluetoothUtils.h' **Ek-G** dosyalarından oluşmaktadır. 'Bluetooth' verileri bu kütüphanede işlenmemektedir, bu kütüphane sadece iletişim kanalı olarak kullanılmaktadır.

2.3.5. Giriş Çıkış İşlevselliği Kütüphanesi

Giriş Çıkış İşlevselliği Kütüphanesi kapasitif buton üzerinde bulunan kapasitans değerlerini sürekli değerlendirip kapasitansta dokunma sebebi ile gerçekleşen değişikliği tespit edip üst üste birden fazla hatalı buton aksiyonu tanımlamaksızın giriş işlevselliklerini gerçekleştirip bu bilgileri Görüntü İşlevselliği Kütüphanesine aktarmaktadır. Kütüphane 'IOUtils.cpp' **Ek-H** ve 'IOUtils.h' **Ek-I** dosyalarından oluşmaktadır.

2.4. Mobil Cihaz Uygulaması

Mobil cihaz ve mikro denetleyici ünitesinin arasında iletişim kurulması için 'Bluetooth Serial' protokolü yardımı ile bir 'Android 9.0' tabanlı, akış diyagramı **Şekil 2.3.1**'de verilen, uygulama geliştirilmiştir. Uygulama 'Android Studio' uygulamasında geliştirilip, terminal kısmı 'SimpleBluetoothTerminal' [5] isimli açık kaynaklı bir uygulamadan türetilmiştir. Bu uygulama aracılığı ile mobil cihazdan mikro denetleyici ünitesine ve mikro denetleyici ünitesinden mobil cihaza çift taraflı iletişim kurulabilmektedir. Bu uygulama sayesinde akıllı gözlük aparatı ile kullanıcıların telefonları arasında bağlantı kurulmaktadır. Mikro denetleyici ünitesi saat ve benzeri bilgileri eşitlemek için kullanıcının telefonunu kullanılmaktadır. Uygulamada seçilen bildirim ayarlarına göre telefondaki bildirimler ile ilgili veri mikro denetleyiciye gönderilebilmektedir. Uygulama bünyesindeki terminal aracılığı ile mobil cihazdan mikro denetleyici ünitesi kontrol edilebilmektedir.

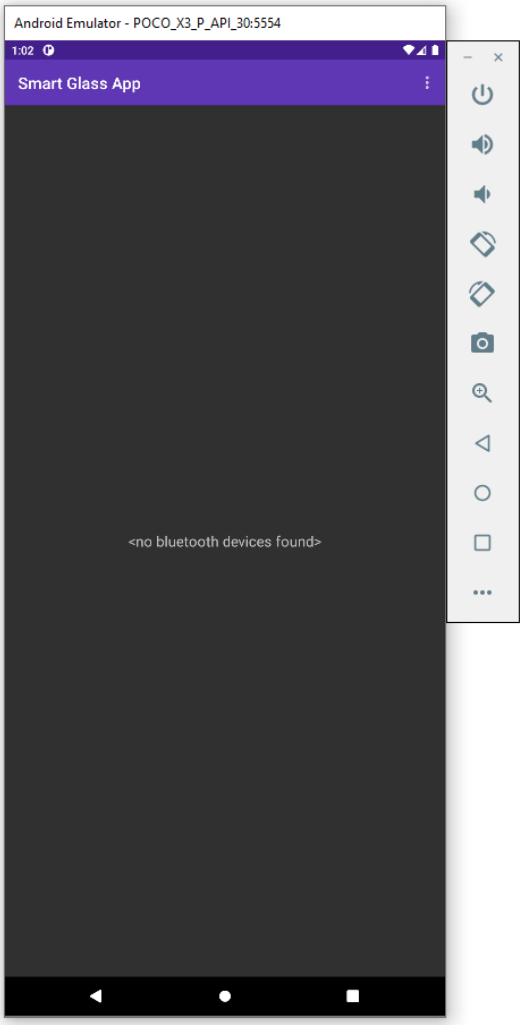


Şekil 2.4.1: Mobil Uygulama Yazılım Akış Diyagramı

3. ÜRÜN DEĞERLENDİRMESİ

3.1. Mobil Cihaz Uygulaması

Mobil cihaz üzerinden yapılan ve bilgisayar ortamında yapılan ‘Android’ emülatörü, **Şekil 3.1.1.,** ile yapılan testlerde, mikro denetleyici ile arasında iletişim sağlanabilmiştir. Bu iletişim tek taraflı olarak kullanılmaktadır.’Android’ mobil cihazdan otomatik olarak saat ve konum bilgileri alınamamaktadır. Terminal komutları ile manuel şekilde gönderilmeleri gerekmektedir.



