MARMARA ÜNİVERSİTESİ
TEKNİK BİLİMLER MESLEK YÜKSEKOKULU

KONDO KHR-3HV ROBOTUNU

RASPBERRY PI 3 UZERINDEN KONTROLÜ

Bitirme Projesi

Ali KAZAR

460115007

Burak AKÇA

460116823

Muratcan ÜNSAL

460115505

Bölüm : Bilgisayar Teknolojileri
Program: Bilgisayar Programlama
Danışman: Doç. Dr. Vedat TOPUZ

MARMARA ÜNİVERSİTESİ
TEKNİK BİLİMLER MESLEK YÜKSEKOKULU

KONDO KHR-3HV ROBOTUNU

RASPBERRY PI 3 UZERINDEN KONTROLÜ

Bitirme Projesi

Ali KAZAR

460115007

Burak AKÇA

460116823

Muratcan ÜNSAL

460115505

Bölüm : Bilgisayar Teknolojileri
Program: Bilgisayar Programlama

Proje Sınavı:
Doç. Dr. Vedat TOPUZ
Ögr.Gör. Dr. Aysun ALTIKARDEŞ
Ögr.Gör. Fatih KAZDAL

Özgünlük Bildirisi

1. Bu çalışmada, başka kaynaklardan yapılan tüm alıntıların, ilgili kaynaklar referans gösterilerek açıkça belirtildiğini,

2. Alıntılar dışındaki bölümlerin, özellikle projenin ana konusunu oluşturan teorik çalışmaların ve yazılım/donanımın benim tarafımdan yapıldığını bildiririm.

Tarih:

Ad :

Soyad :

İmza:

İçindekiler

Şekil Listesi

ÖZET

Robotlar günümüzde hem hobi olarak hem de üniversitelerde çeşitli alanlarda araştırma yapmak adına çok sık kullanılmaktadır. Biz de okulumuzda bulunan Kondo KHR-3HV adlı robotu incelemek, nasıl çalıştığını anlamak ve herhangi bir kullanıcının robotu istediği gibi hareket ettirebilmesini sağlamak adına proje konumuz olarak bu konuyu aldık.

Projeye başlarken öncelikle KHR-3HV robotuna kendi programını kullanarak hareket ettirmeyi hedefledik. Bu aşamayı gerçekleştirirken elde ettiğimiz bilgiler sayesinde esas amacımız olan başka bir işletim sistemi kullanarak robotu hareket ettirmek için bir temel oluşturmuş olduk.

Bu noktadan sonra bir Raspberry Pi 3 üzerine Raspbian adında Linux tabanlı bir işletim sistemi yükledik. Önceki araştırmalarımızdan elde ettiğimiz bilgiler ışığında "libkondo4" adlı, açık kaynak kodlu, C dilinde yazılmış bir kütüphane bulduk. Bu kütüphanenin fonksiyonlarını kullanarak Raspberry Pi 3 üzerinden robota erişmeyi başardık.

"libkondo4" kütüphanesini, Python dilini ve HTML5, Javascript gibi teknolojileri kullanarak herhangi bir kullanıcının, robot üzerinde kolayca çeşitli işlemler yapabileceği bir uygulama yazdık.

1.GİRİŞ

Projemizin amacı, Kondo KHR – 3HV insansı robotunu Raspberry Pi 3 üzerinden erişerek hareket ettirmektir.

Robotu yapan firmanın yayınladığı bir program olan HeartToHeart4[1] bu işlevi görmesine rağmen sadece Windows işletim sistemli bilgisayarlar üzerinde çalışmaktadır. Bizim yapmak istediğimiz ise bu programın yaptıklarının bir kısmını linux tabanlı bir işletim sistemi üzerinden yapabilmekti.

Bu amacı gerçekleştirmek için yapmış olduğumuz araştırmalar sonucunda, bu robot modeli için C programlama dilinde yazılmış olan, açık kaynak kodlu "libkondo4"[2] kütüphanesini bulduk.

Öncelikle bu kütüphaneyi Raspberry Pi 3 üzerine kurmuş olduğumuz Raspbian işletim sistemi üzerinde derledik. Daha sonra ise içeriğini keşfetmek için çeşitli denemeler yaptık.

