

Bilgisayar Proje Raporu

Projenin Adı

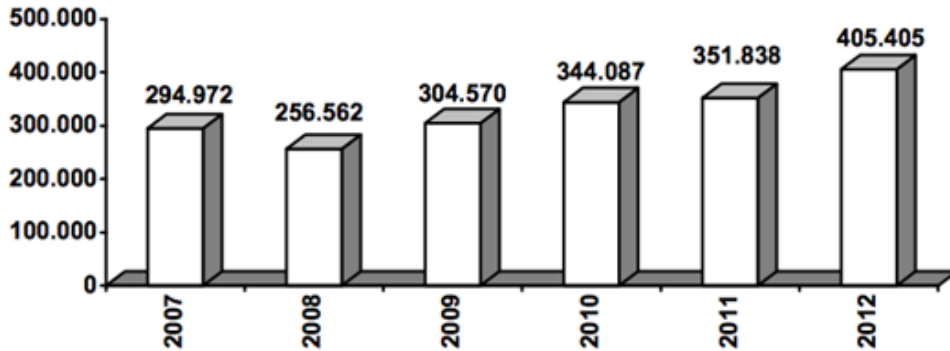
Güvenlik Sorununa Çok Fonksiyonlu Yeni Nesil Robotlu Çözüm.

Projenin Amacı

Amacımız günümüzün önemli bir sorunu olan can ve mal güvenliğine elektronik, bilgisayar ve mekaniği birleştirerek mikroişlemci ve yazılım temelli etkili bir çözümü sunmaktır.

Giriş

Gelişen teknoloji ve sanayileşme can ve mal güvenliği konusunda daha ciddi tedbirler almamızı gerektirmektedir. Ankara Ticaret Odası Başkanlığının verilerine göre 2013 yılında Türkiye’de genelinde 2398 kişi doğal gaz ve karbon monoksitten zehirlenmiş olup bunlardan 211’i hayatını kaybetmiştir^[1]. Yine Ankara Ticaret Odasının verilerine göre 2007 yılında ülkemizde toplam 297,972 hırsızlık olayı gerçekleşmiş olup bu sayı 2012 yılında 405,405’e ulaşmıştır (Şekil 1) ^[1].



Şekil 1.Türkiye’de yıllara göre hırsızlık vakaları

Günümüzde kullanılan güvenlik sistemleri; analog ve dijital kameralar, yangın alarm sistemleri, gaz kaçağı alarm sistemleri, harekete duyarlı kameralardır. Fakat bu sistemlerin bir yere sabitlenmesi gerekmektedir ve kapsama alanları sınırlıdır. Bir noktaya sabitlenmek zorunda olan bu sistemlerin zehirli gaz alarm sistemleri bulundukları yerde sesli uyarı verirken güvenlik kameraları kayıt tutabilmektedir. Bu durum da güncel sistemlerin verimliliğini sınırlandırmaktadır.

Geliştirdiğimiz hareketli ve interaktif güvenlik robotu ile güvenlik sistemlerine yeni bir bakış açısı kazandırmayı hedefledik.

Yöntem

Bu proje kapsamında yangın, hırsızlık ve gaz zehirlenmelerinde daha verimli olabilecek interaktif bir güvenlik robotu geliştirmeyi hedefledik. Bu amaç doğrultusunda aşağıdaki basamaklar gerçekleştirilmiştir.

Güvenlik ve uzaktan kontrol sistemlerini bir araya getirmek ve bunları kontrol etmek için mini bir yazılım geliştirdik. Her yeni elektronik aksam eklememizde yazılımını ve mekanik yapısını yeniden tasarladık. Elektronik aksamaları kullanırken ilgili elektronik parçaların kullanım kılavuzlarına ulaştık. Yaptığımız çalışmalar robotu yirmi farklı kombinasyonda dizerek denedik ve her seferinde ona yeni bir özellik ekleyip geliştirmeye çalıştık. Robota uzaktan görüntü aktarımı, hareket ve ses sensörlerini ekledik. Daha sonra diğer bilişim araçlarıyla entegrasyonunu sağlayarak sensörler ve diğer giriş birimlerinden aldığımız verileri analiz ederek, yorumlayarak sonuçlarını duruma göre cep telefonu ve e-posta gibi diğer bilişim araçlarına yönlendirdik. Her yeni giriş birimi eklediğimizde yazılımı yeniden programladık.

Ortamı algılamada kızılötesi, ses, uzaklık, gaz, ultrasonik sensörler ve ortamın resmini çekmek için kamera kullandık.

Ortamı algılayan elektronik parçalardan gelen verileri analiz etmek için Linux işletim sistemi üzerinde Processing yazılım dilini kullandık.

Ortamdan gelen analizleri deęerlendirip sonularını Wi-Fi internet baęlantısı ve SMS sistemini kullanarak dięer biliřim aralarına ulařtırdık.

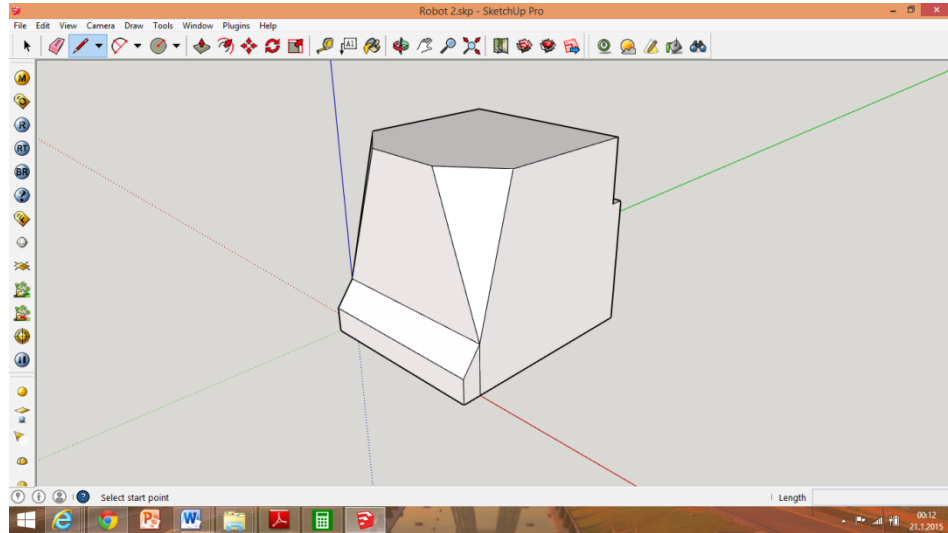
Mekanik kısımda sensörleri ortamı en iyi algılayacak řekilde yerleřtirdik. Bunun iin sensörlerin yerlerini deęiřtirerek ortamı en iyi algılayabilecek řekilde son haline getirdik.