Şekil 3.1.1: Yapılan Mobil Uygulamanın Emülatör Üzerinde Denenmesi

Aynı şekilde testler, ‘Android’ telefon üzerinden de yapılmıştır ve bu durumda veri paketleri düzeyinde iletişim sağlanabilmiştir.

3.2. Bluetooth Tabanlı Harici Akıllı Gözlük Aparatı Ünitesi

Bluetooth Tabanlı Harici Akıllı Gözlük Aparatı Ünitesi başarılı bir şekilde üretilmiş ve birleştirilmiş olup hedeflenen tasarım özelliklerine ulaşılmıştır. Gözlüğün sol tarafına yerleştirilen, batarya ve şarj ünitesini barındıran güç ünitesi 56g ağırlığa sahip olmuştur. Gözlüğün sağ tarafına yerleştirilen, optik çözüm ve mikro denetleyiciyi barındıran akıllı gözlük aparatı ünitesi 58g ağırlığa sahip olmuştur. Ortalama güç tüketimi ve batarya kapasitesi hesaplarına göre sürekli aktif kullanımda ünitenin 16 saat batarya ömrüne sahip olacağı tespit edilmiştir. Kullanıcı konforu ve kullanım alanları hedeflerine başarıyla ulaşılmıştır. Gözlük camı ve prizma arasındaki mesafenin kişiden kişiye ve gözlükten gözlüğe değişmesinden dolayı oluşan optik uyumsuzlukları çözmek için kullanıcı arayüzü kişisel tercihlere göre konumlandırılabilir hale getirilmiştir.

Tasarım boyut kısıtlamalarından dolayı mekanik butonlar yerine kapasitif buton kullanılmıştır. Bu kapasitif buton sayesinde alan daha verimli kullanılmasına rağmen ürün paketlemesinde boyut problemleri yaşanmış olup, daha hafif ve daha konforlu bir ürün üretilmesi için özel üretim devre kartlarına ve malzemelere ihtiyaç duyulmaktadır. Kullanılan Lityum-İyon batarya bu proje için ağır ve fazla büyük bir bataryadır. Daha ideal bir uygulama için daha verimli boyutlarda bir batarya kullanıcının gözlüğü üzerine değil de iki gözlük sapı arasına yerleştirilerek ağırlık dengesi daha iyi bir çözüm gerçekleştirilebilir.

Şekil 3.2.1: Kullanıcı tarafından görülen görüntü örneği

4.SONUÇLAR VE YORUMLAR

Şekil 4.1: Bluetooth Tabanlı Harici Akıllı Gözlük Aparatı

Bluetooth Tabanlı Harici Akıllı Gözlük Aparatı Projesi boyunca başarılı bir prototip ürün üretildi. Üretim aşamasında tasarımın kusurları sürekli olarak düzeltildi ve geliştirildi. Bu geliştirmeler sonucu projenin başlangıcında planlanandan daha farklı bir tasarım olmasına rağmen belirlenen özelliklere ve işlevselliklere ulaşıldı. Görüntü çıktıları kullanıcı tarafından rahatlık ile okunabilir hale getirildi. Bu sayede, kullanıcının günlük hayatını etkilemeden talep edilen bilgileri görebileceği arayüz başarı ile oluşturulmuş oldu. Mobil cihaz uygulamasında ise istenilen otomatik işlemler gerçekleştirilemedi. Eşitleme işlemleri mobil cihaz uygulaması ile manuel bir operasyon ile gerçekleştirilmektedir.