Bu denemeler sonucunda robotun üzerine kendi programı olan HeartToHeart4 v2.5 üzerinden kaydetmiş olduğumuz hareketlere Raspberry Pi üzerinden ulaşıp, robota bu hareketleri yaptırdık. Ayrıca robotun üzerinde bulunan 17 adet servo motoru 'libkondo4' aracalığı ile teker teker kontrol edebilip ve istediğimiz değerde harekete geçirebilmekteyiz.

Tüm bunları yaptıklarımızı bir kullanıcının kontrol edebilmesi için bir web arayüzü tasarladık. Kullandığımız Raspberry Pi 3'ü robota erişmek için kullandığımız gibi ayrıca bu tasarladığımız web arayüzünü de tutan bir web sunucusuna içinde bulundurmaktadır.

Raporun;

2. kısmı Kondo KHR-3HV İnsansı Robot hakkında bilgi vermektedir.

3. kısmı Proje Tanımı ve Planı hakkında bilgi vermektedir.

4. kısımda ise kullanılan kütüphaneler açıklanmaktadır.

5. kısım en geniş bölüm olup, yapılan tüm işlemler, kod açıklamaları bu bölümde anlatılacaktır.

6. kısımda ise projenin sonucu, geliştirilebilecek kısımları, olumsuz yönlerinden bahsediecektir.

2. KONDO KHR-3HV İNSANSI ROBOT

KHR-3HV Japonya'daki en meşhur insansı robottur.

Üzerinde 17 adet KRS-2552RHV model servo motor bulunmaktadır. Bu model servolarda Daisy Chained adında bir seri protokol kullanılmaktadır. Bu protokol sayesinde birden fazla servo bir zincir gibi birbirine bağlanarak kullanılan kablo sayısı düşürüldüğü gibi bağlantıda basitleştirilir.

Ayrıca RCB-4HV adlı bir seri haberleşme uyumlu kontrolör bulunmaktadır. Bu kontrolör ayrıca ICS3.5 protokolünü de desteklemektedir

ICS 3.0(Interactive Communication System) modül(servo) ve kontrolör arasında kullanılan iki yönlü veri haberleşme standartıdır.

Bu protokol, servomotorun "hız" ve "esneme" özeliklerine ek olarak "sıcaklık limiti" , "anlık limit" gibi özelliklerininde işlem sırasında değiştirilmesine olanak sağlamaktadır.[3robosavvy]

Kullanıcı bilgisayarını RCB-4HV ile bağlamak için kullanılan Serial USB Adapter HS adında bir adaptörü vardır. Bu adaptör 1.25 Mbps yüksek hızlı iletişimi desteklemektedir.[4kondo-robot]

3.PROJENİN TANIMI VE PLANI

3.1. Projenin Tanımı

Bu proje sayesinde Kondo KHR-3HV robotunun üzerindeki daha önceden kaydetmiş olduğumuz hareketlere ulaşıp robota bu hareketler yaptırılabilir. Tek tek tüm servo motorlar kontrol edilebilir.

3.2. Projenin Ana Hatları

3.2.1. Robota Raspberry Pi 3 Üzerinden Erişim

Robota Raspberry Pi 3 üzerinden erişebilmek için 'libkondo4' kütüphanesi kullanıldı. Bu kütüphane sayesinde, robot üzerine daha önceden Windows'a özgü olan programını(HeartToHeart4) kullanarak kaydettiğimiz hareketlerden herhangi birini seçerek, robota bu hareketi yaptırıldı.

3.2.2. Kullanıcı Arayüzü

Kullanıcının 'libkondo4' kütüphanesinin özelliklerini kullanabilmesi için bir web arayüzü tasarlandı. Bu arayüzün tasarımında 'HTML5', 'Javascript', 'Css, 'JQuery' , 'Web Socket' gibi teknolojiler kullanılmıştır.

3.3.3 Raspberry Pi 3 Üzerinde Web Sunucusu

Kullanıcı arayüzü ve 'libkondo4' kütüphanesi arasındaki bağlantıyı sağlamak adına Python programlama dilini kullanarak bir web sunucusu kodlandı. Bunun için Python dili için yazılmış 'Flask', 'Twisted', 'Autobahn' gibi kütüphaneler kullanıldı. Ayrıca kodlanan bu sunucu üzerinden 'libkondo4' kütüphanesinin tüm fonksiyonlarına da erişilebilmektedir.