Güvenlięi daha etkin bir biimde saęlayabilmek iin robot kullanmaya karar verdik. Geliřtirme sürecini 4 ařamada gerekleřtirdik.

1. Tasarım

Robotun kaporta kısmını yaparken pleksi tercih ettik. Bunu tercih etmemizin sebebi daha düřük maliyetli ve iřlemesi kolay bir madde olmasıydı.

Kaportasını öncelikle SketchUp programında tasarlamaya bařladık (řekil -2). Pleksi kullanarak yapacaęımız iin köřeli bir yapısı olması gerekti. Tasarladıęımız modeli SketchUp Layout programını kullanarak ölçölendirme iřlemini yaptık. Ardından ölçöleri alınan paraları pleksiglas levhayı kesen lazer makinesinin algılayabilmesi iin Corel Draw programına aktarmamız gerekti.



řekil-2: SketchUp programında kaportanın tasarımı

Kesilen parçaları kloroform ile birbirine yapıştırarak kaportayı elde ettik. Kaportanın daha güzel gözükmesi için LED kullanarak aydınlatma sistemi döşedik.

Robotun hırsızı algıladığında ışıkları açabilmesi gerekiyordu ve bunu her ortamda yapabilmesi için ışığı açan anahtarın üstüne takılabilen mekanik bir parçanın en iyisi olacağını düşündük. Bunun için robotun kaportasında da kullandığımız pleksi tercih ettik. Robot yapılan parça ile 433 Mhz bandında iletişim kuruyor ve robottan alınan sinyale göre mekanik parçadaki servo motor ile anahtarı açıp kapatıyor.

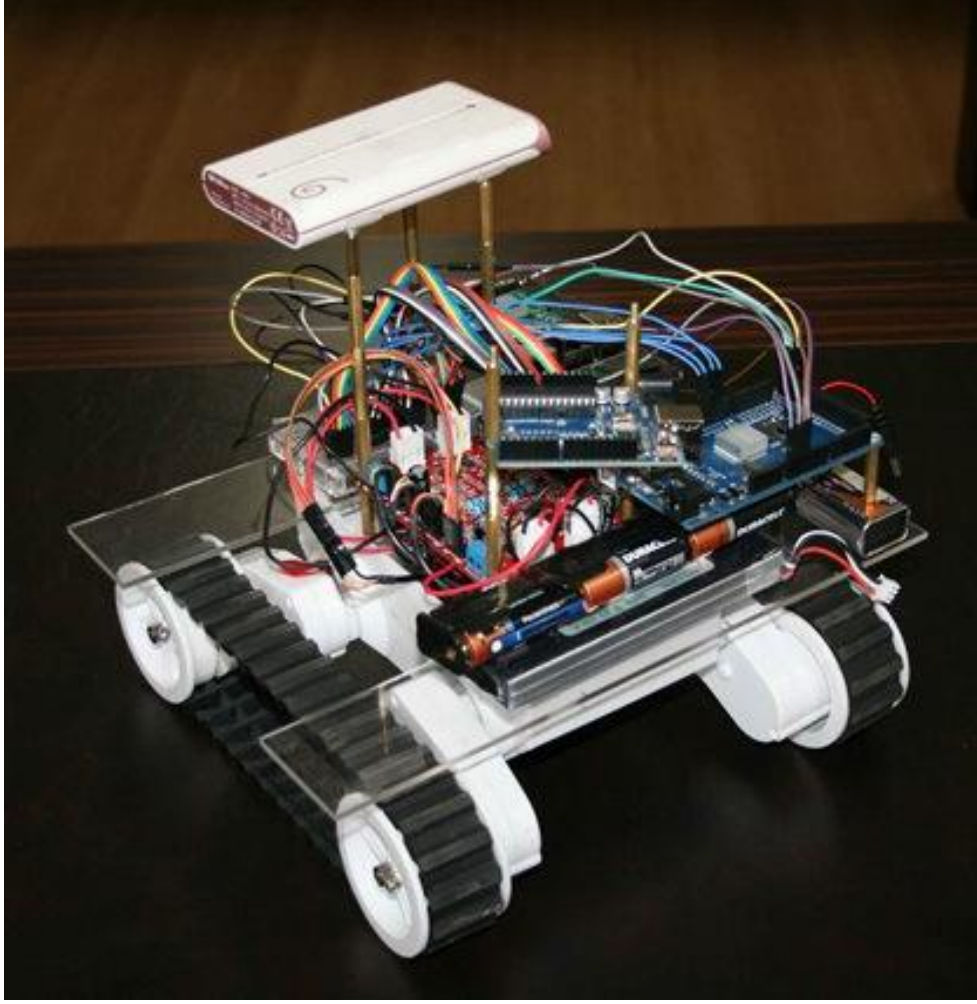
2. Mekanik

Robotun yürüyen aksamı olarak ROVER 5 Robot Platform kullandık. Bunun sebebi paletli ve 4 motorlu olmasıydı. Ayrıca her motorun kendine ait bir dişli takımı bulunduğundan dolayı motorun tek başına ürettiği torktan daha fazla tork ürettiyordu. Bu sayede engelleri aşması kolaylaştı.

Motorları kontrol edebilmemiz için bir de motor sürücü kartı kullandık. Motor sürücü kartı olarak da Rover 5 için üretilmiş olan Rover 5 motor kartını kullandık.

3. Elektronik

Robotun mekanik aksamının üstüne elektronik parçaları ergonomik olarak yerleştirdik. Elektronik devreleri ilk olarak bilgisayarda Proteus ISIS programıyla tasarladık. Tasarladığımız devreleri bilgisayar ortamında simule ettik. Devreleri pertinaks üzerine lehimledik. Daha sonra mekanik aksam üzerine monte ederek robotun elektronik aksamını tamamladık (**Şekil-3**)



Şekil-3: Robotun kaporta takılmadan önceki hali

Kullandığımız Rover 5 motor sürücü kartı sayesinde her motorun hızını ve yönünü kontrol edebildik. Robota güç verebilmek için çeşitli güç kaynakları kullandık (Şekil 4). Bunlar;

- 7.4V 2S 3400 mAh Li-Po Batarya (Ana Motorlar)
- 11.1V 3S 450 mAh Li-Po Batarya (Sensörler)
- 6000 mAh GP Taşınabilir Batarya (Raspberry Pi)
- 1.5V 6 tane AA Alkalın Pil (Servo Motorlar)
- 1.5V 2 tane AA Alkalın Pil (Aydınlatma)