Bu proje sürecinde Elektronik Mühendisliği bölümünden öğrenmiş olduğumuz bilgilerimizin üzerine, bilgisayar tabanlı modelleme, optik çözümler ve ürün üretim alanlarında da bilgiler edinerek yeteneklerimizi geliştirmiş bulunduk.

5. ÖNERİLER VE GELİŞTİRMELER

Optik çözümde daha optimize tasarımlar elde edinilebilir durumdadır. Kullanılan ekrana kıyasla daha yüksek bir parlaklığa sahip bir ekran sayesinde 50:50 oranlı bir prizma yerine 70:30 oranlı bir prizma kullanılıp, %70 geçirgenliğe sahip kısım dışarından gelen ortam ışığını alacak şekilde konumlandırıldığında bu projedeki çözüme kıyas ile arka plan görüntüsünü daha az karartan bir arayüz elde edilebilir. Görüntü yansıtma metodu olarak aynalama kullanılmadan direkt ışın ayırıcı prizmaya görüntü verilebilir veya projeksiyon metotları da uygulanabilir. Batarya ve batarya şarj üniteleri kullanılan güç miktarına göre boyut ve fiyat olarak optimize edilebilir. Çeşitli elektronik elementlerin konumlandırmaları ağırlık dengesi açısından geliştirilebilir. Özel üretim devre kartları ile ürünün boyutu ve ağırlığı azaltılabilir.

6.KAYNAKÇA

[1] Rizea, Adrian & Popescu, I.M.. (2012). Design techniques for all-dielectric polarizing beam splitter cubes, under constrained situations. Romanian Reports in Physics. 64. 482-491.

[2] Zuk, S., Pietrikova, A. and Vehec, I. (2018), "Capacitive touch sensor", Microelectronics International, Vol. 35 No. 3, pp. 153-157. https://doi.org/10.1108/MI-12-2017-0071

[3] ESP32-WROOM-32D & ESP32-WROOM-32U Datasheet, Espressif, https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf

[4] TFT LCD MODULE 0.96 inch 80RGB*160DOTS, Zhong JY, https://www.smart-prototyping.com/image/data/2020/11/102105%200.96%20TFT%20IPS%20Display%20Module%20/Datasheet.pdf

[5] SimpleBluetoothTerminal, Kai Morich, https://github.com/kai-morich/SimpleBluetoothTerminal

EKLER

Ek A: Main.cpp

```
#include <Arduino.h>
#include <BluetoothSerial.h>
#include <BluetoothUtils.h>
#include <DisplayUtils.h>
#include <GlassUtils.h>
#include <Adafruit_SSD1306.h>
#include <SPI.h>
#include <TFT_eSPI.h>

GlassUtils Glass;

void setup()
{
  Serial.begin(115200);

  Glass.Bluetooth.Begin();
  Glass.StartBluetoothWriteToBufferTask();

  Glass.Display.Begin();
  Glass.StartDisplayTask();
  Glass.StartDisplayUpdateTimeTask();
  Glass.StartIOUpdateIOTask();
}

void loop()
{

}
```