3.3 Proje İş Bölümü

Projenin ilk kısmı olan 'libkondo4' kütüphanesi kullanarak robota erişme, Muratcan Ünsal tarafından yapıldı.

İkinci kısım olan arayüz tasarımı ve bu arayüz içinde kullanmış olduğumuz tüm teknolojiler Ali Kazar tarafından kodlandı.

Üçüncü kısım olan web sunucusu ve web sunucusunun 'libkondo4' ile bağlantıları Burak Akça tarafından Python dilinde kodlandı..

4.KULLANILAN KÜTÜPHANELER

Projenin yapımı sırasında kolaylık sağlaması için bir çok farklı kütüphane kullanılmıştır. Bunlardan 'libkondo4' robotla bağlantı kurmamızı sağlarken, web sunucusu üzerinde kullanılan 'Autobahn' kütüphanesi projeye web soket desteği sağlamıştır. Bu bölümde bu bilgilere kısaca değinilecektir.

4.1. 'libkondo4'

Robota linux tabanlı bir işletim sistemi olan Raspbian üzerinden ulaşmak için kullanılan açık kaynak kodlu bir 'C' kütüphanesidir. Chistopher Vo adlı şahıs tarafından yazılmış. Aşağıdakileri içermektedir

rcb4.c -> Kondo RCB-4HV seri servo kontrolörünü kontrol etmekte kullanılır.

ics.c -> Servoları ICS3.0 protokolüyle kontrol etmek icin kullanılır.

'libkondo4' fonksiyonlarının Java ve Python dillerinde kullanılabilmesine olanak sağlayan bağlayıcılar bulunmaktadır.[5]

4.2. 'Flask'

Python dilinde yazılmış mikro web kütüphanesi. Mikro denmesinin sebebi başka herhangi bir kütüphane gerektirmeden direkt Python diliyle beraber gelmektedir. Herhangi bir veritabanı soyutlama katmanı, form onaylama gibi bazı işlevleri sağlamaz programcı bunu kendisi eklemek durumundadır.[6]

4.3. Twisted

Twisted, etkiliğe dayalı(event-driven) bir network programlama kütüphanesidir. Twisted TCP, UDP, SSL/TLS, IP multicas, UNIX domain soketleri, bir çok protokol(HTTP, XMPP, NNTP, IMAP, SSH, IRC, FTP) desteklemektedir.

Olaya dayalı programlama bir progralama paradigmasıdır. Kısaca programın akışı kullanıcı tarafından tarafından yönlendirilir.[7]

Projede web soket desteği sağlamak için kullanılmıştır.

4.4 Autobahn|Python

Autobahn adlı projenin Python için olan alt projesidir. Twisted üzerinde çalışabilmektedir. [8]

Projede web soket kullanımını mümkün kılmıştır.

4.5 JQuery Mobile

JQuery Mobile, tüm akıllı telefon, tablet ve masaüstü cihazlarda erişilebilen responsive web siteleri ve uygulamaları yapmak için tasarlanmış bir HTML5 tabanlı kullanıcı arayüzü sistemidir.[9]Projenin arayüz tasarımında grid sistemi kullanılmıştır.[10]

5. GERÇEKLEME

Projenin çalışma mantığını anlamak için öncelikle "libkondo4" kütüphanesini ve web sunucusu üzerinden bu kütüphaneye nasıl ulaşıldığı iyi anlamak gerekmektedir.

Sırasıyla;

"libkondo4" kütüphanesine,

"RealKondo" sınıfına,

Kullanıcının arayüz üzerinden yapabileceği işlemlere değinilecektir.

5.1. "libkondo4"

C dilinde yazılmış bir kütüphane olup projenin en önemli kısmıdır. Sayesinde robotun üzerinde bulunan RCB-4HV kontrolörüne erişilinebilmektedir.

5.1.1. 'rcb4.h'

Bu kod parçası rcb4.c kodunun header kısmıdır.

Bu kodda kütüphanenin içinde kullanılan bir çok sabit verinin bir belirtildiği yerdir. Örneğin robotu bilgisayara bağlamak için kullandığımız USB-HS adaptörün VID, PID sabit değerleri burda atanır.