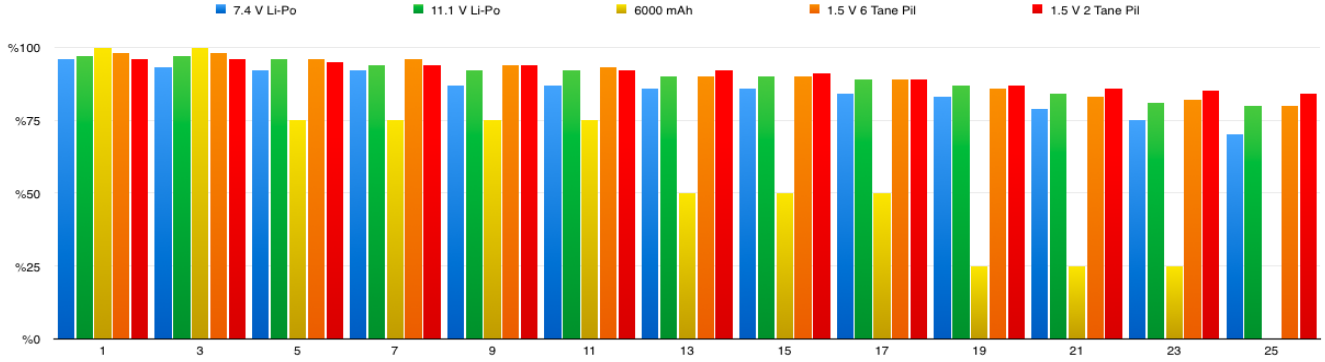


Şekil-4: 11.1V 3S 450 mAh Li-Po Batarya

Bu bataryaların tam dolu halinden sistemin kapanmasına kadar yaptığımız testlerde 6000 mAh’lik güç kaynağının erken bittiğini ve sistemi kapattığını gözlemledik. Bunu daha iyi bir batarya kullanarak çözebileceğimizi kaydettik. Diğer bataryaların ise en fazla %30 güç kullandıklarını gözlemledik (Şekil 5, Şekil 6) .

Saat	7.4 V Li-Po	11.1 V Li-Po	6000 mAh	1.5 V 6 Tane Pil	1.5 V 2 Tane Pil
1	%96	%97	%100	%98	%96
3	%93	%97	%100	%98	%96
5	%92	%96	%75	%96	%95
7	%92	%94	%75	%96	%94
9	%87	%92	%75	%94	%94
11	%87	%92	%75	%93	%92
13	%86	%90	%50	%90	%92
15	%86	%90	%50	%90	%91
17	%84	%89	%50	%89	%89
19	%83	%87	%25	%86	%87
21	%79	%84	%25	%83	%86
23	%75	%81	%25	%82	%85
25	%70	%80	%0	%80	%84

Şekil-5: Bataryaların zamana göre değişim grafiği



Şekil-6: Bataryaların zamana göre değişim grafiği

Güç kaynaklarını bağlamak ve dağıtmak için devre kartı hazırladık. Devre kartına her bir bataryanın gücünü kontrol edebilecek anahtarlar yerleştirdik. Böylece herhangi bir sorun çıkması durumunda kolayca bataryaları kontrol edebildik. Li-Po bataryaların güç durumunu ölçmek için Li-Po ölçer kullandık. Pilleri ise Arduino'ya bağlayarak anlık olarak güç durumlarını kontrol ettik. 1.5V olan pilleri 6-2 olacak şekilde ayırdık ve seri olarak bağladık. Bize gerekli olan 5V'u elde edebilmek için LM7805 voltaj regülatörü kullandık.

Güvenliği sağlarken kullanabileceğimiz sensörleri araştırdık ve liste haline getirdik. Bu liste;

- Hareket Sensörü (HC-SR501)
- Ses Sensörü (**Şekil-7**)
- Işık Sensörü (LDR)
- Uzaklık Sensörü x2 (HC-SR04)
- Uzaklık Sensörü x4 (Sharp GP2Y0A21YK0F) (**Şekil-8**)
- Yakınlık Sensörü (TCRT 5000)
- Gaz Sensörü (LPG Gas Sensor - MQ-5)
- Sıcaklık Sensörü (LM35)
- Kamera (Logitech C270)



Şekil-7: Kullandığımız ses sensörü



Şekil-8: Kullandığımız Sharp GP2Y0A21YK0F sensörü

Sensörlerden gelen verileri kontrol edebilmek ve motorlara veri gönderebilmek için kullanabileceğimiz kartları araştırdık. Kullanabileceğimiz en uygun kartın Arduino olduğuna karar verdik. Bunun sebebi önceden Arduino ile çalışmalar yapmamız ve fazlaca dokümantasyonu bulunması oldu. Kullanacağımız donanım fazla olduğu için iki

tane Arduino Mega kullandık. Böylece Arduino'lar ile bilgisayar arasındaki iletişimi hafifleterek daha hızlı hale getirdik.

Ultrasonik uzaklık sensörleri çalışırken fazla zaman kaybettirdiği için ana sistemi çok fazla beklettiğini gördük. Bunu engellemek için Arduino programı yazdık. Programı Arduino'ya yükledik ve Atmega 328p işlemcisini çıkardık. İşlemciyi breadboard üzerinde kurarak kendi Arduino'muzu yapmış olduk. Bu işlemci asenkron olarak çalışıyor ve belirli aralıklarda ana sisteme pinler üzerinden sinyal yolladı. Bu sinyalleri her sensör için 3 tane çıkış pini yaparak hazırladık. Pinlerden gelen veri 1 veya 0 oluyor. Bu durumda her sensör için $3!=6$ durum ortaya çıkıyor. Her aralığı belli bir değer arasında değerlendirerek gerekli çıkışı yaptık ve ana sistemden aynı şekilde yorumlayarak uzaklığın aralığını bulmuş olduk. Böylece ana sistemi yavaşlatmadan ultrasonik sensörleri kullanmış olduk.