Ek B: GlassUtils.cpp

```
#include <Arduino.h>
#include <BluetoothSerial.h>
#include <BluetoothUtils.h>
#include <GlassUtils.h>
#include <DisplayUtils.h>
#include <Adafruit_SSD1306.h>
#include <SPI.h>
#include <TFT_eSPI.h>

using namespace Bluetooth;

GlassUtils::GlassUtils()
{
  update_flag = true;
  button_flag = false;
}

GlassUtils::~GlassUtils()
{

}
```

```
// BLUETOOTH TASKS

// WRITE TO BUFFER
void GlassUtils::BluetoothWriteToBufferTask()
{
  for(;;)
  {
    if(Bluetooth.Available())
    {
      // sets update flag if the bluetooth transmission is recieved
      Bluetooth.WriteToBuffer();
      // parse the bluetooth transmission
      Display.ParseBT(Bluetooth.buffer.get(), Bluetooth.buffer_size,
this->update_flag);
    }
    else
    {
      /*
      for(size_t ii=0; ii<Bluetooth.buffer_size; ++ii)
      {
        Serial.write(Bluetooth.buffer[ii]);
      }
      */
      vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
  }
}

void GlassUtils::StartBluetoothWriteToBufferTaskImpl(void* _this)
{
  ((GlassUtils*)_this)->BluetoothWriteToBufferTask();
}

void GlassUtils::StartBluetoothWriteToBufferTask()
{
  xTaskCreate
  (
    this->StartBluetoothWriteToBufferTaskImpl, //func ptr
    "StartBTBuffer", //task name
    8192, //stack size
    (void*) this, //task parameters
    2, //task priority
    NULL //task handle
  );
}

// DISPLAY TASKS

void GlassUtils::DisplayTask()
{
  for(;;)
  {
    Display.Refresh(this->update_flag, this->button_flag);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}

void GlassUtils::StartDisplayTaskImpl(void* _this)
```

```
{
  ((GlassUtils*)_this)->DisplayTask();
}

void GlassUtils::StartDisplayTask()
{
  xTaskCreate
  (
    this->StartDisplayTaskImpl, //func ptr
    "StartDPTest", //task name
    65536, //stack size
    (void*) this, //task parameters
    1, //task priority
    NULL //task handle
  );
}

// TIME

void GlassUtils::DisplayUpdateTimeTask()
{
  for(;;)
  {
    Display.UpdateTime(this->update_flag);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}

void GlassUtils::StartDisplayUpdateTimeTaskImpl(void* _this)
{
  ((GlassUtils*)_this)->DisplayUpdateTimeTask();
}

void GlassUtils::StartDisplayUpdateTimeTask()
{
  xTaskCreate
  (
    this->StartDisplayUpdateTimeTaskImpl, //func ptr
    "StartTimeTest", //task name
    1024, //stack size
    (void*) this, //task parameters
    1, //task priority
    NULL //task handle
  );
}

// IO

void GlassUtils::IOUpdateIOTask()
{
  for(;;)
  {
    IO.UpdateIO(this->update_flag, this->button_flag);
    vTaskDelay(200 / portTICK_PERIOD_MS);
  }
}

void GlassUtils::StartIOUpdateIOTaskImpl(void* _this)
{
```

```
  ((GlassUtils*)_this)->IOUpdateIOTask();
}

void GlassUtils::StartIOUpdateIOTask()
{
  xTaskCreate
  (
    this->StartIOUpdateIOTaskImpl, //func ptr
    "StartUpdateIO", //task name
    1024, //stack size
    (void*) this, //task parameters
    2, //task priority
    NULL //task handle
  );
}
```

Ek C: GlassUtils.h

```
#pragma once
#include <Arduino.h>
#include <BluetoothSerial.h>
#include <BluetoothUtils.h>
#include <DisplayUtils.h>
#include <IOUtils.h>
#include <Adafruit_SSD1306.h>

using namespace Bluetooth;
using namespace Display;
using namespace IO;

class GlassUtils
{
    public:
        GlassUtils();
        ~GlassUtils();

        void StartBluetoothWriteToBufferTask();
        static void StartBluetoothWriteToBufferTaskImpl(void* _this);
        void BluetoothWriteToBufferTask();

        void StartDisplayTask();
        static void StartDisplayTaskImpl(void* _this);
        void DisplayTask();

        void StartDisplayUpdateTimeTask();
        static void StartDisplayUpdateTimeTaskImpl(void* _this);
        void DisplayUpdateTimeTask();

        void StartIOUpdateIOTask();
        static void StartIOUpdateIOTaskImpl(void* _this);
        void IOUpdateIOTask();

        bool update_flag;
        bool button_flag;

        BluetoothUtils Bluetooth;
        DisplayUtils Display;
        IOUtils IO;
```

```
};
```