Ayrıca tüm rcb4.c fonksiyonlarında kullanılan KondoInstance "struct" ının tanımıda burda yapılmaktadır. Bu "struct"'ın aşağıda gösterildiği üzere 5 adet niteliği vardır.

```
// KondoInstrance verisi ------------------------------------------------------------
typedef struct
{
    struct ftdi_context ftdic; // ftdi durumu
    UCHAR swap[RCB4_SWAP_SIZE]; // swap dizisi
    char error[128]; // error mesajları
    UCHAR opt[RCB4_OPT_BYTES]; // option byteları
    int debug; // debug bilgilerini print edilip edilmeyeceği
} KondoInstance;

// typedefs -------------------------------------------------------------------
typedef KondoInstance* KondoRef;
```

Şekil 5.1. "KondoInstance" Fonksiyonu

5.1.2. ' rcb4.c'

Tüm operasyonları gerçekleştiren fonksiyonlar burada tutulmaktadır.

"libkondo4" kütüphanesini kullanabilmek için öncelikle robotun USB aygıtını Raspberry Pi 3 e bağlamamız gerekmektedir. Bunu yaptıktan sonra robotla bilgisayar arasında bağlantıyı sağlamak için kütüphanenin "kondo_init" veya "kondo_custom_init" fonksiyonlarını kullanılması gerekmektedir.

```
/*-------------------------------------------------------------------------------
 * Kondo Instance ı olusturur.
 * Baud, VID, PID için varsayılan degerleri kullanır.
 * Bu fonksiyon USB Adaptörüne erisir ve kullanıma hazır hale getirir.
 * Basarılıysa 0 dondurur.
 */
int kondo_init(KondoRef ki)
{
    return kondo_init_custom(ki, RCB4_BAUD, RCB4_USB_VID, RCB4_USB_PID,
            INTERFACE_ANY);
}
```

Şekil 5.2. “kondo_init” Fonksiyonu

```
/*-------------------------------------------------------------------------------
 * Kondo Instance ı olusturur.
 * Baud rate, vid, pid ve interface parametre olarak alır.
 * Bu fonksiyon USB Adaptörüne erisir ve kullanıma hazır hale getirir.
 * baud: the baud rate - e.g. 115200
 * vid: the USB vendor ID. Ornek icin rcb4.h ye bakılabilir.
 * pid: the USB product ID. Ornek icin rcb4.h ye bakılabilir.
 * interface: ftdi.h icinde tanımlanmıstır. Asagidakilerden biri secilebilir.
 *   INTERFACE_ANY, INTERFACE_A, INTERFACE_B, INTERFACE_C, INTERFACE_D
 * Basarılıysa 0 dondurur.
 */
int kondo_init_custom(KondoRef ki, int baud, int vid, int pid, int interface)
{
    assert(ki);
    int i;
    ki->debug = 0;

    // init usb
    if (ftdi_init(&ki->ftdic) < 0)
        kondo_ftdi_error(ki);

    // ilk interface i sec
    if (ftdi_set_interface(&ki->ftdic, interface) < 0)
        kondo_ftdi_error(ki);

    // usb aygitini ac
    if (ftdi_usb_open(&ki->ftdic, vid, pid) < 0)
        kondo_ftdi_error(ki);

    // baud rate i ata
    if (ftdi_set_baudrate(&ki->ftdic, baud) < 0)
        kondo_ftdi_error(ki);

    // hat parametreleri ata (8E1)
    if (ftdi_set_line_property(&ki->ftdic, BITS_8, STOP_BIT_1, EVEN) < 0)
        kondo_ftdi_error(ki);

    // ping robot
    if ((i = kondo_ack(ki)) < 0)
        return i;

    // options al
    if ((i = kondo_get_options(ki)) < 0)
        return i;

    return 0;
}
```

Şekil 5.3. “kondo_custom_init” Fonksiyonu

“kondo_init” fonksiyonu robotun adaptörüyle bilgisayar arasındaki bağlantıyı sağlayan, varsayılan olarak kullanılan fonksiyonudur. “rcb4.h” icinde atamış olduğumuz VID, PID gibi değişkenleri “kondo_custom_init” fonksiyonuna gönderir. Aslında tüm işi “kondo_init_custom” yapmaktadır. Biz kendi kodumuzdan direkt

olarak “kondo_init_custom” fonksiyonunu da kullanabilir ve istediğimiz parametreleri gönderebiliriz.