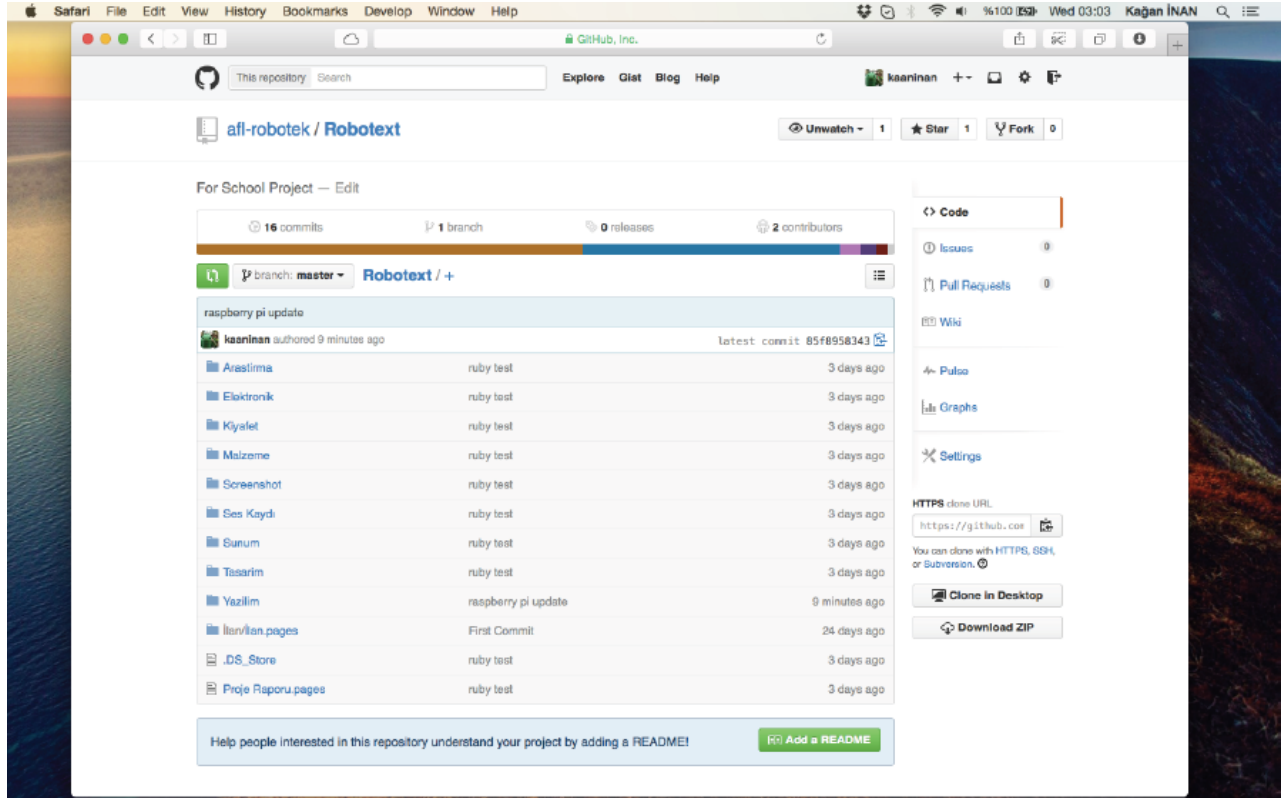
Sensörleri kullanırken kablo karmaşasının arttığını gördük ve bunu engellemek için elektronik devreler yaptık. Bize gereken 5V için voltaj regülatör kartları oluşturduk. Motorları test etmek için transistör devresi kurduk. Aynı tip sensörleri bir araya toplamak için küçük devreler yaptık. Devreleri Fritzing programı ile tasarladık, Proteus ISIS programı ile sanal ortamda test ettik. Simülatör üzerinde çalışmayan kısımları gördük ve gerekli değişiklikleri yaptık. Devreler son haline ulaştıklarında delikli pertinaks üzerinde lehimledik ve gerçek testlerini yaptık.

Arduino'ları görev paylaşımı yaparak kullandık. Birinci Arduino Mega'ya ana motorları, servo motorları ve sensörleri ikinci Arduino Mega'ya SMS shield'ı, ekranı, aydınlatma için kullandığımız ledleri, 433 mHz verici devresi taktık.

Yazılım

Yazılım dosyalarını kaydetmek ve birlikte çalışabilmek için Git sistemini tercih ettik (**Şekil-9**). Böylece projenin aşamalarını tek tek görebildik ve ortak olarak dosyalar üzerinde çalışabildik. Sorun olduğunda ise eski dosyalara kolayca dönebildik.

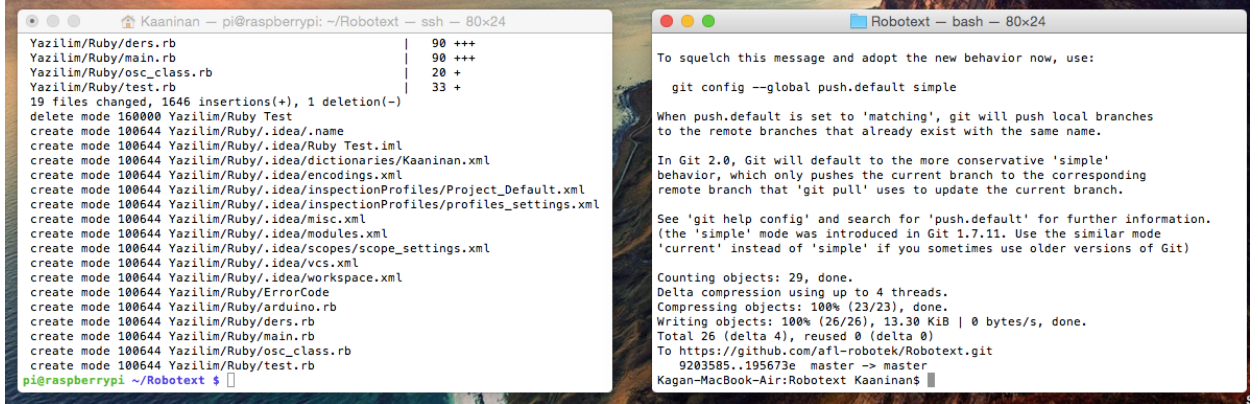
Dosyaların adresi: <https://github.com/afl-robotek/Robotext>



Şekil-9: Projenin GitHub sayfası

Beyin olarak kullanacağımız platformu Linux olarak belirledik. Bunun sebebi ücretsiz ve kısıtlı donanımlarda çalışabilir olmasıydı. Linux kullanan kartları araştırdık ve kullanabileceğimiz en uygun kartın Raspberry Pi olduğuna karar verdik. Bunun sebebi çok satılan bir kart olmasıydı. Bunun sayesinde bir hatayla karşılaştığımız zaman araştıracağımız kaynakları arttırdık. Elektronik donanımı ise kontrol edebileceğimiz en uygun kartın Arduino olduğuna karar verdik. Bunun sebebi önceden Arduino ile

çalışmalar yapmış olmamız ve kaynakların fazla olması oldu. Raspberry Pi'ye uzaktan bağlanmak için Şekil-10 ve Şekil-11'de görüldüğü üzere ssh ve vnc protokollerini kullandık. Raspberry Pi'yi ise WiFi adaptörü ile internete bağladık. Her yerden erişebilmeyi ise modemden 22 no'lu portu yönlendirerek sağladık.