Ek D: DisplayUtils.cpp

```
#include <Arduino.h>
#include <BluetoothSerial.h>
#include <BluetoothUtils.h>
#include <GlassUtils.h>
#include <DisplayUtils.h>
#include <Adafruit_SSD1306.h>
#include <SPI.h>
#include <TFT_eSPI.h>
#include <FS.h>
#include <SPIFFS.h>

#define GLASS_DEBUG

namespace Display
{

DisplayUtils::DisplayUtils() : tft()
{
    // initial state for the display
    current_state = display_state::main_screen;
    // initial values for the display
    hour = 0;
    minute = 0;
    second = 0;

    time_x = 0;
    time_y = 0;

    date_x = 60;
    date_y = 0;

    sender_x = 0;
    sender_y = 10;

    message_x = 0;
    message_y = 20;

    offset_x = 0;
    offset_y = 0;

    time_buffer_size = 6;
    time_buffer = std::unique_ptr<char>(new char[time_buffer_size]);
    sprintf(time_buffer.get(), "%02d:%02d", hour, minute);
    time_buffer.get()[time_buffer_size-1] = '\0';

    sender_buffer_size = 64;
    sender_buffer = std::unique_ptr<char>(new char[sender_buffer_size]);
    message_buffer_size = 128;
    message_buffer = std::unique_ptr<char>(new
char[message_buffer_size]);
    sprintf(sender_buffer.get(),"Error");
    sprintf(message_buffer.get(),"Last Message Not Found");

    date_buffer_size = 8;
```

```
    date_buffer = std::unique_ptr<char>(new char[date_buffer_size]);

    show_message_counter = 0;
    time_flag = false;
    sender_flag = false;
    message_flag = false;
    show_message_flag = false;
    date_flag = false;
}

DisplayUtils::~DisplayUtils()
{
    tft.~TFT_eSPI();
}

void DisplayUtils::Begin()
{
    tft.init();
    tft.setRotation(3);
}

void DisplayUtils::ParseBT(char* buffer, size_t buffer_size, bool&
update_flag)
{
    switch(buffer[0])
    {
        case 'T':
            // if the packet tag is Time (-1 because first is reserved
for 'T' tag)
            this->hour = (((uint8_t) buffer[1] - 48) * 10) + (((uint8_t)
buffer[2] - 48) * 1);
            this->minute = (((uint8_t) buffer[3] - 48) * 10) +
(((uint8_t) buffer[4] - 48) * 1);
            // parse values into a c string
            sprintf(time_buffer.get(), "%02d:%02d", hour, minute);
            time_buffer.get()[time_buffer_size-1] = '\0';
            time_flag = true;
            break;

        case 'S':
            // if the packet tag is Sender (-1 because first is reserved
for 'S' tag)
            memset(sender_buffer.get(), ' ', sender_buffer_size);
            strncpy(sender_buffer.get(), (buffer+1),
sender_buffer_size);
            sender_buffer.get()[sender_buffer_size-1] = '\0';
            sender_flag = true;
            break;

        case 'M':
            // if the packet tag is Message (-1 because first is
reserved for 'M' tag)
            memset(message_buffer.get(), ' ', message_buffer_size);
            strncpy(message_buffer.get(), (buffer+1),
message_buffer_size-1);
            message_buffer.get()[message_buffer_size-1] = '\0';
            message_flag = true;
            break;
```

```
        case 'D':
            // if the packet tag is Date (-1 because first is reserved
for 'D' tag)
            memset(date_buffer.get(), ' ', date_buffer_size);
            strncpy(date_buffer.get(), (buffer+1), date_buffer_size-1);
            date_buffer.get()[date_buffer_size-1] = '\0';
            date_flag = true;
            break;

        case 'X':
            {
            // if the packet tag is X offset (-1 because first is
reserved for 'X' tag)
            uint8_t end_index = 0;
            for(int ii=0; ii<buffer_size; ++ii)
            {
                if(buffer[ii] == ' ')
                {
                    end_index = ii;
                    break;
                }
            }

            #ifdef GLASS_DEBUG
                Serial.printf("end index: %d\n", end_index);
            #endif

            switch (end_index)
            {
                case 2:
                    this->offset_x = (((uint8_t) buffer[1] - 48) * 1);
                    break;

                case 3:
                    this->offset_x = (((uint8_t) buffer[1] - 48) * 10) +
(((uint8_t) buffer[2] - 48) * 1);
                    break;

                case 4:
                    this->offset_x = (((uint8_t) buffer[1] - 48) * 100)
+ (((uint8_t) buffer[2] - 48) * 10) + (((uint8_t) buffer[3] - 48) * 1);
                    break;

                default:
                    // failed to assign to offset values
                    break;
            }

            #ifdef GLASS_DEBUG
                Serial.printf("offset_x: %d\n", offset_x);
            #endif

            break;
            }

        case 'Y':
            {
            // if the packet tag is X offset (-1 because first is
reserved for 'X' tag)
```

```
            uint8_t end_index = 0;
            for(int ii=0; ii<buffer_size; ++ii)
            {
                if(buffer[ii] == ' ')
                {
                    end_index = ii;
                    break;
                }
            }

            #ifdef GLASS_DEBUG
                Serial.printf("end index: %d\n", end_index);
            #endif

            switch (end_index)
            {
                case 2:
                    this->offset_y = (((uint8_t) buffer[1] - 48) * 1);
                    break;

                case 3:
                    this->offset_y = (((uint8_t) buffer[1] - 48) * 10) +
(((uint8_t) buffer[2] - 48) * 1);
                    break;

                case 4:
                    this->offset_y = (((uint8_t) buffer[1] - 48) * 100)
+ (((uint8_t) buffer[2] - 48) * 10) + (((uint8_t) buffer[3] - 48) * 1);
                    break;

                default:
                    // failed to assign to offset values
                    break;
            }

            #ifdef GLASS_DEBUG
                Serial.printf("offset_y: %d\n", offset_y);
            #endif

            break;
            }

        default:
            // tag not known
            break;
    }

    if(sender_flag && message_flag)
    {
        show_message_flag = true;
        show_message_counter = 10;
    }

    #ifdef GLASS_DEBUG
        Serial.print(buffer[0]);
        Serial.print(buffer[1]);
        Serial.print(buffer[2]);
        Serial.print(buffer[3]);
        Serial.print(buffer[4]);
```

```
        Serial.print('\n');
    #endif

    #ifdef GLASS_DEBUG
        // update flag for screen refresh
        Serial.print("flag set\n");
    #endif

    // set update flag
    update_flag = true;
}

void DisplayUtils::UpdateTime(bool& update_flag)
{
    if(time_flag)
    {
        ++second;
        if(second == 60)
        {
            second = 0;
            ++minute;
            #ifdef GLASS_DEBUG
                Serial.write("time refresh\n");
            #endif
            sprintf(time_buffer.get(), "%02d:%02d", hour, minute);
            time_buffer.get()[time_buffer_size-1] = '\0';
            // if change is  visible return update flag as true
            update_flag = true;
        }
        if(minute == 60)
        {
            minute = 0;
            ++hour;
            #ifdef GLASS_DEBUG
                Serial.write("time refresh\n");
            #endif
            sprintf(time_buffer.get(), "%02d:%02d", hour, minute);
            time_buffer.get()[time_buffer_size-1] = '\0';
            // if change is  visible return update flag as true
            update_flag = true;
        }
        if(hour == 24)
        {
            hour = 0;
            #ifdef GLASS_DEBUG
                Serial.write("time refresh\n");
            #endif
            sprintf(time_buffer.get(), "%02d:%02d", hour, minute);
            time_buffer.get()[time_buffer_size-1] = '\0';
            // if change is  visible return update flag as true
            update_flag = true;
        }
    }

    // message counter update
    if(sender_flag && message_flag && (show_message_counter == 0))
    {
        sender_flag = false;
        message_flag = false;
```

```
        show_message_flag = false;
    }
    else
    {
        show_message_counter--;
    }
}

void DisplayUtils::Refresh(bool& update_flag, bool& button_flag)
{
    // check update flag
    if(update_flag)
    {
        #ifdef GLASS_DEBUG
            Serial.write("screen refresh\n");
        #endif

        if(button_flag)
        {
            switch(current_state)
            {
                case display_state::main_screen:
                    current_state = display_state::last_notification;
                    break;

                case display_state::last_notification:
                    current_state = display_state::status_screen;
                    break;

                case display_state::status_screen:
                    current_state = display_state::test;
                    break;

                default:
                    current_state = display_state::main_screen;
                    break;
            }
            button_flag = false;
        }

        switch(current_state)
        {
            case display_state::main_screen:
                tft.fillScreen(TFT_BLACK);
                if(time_flag)
                {
                    tft.setTextSize(2);
                    tft.drawString(time_buffer.get(), offset_x + time_x,
offset_y + time_y);
                }
                if(date_flag)
                {
                    tft.setTextSize(1);
                    tft.drawString(date_buffer.get(), offset_x + date_x,
offset_y + date_y);
                }
                if(show_message_flag)
                {
                    tft.setTextSize(1);
```

```
                    tft.drawString("New Message", offset_x + time_x,
offset_y + time_y + 17);
                    tft.drawString("From: ", offset_x + time_x, offset_y
+ time_y + 27);
                    tft.drawString(sender_buffer.get(), offset_x +
sender_x + 30, offset_y + sender_y + 17);
                    tft.drawString(message_buffer.get(), offset_x +
message_x, offset_y + message_y + 17);
                }
                break;

            case display_state::last_notification:
                tft.fillScreen(TFT_BLACK);
                tft.setTextSize(1);
                if(time_flag)
                {
                    tft.drawString(time_buffer.get(), offset_x + time_x,
offset_y + time_y);
                }
                tft.drawString("Last Message", offset_x + time_x ,
offset_y + time_y + 10);
                tft.drawString("From: ", offset_x + time_x, offset_y +
time_y + 20);
                tft.drawString(sender_buffer.get(), offset_x + sender_x
+ 30, offset_y + sender_y + 10);
                tft.drawString(message_buffer.get(), offset_x +
message_x, offset_y + message_y + 10);
                break;

            case display_state::status_screen:
                tft.fillScreen(TFT_BLACK);
                tft.setTextSize(1);
                if(time_flag)
                {
                    tft.drawString(time_buffer.get(), offset_x + time_x,
offset_y + time_y);
                }
                tft.drawString("System Status", offset_x, offset_y +
time_y + 10);
                tft.drawString("BT On", offset_x, time_y + offset_y +
20);
                break;

            case display_state::test:
                tft.fillScreen(TFT_BLACK);
                tft.setTextSize(1);
                tft.drawString("a1a2a3a4a5a6a7a8a9a1a2a3a4", 0 ,0);
                tft.drawString("b1b2b3b4b5b6b7b8b9b1b2b3b4", 0 ,15);
                tft.drawString("c1c2c3c4c5c6c7c8c9c1c2c3c4", 0 ,30);
                tft.drawString("d1d2d3d4d5d6d7d8d9d1d2d3d4", 0 ,45);
                tft.drawString("e1e2e3e4e5e6e7e8e9e1e2e3e4", 0 ,60);
                tft.drawString("f1f2f3f4f5f6f7f8f9f1f2f3f4", 0 ,75);
                break;
        }
        // clear update flag after refresh
        update_flag = false;
        #ifdef GLASS_DEBUG
            Serial.write("flag cleared\n");
        #endif
```

```
    }
    else
    {
        // do nothing
    }
}

}
```