Kendi yazdığımız kodda işlemimiz bittiği zaman istersek “kondo_close” fonksiyonunu kullanabiliriz.

```
/*-------------------------------------------------------------------------
 * KondoInstance ı yok eder.
 * Bu fonksiyon USB adaptörünün kapatılmasından sorumludur.
 * Basarılıysa 0, degilse <0 dondurur.
 */
int kondo_close(KondoRef ki)
{
    assert(ki);

    // usb aygıtını kapat
    if (ftdi_usb_close(&ki->ftdic) < 0)
        kondo_ftdi_error(ki);

    // deinit
    ftdi_deinit(&ki->ftdic);

    return 0;
}
```

Şekil 5.4. “kondo_close” Fonksiyonu

Bu fonksiyon ile robotun adaptörü ile olan bağlantı kesilir.

```
/*-------------------------------------------------------------------------
 * İşlem şablonu: Temizle, robota out_bytes gonder, in_bytes al.
 * Hata varsa <0 doner
 * >=0 adet donen byte sayısını dondurur.
 */
int kondo_trx(KondoRef ki, int out_bytes, int in_bytes)
{
    assert(ki);
    int i;
    int j;

    if ((i = kondo_purge(ki)) < 0)
        return i;
    if ((i = kondo_write(ki, out_bytes)) < 0)
        return i;

    // debug printing
    if (ki->debug) {
        printf("send %d bytes: ", i);
        for (j = 0; j < i; j++)
            printf("%x ", ki->swap[j]);
        printf("\n");
    }

    i = kondo_read_timeout(ki, in_bytes, RCB4_RX_TIMEOUT);

    // debug printing
    if (ki->debug) {
        printf("recv %d bytes: ", i);
        for (j = 0; j < i; j++)
            printf("%x ", ki->swap[j]);
        printf("\n");
    }

    return i;
}
```

Şekil 5.5. “kondo_trx” Fonksiyonu

Tüm fonksiyonlarda ortak olarak kullanılan “kondo_trx” fonksiyonu robota gidecek olan verilerin sondan bir önceki durağıdır. “kondo_purge” ve “kondo_write” adında iki fonksiyon bu fonksiyonun içinden çağrılmaktadır.

```
/*-------------------------------------------------------------------------
 * TX ve RX bufferlarını temizle.
 * Başarılıysa 0 , hata varsa <0 dondurur.
 */
int kondo_purge(KondoRef ki)
{
    assert(ki);
    if (ftdi_usb_purge_buffers(&ki->ftdic) < 0)
        kondo_ftdi_error(ki);
    return 0;
}
```

Şekil 5.6. “kondo_purge” Fonksiyonu

“kondo_purge” ın görevi “TX” ve “RX” bufferlarını temizlemektedir. “kondo_write” ise robota gidecek olan ”KondoInstance” ‘ı robota gondermektedir.

```
/*-------------------------------------------------------------------------
 * Swap dizisinden robota "n" adet "byte" gonderilir.
 * Basariliysa gonderilmis olan byte sayısını "n" dondurur. Hata varsa <0
 */
int kondo_write(KondoRef ki, int n)
{
    assert(ki);
    int i;
    if ((i = ftdi_write_data(&ki->ftdic, ki->swap, n)) < 0)
        kondo_ftdi_error(ki);
    return i;
}
```

Şekil 5.7. “kondo_write” Fonksiyonu

En çok kullanılan fonksiyonları tanımlandıktan sonra şimdi projenin en önemli kısımlarını oluşturan 2 fonksiyon üzerinde durulacaktır.

“kondo_play_motion” fonksiyonu robotun üzerine daha önceden kendi programını kullanarak kaydetmiş olduğumuz hareketleri seçilen hareketin robot üzerindeki adresine göre çalıştırmamızı sağlamaktadır.

Bu işlem 4 aşamadan gerçekleşmektedir.