The image shows two terminal windows. The left window, titled 'Kaaninan — pi@raspberrypi: ~/Robotext — ssh — 80x24', displays the output of a git push command, showing file changes and commit details. The right window, titled 'Robotext — bash — 80x24', shows the output of a git config command, indicating that the push.default setting has been changed to 'simple'.

```
Yazilim/Ruby/ders.rb | 90 +++
Yazilim/Ruby/main.rb | 90 +++
Yazilim/Ruby/osc_class.rb | 20 +
Yazilim/Ruby/test.rb | 33 +
19 files changed, 1646 insertions(+), 1 deletion(-)
delete mode 160000 Yazilim/Ruby Test
create mode 100644 Yazilim/Ruby/.idea/.name
create mode 100644 Yazilim/Ruby/.idea/Ruby Test.iml
create mode 100644 Yazilim/Ruby/.idea/dictionaries/Kaaninan.xml
create mode 100644 Yazilim/Ruby/.idea/encodings.xml
create mode 100644 Yazilim/Ruby/.idea/inspectionProfiles/Project_Default.xml
create mode 100644 Yazilim/Ruby/.idea/inspectionProfiles/profiles_settings.xml
create mode 100644 Yazilim/Ruby/.idea/misc.xml
create mode 100644 Yazilim/Ruby/.idea/modules.xml
create mode 100644 Yazilim/Ruby/.idea/scopes/scope_settings.xml
create mode 100644 Yazilim/Ruby/.idea/vcs.xml
create mode 100644 Yazilim/Ruby/ErrorCode
create mode 100644 Yazilim/Ruby/arduino.rb
create mode 100644 Yazilim/Ruby/ders.rb
create mode 100644 Yazilim/Ruby/main.rb
create mode 100644 Yazilim/Ruby/osc_class.rb
create mode 100644 Yazilim/Ruby/test.rb
pi@raspberrypi ~/Robotext$
```

```
To squelch this message and adopt the new behavior now, use:

git config --global push.default simple

When push.default is set to 'matching', git will push local branches to
the remote branches that already exist with the same name.

In Git 2.0, Git will default to the more conservative 'simple'
behavior, which only pushes the current branch to the corresponding
remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Counting objects: 29, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (23/23), done.
Writing objects: 100% (26/26), 13.30 KiB | 0 bytes/s, done.
Total 26 (delta 4), reused 0 (delta 0)
To https://github.com/afl-robotek/Robotext.git
9203585..195673e master -> master
Kagan-MacBook-Air:Robotext Kaaninan$
```

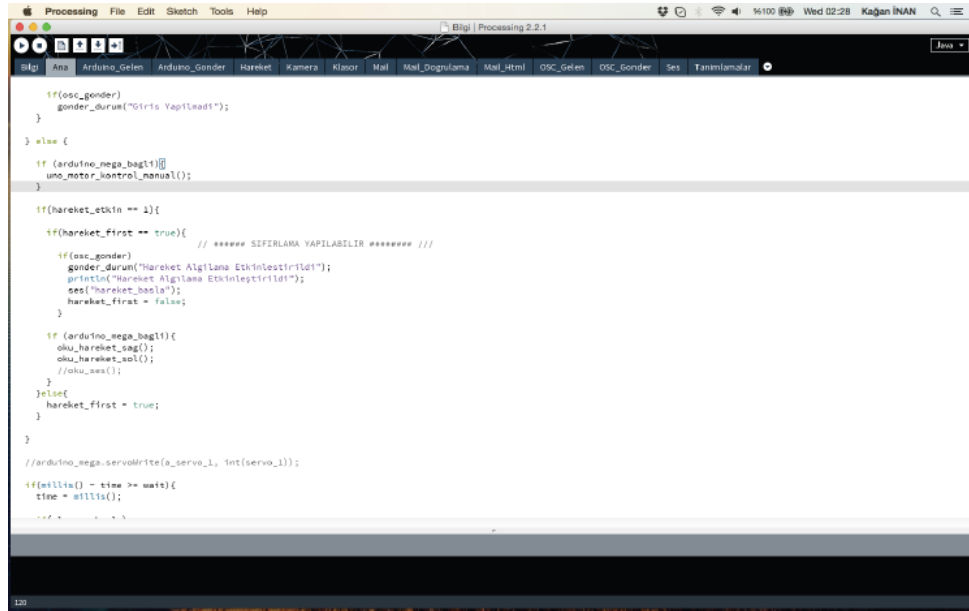
Şekil-10: Raspberry Pi'ye ssh üzerinden bağlanma



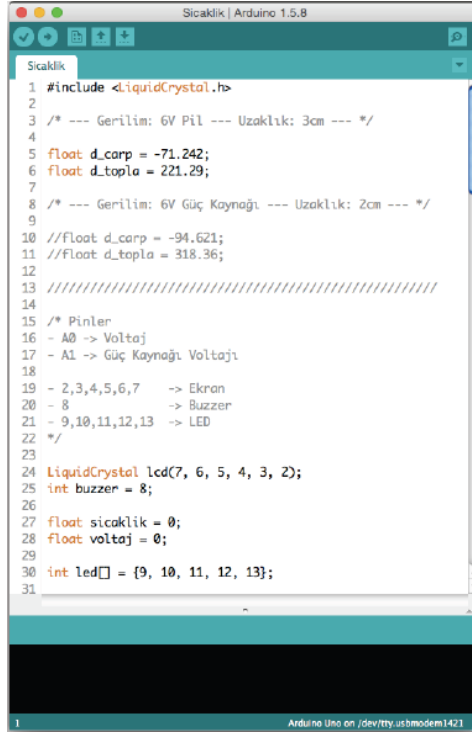
Şekil-11: Raspberry Pi'ye vnc üzerinden bağlanma

Yazılımı yapmadan önce bir algoritma oluşturduk. Bu algoritmayı en kolay ve hızlı yoldan hayata geçirebileceğimiz iki programlama dili seçtik. Bunlardan biri Ruby, diğeri ise Java tabanlı Processing diliydi. Processing önce derlenen, sonra yorumlanan bir dil, Ruby ise sadece yorumlanan bir dil. İkisine de test uygulaması yazdık ve çalıştırdık. Processing'in daha hızlı çalıştığını ve internetteki kaynaklarının daha fazla olduğunu tespit ettik.

Elektronik kısmın kontrolü için Arduino kartını kullandık. Bu kartı Processing adında Java tabanlı programlama dilinde Şekil 12'da görülen program aracılığı ile programladık. İlk başta gelen ve giden verileri kontrol etmek için serial port'u kullandık. Bilgisayar tarafında ise Processing ile motor kontrol uygulaması yazdık. Burada Arduino gelen harflere göre motorları hareket ettiriyordu. Motorların çokluğundan dolayı serial portun bizim kullanımımızda yavaş çalıştığını gördük. Bunu çözmek için Arduino'nun sağladığı Firmata protokolüne geçtik (Şekil 13). Bu sayede kontrolleri Arduino programının içinde değil bilgisayar tarafındaki Processing programından gerçekleştirdik. Bu kullanımda verilere daha rahat eriştik ve daha hızlı hale getirdik. Raspberry Pi'deki işlemcinin ARM tabanlı olması nedeniyle Processing programı düzgün çalışmadı. Bunun Java'dan dolayı sorun çıkardığını bulduk ve Java versiyonu OpenJDK 7 ARM olarak değiştirdik.