Ek E: DisplayUtils.h

```
#pragma once
#include <Arduino.h>
#include <BluetoothSerial.h>
#include <BluetoothUtils.h>
#include <Adafruit_SSD1306.h>
#include <SPI.h>
#include <TFT_eSPI.h>

namespace Display
{

class DisplayUtils
{
    public:
        DisplayUtils();
        ~DisplayUtils();

        void Begin();
        void Refresh(bool&, bool&);
        void ParseBT(char*, size_t, bool&);
        void UpdateTime(bool&);

        enum class display_state:uint8_t { main_screen = 0,
last_notification = 1, status_screen = 2, test = 3};
        display_state current_state;

        uint8_t hour;
        uint8_t minute;
        uint8_t second;

        uint8_t time_x;
        uint8_t time_y;

        uint8_t date_x;
        uint8_t date_y;

        uint8_t sender_x;
        uint8_t sender_y;

        uint8_t message_x;
        uint8_t message_y;

        uint8_t offset_x;
        uint8_t offset_y;

        std::unique_ptr<char> time_buffer;
        size_t time_buffer_size;
```

```
        std::unique_ptr<char> sender_buffer;
        size_t sender_buffer_size;
        std::unique_ptr<char> message_buffer;
        size_t message_buffer_size;
        std::unique_ptr<char> date_buffer;
        size_t date_buffer_size;

        size_t show_message_counter;
        bool time_flag;
        bool sender_flag;
        bool message_flag;
        bool date_flag;
        bool show_message_flag;

    private:
        TFT_eSPI tft;
};
} //namespace
```