1- Çalışmakta olan EEPROM programı durdurulur.
2- Robot üzerinde istenilen hareketin adresini hesaplayıp bu adrese göre “swap” dizisini güncelleyip robota gönderilir.
3- EEPROM programını devam ettirilir.
4- Hareketin bitip bitmediği 50 sn de bir kontrol edilir.

```
/*-------------------------------------------------------------------------
 * Verilen hareket_id sine gore hareketi oynatır.
 * Timeout zamanı gecene kadar ya da hareket bitene kadar etkileşime izin vermez.
 * Bu izinden kurtulmak icin timeout = 0 verilebilir.
 * Dondurur < 0: Error
 * Dondurur 0: Her sey yolunda
 */
int kondo_play_motion(KondoRef ki, UINT num, long timeout)
{
    assert(ki);
    int i;
    UCHAR chk;
    UINT mot_addr;

    // 4 basamaklı bir işlemdir.
    // 1. Calismakta olan EEPROM programını durdur.
    // 2. Hareket scriptini cagir.
    // 3. EEPROM u devam ettir.
    // 4. Bitip bitmediğini her 50 saniyede bir kontrl et.

    // (1) Calismakta olan EEPROM programını durdur. ---------------------------------------
    // Bunu yapmak icin EEPROM kapatıp PGC yi hafızaya yazmanınız gerekir
    // ki sonra burdan eski haline gelebilsin.

    ki->opt[0] &= ~RCB4_OPT_EEPROM; // disable eeprom
    ki->opt[0] &= ~RCB4_OPT_RESP; // servo responsunu kapat
    ki->opt[0] |= RCB4_OPT_SIO; // servoların calistigindan emin l
    ki->swap[0] = 19; // number of bytes
    ki->swap[1] = RCB4_CMD_MOV; // move command
    ki->swap[2] = RCB4_COM_TO_RAM; // com-->ram
    ki->swap[3] = (UCHAR) (RCB4_ADDR_OPT); // option ram L
    ki->swap[4] = (UCHAR) (RCB4_ADDR_OPT >> 8); // option ram M
    ki->swap[5] = (UCHAR) (RCB4_ADDR_OPT >> 16); // option ram H
    ki->swap[6] = ki->opt[0]; // option data
    ki->swap[7] = ki->opt[1]; // option data
    ki->swap[8] = (UCHAR) (RCB4_ADDR_MAIN); // eeprom program main L
    ki->swap[9] = (UCHAR) (RCB4_ADDR_MAIN >> 8); // eeprom program main M
    ki->swap[10] = (UCHAR) (RCB4_ADDR_MAIN >> 16); // eeprom program main H
```

Şekil 5.8. "kondo_play_motion" Fonksiyonu 1. Kısım

```
    ki->swap[11] = 0;
    ki->swap[12] = 0;
    ki->swap[13] = 0;
    ki->swap[14] = 0;
    ki->swap[15] = 0;
    ki->swap[16] = 0;
    ki->swap[17] = 0;
    chk = kondo_checksum(ki, 18);
    ki->swap[18] = chk;

    // 19 byte gonder 4 byte bekle
    if ((i = kondo_trx(ki, 19, 4)) < 0)
        return i;

    // response un checksum ını onayla.
    if (i != 4 || ki->swap[18] != chk)
        kondo_error(ki, "Bad response trying to stop EEPROM.");

    // (2) Hareket scriptini cagir. ---------------------------------------------------
    // Hareket adresi hesaplanır. (num*2048 + 3000).
    mot_addr = RCB4_MOT_SIZE * num + RCB4_ADDR_MOT_BASE;
    ki->swap[0] = 7; // num bytes
    ki->swap[1] = RCB4_CMD_CALL; // command
    ki->swap[2] = (UCHAR) (mot_addr); // motion address l
    ki->swap[3] = (UCHAR) (mot_addr >> 8); // motion address m
    ki->swap[4] = (UCHAR) (mot_addr >> 16); // motion addres h
    ki->swap[5] = 0;
    ki->swap[6] = kondo_checksum(ki, 6);

    // 7 byte gonder 4 byte bekle
    if ((i = kondo_trx(ki, 7, 4)) < 0)
        return i;

    // response u onayla (ACK)
    if (i != 4 || ki->swap[2] != RCB4_ACK_BYTE)
        kondo_error(ki, "Bad response trying to call the motion.");
```