Şekil 12 : Processing Programı



```
Sıcaklık | Arduino 1.5.8
1 #include <LiquidCrystal.h>
2
3 /* --- Gerilim: 6V Pil --- Uzaklık: 3cm --- */
4
5 float d_carp = -71.242;
6 float d_topla = 221.29;
7
8 /* --- Gerilim: 6V Güç Kaynağı --- Uzaklık: 2cm --- */
9
10 //float d_carp = -94.621;
11 //float d_topla = 318.36;
12
13 ///////////////////////////////////////////////////
14
15 /* Pinler
16 - A0 -> Voltaj
17 - A1 -> Güç Kaynağı Voltajı
18
19 - 2,3,4,5,6,7 -> Ekran
20 - 8 -> Buzzer
21 - 9,10,11,12,13 -> LED
22 */
23
24 LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
25 int buzzer = 8;
26
27 float sıcaklik = 0;
28 float voltaj = 0;
29
30 int led[] = {9, 10, 11, 12, 13};
31
```

Şekil 13: Arduino Programı

Arduino'lardan birine Firmata protokolüyle diğerine serial üzerinden bağlanacak şekilde tasarladık. Bunun sebebi kullandığımız bazı parçaların (ör: ekran, sms shield) Firmata protokolü ile kullanılmamasıydı. Serial ile kullandığımız Arduino'yu Processing üzerinden kontrol ettik. Processing tarafında da iki kütüphaneyi bir arada kullandık.

Sistemi test etmek için telefon kontrollü bir yazılım hazırladık. Arduino'ya Firmata protokolüyle bağlandık, verileri Processing programında işledik ve TouchOSC programını kullanarak telefonla veri alışverişi sağladık. Bu sayede telefon veya tableten robotun kontrollerine erişmiş olduk. Şekil 14'de ki TouchOSC programı bilgisayar ile internet üzerinden haberleşen, tasarımının bilgisayar üzerinde yapıldığı bir program olması bize çok kolaylık sağladı.

Robotun bir başka özelliği ise robotun sunucusuna bağlanarak evin içinde canlı olarak gezmektir . Robotun arayüzünde girerek robotu istenilen odaya yönlendiriyor. Kamerasından aktarılan canlı görüntü sayesinde evin her tarafını anlık olarak görebiliyorsunuz

Robotun arayüzünde çalıştığı mod, çekilen resimler, hareket kontrolleri, pil durumu yer alıyor. Bu arayüz üzerinden robotu internet olan her yerden kontrol etmek mümkün hale geliyor.



Şekil-14: TouchOSC Layout Editor program ile hazırlanmış kontroller

Robotun konuşabilmesi için gerekli ses çalma kütüphanelerini inceledik. Karşımıza Türkçe okuyabilen kütüphane çıkmadığı için sesi kendimiz kaydettik ve mp3 formatına çevirdik. Raspberry Pi'nin ses kayıtlarını çalması kullanacağımız programları belirledik. Processing ile ses kaydı çalmanın çok zor olduğunu gördük ve çalmak için Processing'den komut satırına erişim omxplayer programını kullanarak ses kayıtlarını çaldık.

Robotun arayüzünde çalışacak internet sitesi için geniş çaplı bir araştırma yaptık. Robotun uzaktan kontrol edilmesi için dinamik bir internet sitesi kullanmaya karar verdik. Dinamik olabilmesi için Raspberry Pi'de bir sunucu çalışması gerekiyordu. Kullanabileceğimiz sanal sunuculara baktık ve Xampp ve Nodejs'i belirledik. İkisini de test ettik ve Xampp'ın Raspberry Pi'de kaynak kullanımını çok fazla arttırdığını ve sistemin genel çalışmasını çok olumsuz etkilediğini tespit ettik. Nodejs'de ise kaynak

kullanımını fazla etkilemediğini ve yazmasının kolay olduğunu belirledik. (Şekil 15, Şekil 15)



Şekil-15: Arayüz görüntüsü: Genel



Uyarılar

⚠ Saat 23:55'de oturma odasında yüksek şiddetli ses algılandı!

Rapor

Tarih: 23.12.2014



Mutfak



Oturma Odası



Yatak Odası



Çocuk Odası

Saat

Olay

İşlem

20:33

Odanın sağ tarafında kısa süreli hareket algılandı.



Resme Bak

✗ Sil

23:55

Ani bir ses çıkışı algılandı.

Onaylandı



Resme Bak

✗ Sil

02:45

⚠ Uzun süreli hareket algılandı.