Ek F: BluetoothUtils.cpp

```
#include <Arduino.h>
#include <BluetoothSerial.h>
#include <BluetoothUtils.h>
#include <GlassUtils.h>
#include <DisplayUtils.h>
#include <Adafruit_SSD1306.h>
#include <SPI.h>
#include <TFT_eSPI.h>

namespace Bluetooth
{

BluetoothUtils::BluetoothUtils() : SerialBT()
{
    // define buffer size to hold the bluetooth data
    buffer_size = 128;
    // allocate buffer using unique pointer
    buffer = std::unique_ptr<char>(new char [buffer_size]);
}

BluetoothUtils::~BluetoothUtils()
{
    // call destructor for member subclass
    SerialBT.~BluetoothSerial();
}

void BluetoothUtils::Begin()
{
    // initialize serial bluetooth communication
    SerialBT.begin("Smart Glass V1.0");
}

void BluetoothUtils::WriteToBuffer()
{
    // if there is a message inbound
    if(this->Available())
    {
```

```
        // resets old contents
        memset(buffer.get(), ' ', buffer_size);
        // writes the incoming telemetry to buffer
        SerialBT.readBytes(buffer.get(), buffer_size);

        #ifdef GLASS_DEBUG
            for(int i=0; i<buffer_size; i++)
            {
                Serial.print(buffer[i]);
            }
            Serial.print('\n');
        #endif
    }
    else
    {
        // do nothing
    }
}

bool BluetoothUtils::Available()
{
    // return bluetooth availability
    return SerialBT.available();
}
}; // namespace
```