Onaylanmadı



Resme Bak

✗ Sil

Resimler

Tarih: 23.12.2014



Mutfak



Oturma Odası



Yatak Odası

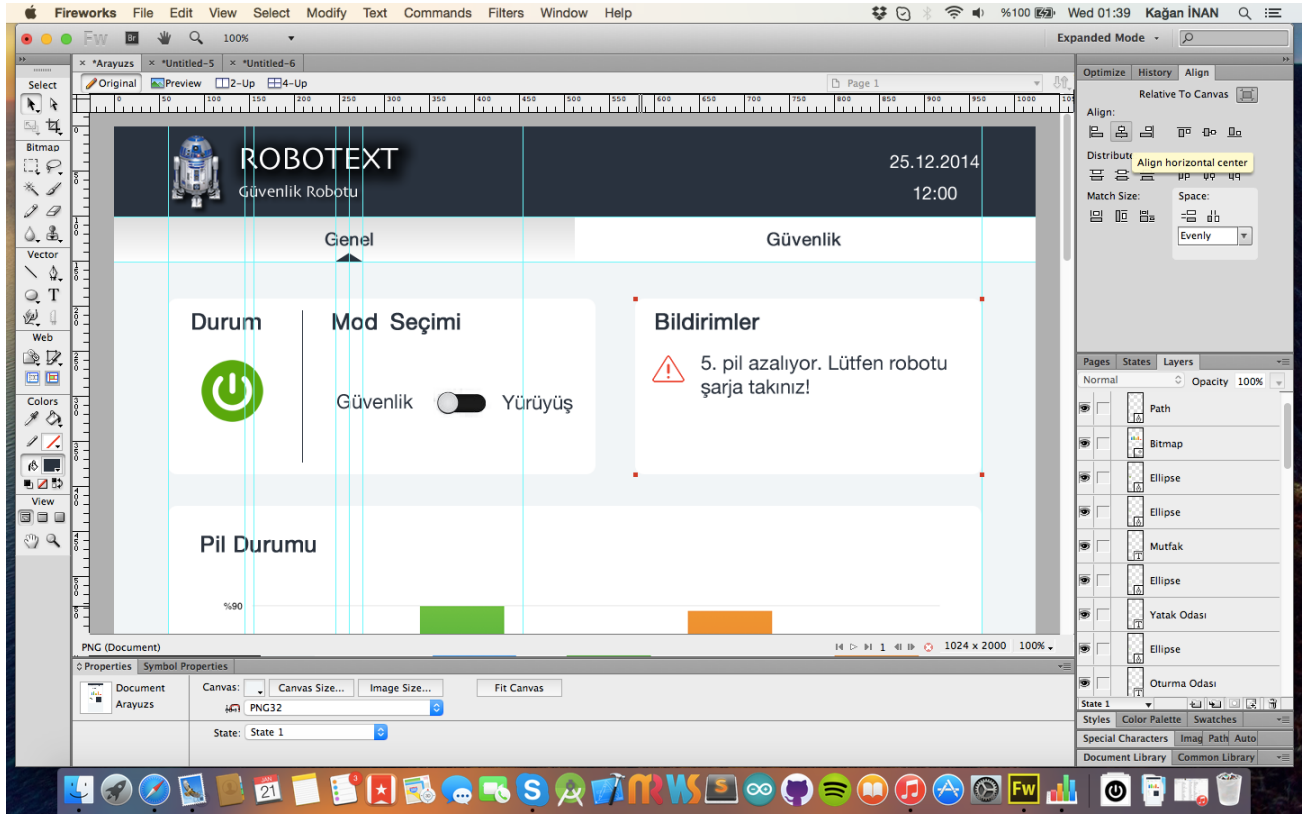


Çocuk Odası



Şekil-16: Arayüz görüntüsü: Güvenlik Bölümü

Arayüzü tasarlamak için Şekil 17’te ki Adobe Fireworks programını kullandık. Tasarımı bitirdikten sonra Nodejs eklentisi olan Mean Stack üzerinde kodladık. Mean Stack sayesinde hızlı ve kolay yoldan arayüzü tasarlamış olduk. Arayüzü ana programla WebSocket üzerinden konuşturduk. Bunu seçmemizin nedeni Processing’i desteklemesi ve esnek bir kullanım sunması oldu.



Şekil-17: Adobe Fireworks programında arayüz tasarımı

Sonuç ve Tartışma

Yaptığımız araştırmalar sonucunda ve ulaştığımız veriler ışığında elde ettiğimiz bilgilere göre günümüzde kullanılan güvenlik sistemleri ile ortalama bir evi donatmanın fiyatı yaklaşık olarak 2500 lira civarındadır.^[3] Montaj fiyatını ile beraber 3000 liralara kadar ulaşmaktadır. Bu fiyatlar pek çok ev ve iş yeri sahibi için fazla gelmekte ve bundan dolayı güvenlik sistemleri fazla yaygınlaşmamaktadır. Bunun sonucu olarak Türkiye’de hırsızlık vakalarının gün geçtikçe arttığı görülmekte ve en sık rastlanan hırsızlık türlerinin %25,56’sı evden hırsızlık olduğu belirtilmektedir.

Bu vakalarda hem cana hem de mala zarar gelmektedir. Mevcut sistemler kapsama alanlarının yetersizliği, pahalı oluşu ve kurulum zorlukları sebebiyle tercih edilmemekte ve suç oranlarını azaltmakta yeterince başarılı olmamaktadır.

Bizim geliştirdiğimiz robot bu sistemlerin yarı fiyatında olmasına rağmen, hareket anında veya ses algıladığında, odanın ışıklarını açıyor hareketin veya sesin olduğu noktanın resmini çekiyor. Bu resim üzerinde yüz tanınması yapıyor ve kayıtlı yüzlerle eşleştiriyor. Eğer resimde kayıtlı yüz varsa normal çalışmasına devam ediyor. Yüz tanımadığı durumda alarm çalmaya başlıyor. Resimi detaylı şekilde kullanıcıya mail atıyor ve kullanıcının telefonuna bildirim mesajı atıyor. Eğer bildirimi kullanıcı onaylarsa alarm kapanıyor. Hareketin devam etmesi durumunda resim çekmeye devam ediyor. İkinci mail olarak çektiği resimleri, hareket grafiğini ve ses grafiğini atıyor. Hareketin devam etmemesi durumunda 1 dk sonra genel olarak odayı çekiyor ve mail atıyor. Çekilen resimler, çekildiği tarih ve saate göre klasörlendiriliyor.

Bir başka güvenlik problemi ise karbondioksit ve doğal gaz zehirlenmeleridir. Türkiye’de her yıl ortalama 2211 kişi bu şekilde zehirlenmekte, ortalama 220 kişi ise hayatını kaybetmektedir.

Zehirlenmeden kaynaklı insan kayıplarını engellemek için robota gaz sensörü ve duman sensörü ekledik. Bu sensörler robotla beraber hareket ettiğinden evin her yerinde ölçüm yapabilmektedir. Eğer doğal gaz veya duman tespit ederse belirlenen kişiye ayrıntılı olarak SMS ve e-mail göndermektedir.

Robot son haliyle bir güvenlik görevlisinin yapabileceği iş ve işlemleri asgari düzeyde yapabilmektedir. Ortamdan gelen verileri farklı sensörler yardımıyla algılayabiliyor ve bu algıladığı sinyaller üzerinde analiz yapıp sonucunu ilgili kişilere eposta (Şekil-18) ve SMS (Şekil-19) olarak iletebilmektedir.

Ayrıca robot internet üzerinden, mevcut bilişim araçları (Cep telefonu, tablet, bilgisayar) yardımıyla uzaktan kontrol etme, anlık görüntü ve bilgi aktarımı yapabilmektedir.

Mevcut durumumuzu hedeflediğimiz tam teşekküllü güvenlik robotunun başlangıç aşaması olarak değerlendirebiliriz.

Geliştirmemiz gereken özellikler ; sensörlerin mesafelerinin artırılması, ortamdan gelen verilerinin analizlerinin yapılması ve sonuçlarının diğer bilişim araçlarına ulaştırılmasının hızlandırılması ve robotu mekanik olarak ilgi çekici bir tasarımla şekillendirmek olarak sıralayabiliriz.



Dikkat! Ters Giden Bir Şeyler Var

Uyarı



Oturma odasında uzun süreli hareket algılandı!

Komutlar



Kayıtlı Resimler



Onaylıyorum



Canlı Görüntü

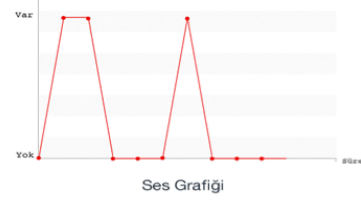
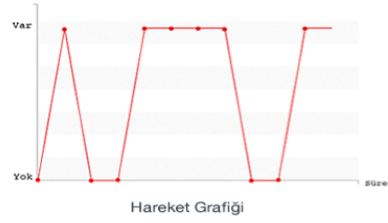


Onaylamıyorum

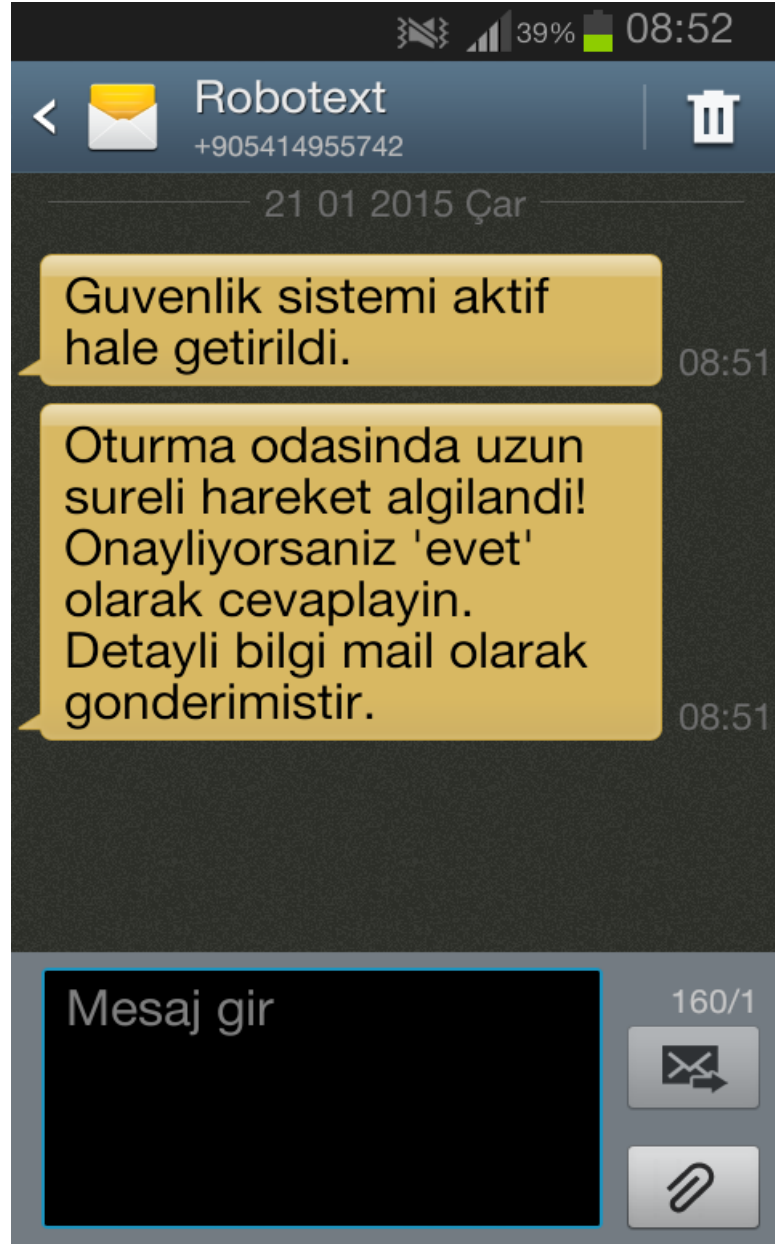
Resimler

GÖRÜNTÜ

Grafikler



Şekil-18:Robot tarafından gönderilen e-posta arayüzü



Şekil-19:Robot tarafından gönderilen SMS örneği

KAYNAK

1. ^[1] <http://www.atonet.org.tr/yeni/index.php?p=316&l=1>
2. ^[2] <http://www.nsm.gov.tr/nsm/?tag=zehirlenme>
3. <https://github.com/peterbraden/node-opencv>
4. <http://mentalized.net/journal/2010/03/08/5-ways-to-run-commands-from-ruby/>
5. <http://blog.miguelgrinberg.com/post/how-to-build-and-run-mjpg-streamer-on-the-raspberry-pi>
6. <http://kaleemullah.wordpress.com/2011/05/16/install-dnssd-on-ubuntu/>
7. <http://socket.io/get-started/chat/>
8. <http://weworkweplay.com/play/raspberry-pi-nodejs/>
9. <http://www.letsmakerobots.com/node/39969>
10. <https://github.com/getflourish/STT>
11. http://wolfpaulus.com/journal/embedded/raspberrypi_webcam/
12. <http://www.raspberrypi-spy.co.uk/2013/06/raspberry-pi-command-line-audio/>
13. <http://scruss.com/blog/2014/01/07/processing-2-1-oracle-java-raspberry-pi-serial-arduino-/>
14. <https://github.com/ruby-opencv/ruby-opencv>
15. <http://forum.processing.org/one/topic/speech-to-text.html>
16. <http://doctus.org/archive/index.php/t-5290.html>
17. <http://www.opengeospatial.org/standards/wps>
18. <http://www.robotistan.com>
19. <http://www.direnc.net>
20. <http://www.roboweb.net>
21. <http://www.elektrovadi.com>
22. <http://www.raspberrypi.org>
23. <http://www.sparkfun.com>
24. <http://www.resuldolaner.com/arduino-ile-bluetooth-kontrol-akilli-ev-uygulamasi/>
25. <http://www.arduinoturkiye.com/arduino-gsm-shield/>
26. <https://www.devcenter.heroku.com/articles/ruby-websockets>
27. <http://www.ubaa.net/shared/processing/opencv/>
28. <http://www.playground.arduino.cc/Learning/Linux>
29. <http://www.playground.arduino.cc/Main/CapacitiveSensor?from=Main.CapSense>
30. <https://www.jasperproject.github.io/>
31. <http://www.mitchtech.net/raspberry-pi-opencv/>
32. <http://www.raspberrypi.org/an-image-processing-robot-for-robocup-junior/>
33. <http://www.mitchtech.net/raspberry-pi-opencv/>
34. <http://www.xseignard.github.io/2013/04/24/processing-and-gpios-on-raspberry-pi/>
35. <http://www.cagewebdev.com/index.php/raspberry-pi-running-processing-on-your-raspi/>
36. <https://www.github.com/shokai/node-arduino-firmata>