Ek G: BluetoothUtils.h

```
#pragma once
#include <Arduino.h>
#include <BluetoothSerial.h>
#include <DisplayUtils.h>
#include <Adafruit_SSD1306.h>

namespace Bluetooth
{

class BluetoothUtils
{
    public:
        BluetoothUtils();
        ~BluetoothUtils();

        void Begin();
        void WriteToBuffer();
        bool Available();

        size_t buffer_size;
        std::unique_ptr<char> buffer;

    private:
        BluetoothSerial SerialBT;
};
} //namespace
```

Ek H: IOUtils.cpp

```cpp
#include <Arduino.h>
#include <BluetoothSerial.h>
#include <SPI.h>
#include <IOUtils.h>

#define GLASS_DEBUG

namespace IO
{

IOUtils::IOUtils()
{
    ignore_cycle = 0;
    IOTouch = 0;
}

IOUtils::~IOUtils()
{

}

void IOUtils::UpdateIO(bool& update_flag, bool& button_flag)
{
    if(ignore_cycle != 0)
    {
        --ignore_cycle;
    }

    IOTouch = touchRead(T7);

    #ifdef GLASS_DEBUG
        Serial.println(IOTouch);
    #endif

    if(IOTouch < 15 && ignore_cycle == 0)
    {
        button_flag = true;
        update_flag = true;
        ignore_cycle = 5; // 200ms delay per cycle, 1000 ms per check
    }
}

}
```

Ek I: IOUtils.h

```cpp
#pragma once
#include <Arduino.h>
#include <BluetoothSerial.h>
#include <SPI.h>

namespace IO
{

class IOUtils
{
    public:
        IOUtils();
```

```
        ~IOUtils();

        void UpdateIO(bool&, bool&);

        uint8_t ignore_cycle;
        uint16_t IOTouch;

};

} //namespace
```