Şekil 5.9. "kondo_play_motion" Fonksiyonu 2. Kısım

```
    // (3)EEPROM u devam ettir ----------------------------------------------------
    ki->opt[0] |= RCB4_OPT_EEPROM; // enable EEPROM
    ki->opt[0] &= ~RCB4_OPT_VEC; // vector jump flag temizle
    ki->swap[0] = 9; // num bytes
    ki->swap[1] = RCB4_CMD_MOV; // move command
    ki->swap[2] = RCB4_COM_TO_RAM; // com-->ram
    ki->swap[3] = (UCHAR) (RCB4_ADDR_OPT); // option addr L
    ki->swap[4] = (UCHAR) (RCB4_ADDR_OPT >> 8); // option addr M
    ki->swap[5] = (UCHAR) (RCB4_ADDR_OPT >> 16); // option addr H
    ki->swap[6] = ki->opt[0]; // option low byte
    ki->swap[7] = ki->opt[1]; // option high byte
    ki->swap[8] = kondo_checksum(ki, 8); // checksum

    // 9 byte gonder 4 byte bekle
    if ((i = kondo_trx(ki, 9, 4)) < 0)
        return i;

    // response u onayla (ACK)
    if (i != 4 || ki->swap[2] != RCB4_ACK_BYTE)
        kondo_error(ki, "Bad response while trying to restart EEPROM.");

    // (4) 4. Bitip bitmediğini her 50 saniyede bir kontrl et. ------------------

    // simdiki zamanı al
    static struct timeval tv, end;
    gettimeofday(&tv, NULL);

    // zamanı hesapla
    end.tv_sec = tv.tv_sec + timeout / RCB4_SECOND;
    end.tv_usec = tv.tv_usec + timeout % RCB4_SECOND;
    if (end.tv_usec > RCB4_SECOND) {
        end.tv_sec += 1;
        end.tv_usec -= RCB4_SECOND;
    }
```

Şekil 5.10. “kondo_play_motion” Fonksiyonu 3. Kısım

```
    //time out suresi gelmedikce
    while ((tv.tv_sec < end.tv_sec) || (tv.tv_usec < end.tv_usec)) {
        // options ları al, vector jump flag ı temizle.
        if ((i = kondo_get_options(ki)) < 0)
            return i;
        if ((ki->opt[0] & RCB4_OPT_VEC) == RCB4_OPT_VEC)
            break;

        // options verilerini yuklemesi 50sn almaktadır.
        // 50 saniye bekle.
        usleep(RCB4_50MS);

        //zamanı kontrol et
        gettimeofday(&tv, NULL);
    }

    return 0;
}
```

Şekil 5.11. “kondo_play_motion” Fonksiyonu 4. Kısım

Servo motorların teker teker kontrolü ise projenin bir diğer önemli kısmıdır. Burda iki fonksiyon peşpeşe kullanılmaktadır. "kondo_set_servo_pos" fonksiyonu, değerini değiştirmek istediğimiz servo motorun "id" sini alarak robota gönderilecek komuta eklenmek üzere 5 byte lık "servos" adında bir dizi hazırlar. Hazırlanan bu diziyi dönüş miktarı "frame" ile beraber "kond_send_ics_pos" adlı fonksiyona çalıştırmak için kullanır.

```
int kondo_set_servo_pos(KondoRef ki, UINT servo_id, UINT frame)
{

    //Tek bir servo yu kontrol etmek icin kullanılır.
    //Istenen servo_id ye gore, kondo_send_ics_pos gitmek uzere istenen servos
    //bitfield ini hazırlar ve gonderir.

    UCHAR servos[5];
    int no;
    for(no=0; no<5; no++)
        servos[no] = 0b0000000;

    div_t divresult;
    divresult = div (servo_id,8);
  servos[4 - divresult.quot] = 0b00000001 << divresult.rem;

    return kondo_send_ics_pos(ki, servos, frame);

}
```

Şekil 5.12. "kondo_set_servo_pos" Fonksiyonu

“kondo_send_ics_pos” fonksiyonu ise robota gitmek üzere “swap” dizisini baştan ayarlar. Parametre olarak aldığı “servos” dizisinin elemanlarını “swap” dizisine yerleştirdikten sonra kullanılan servo motorun yeni “frame” değerini de “swap” dizisine ekler.

```
int kondo_send_ics_pos(KondoRef ki, UCHAR servos[5], UINT frame)
{
    assert(ki);
    int ret;

    // ICS frame i gondermek icin komut yap.
    ki->swap[1] = RCB4_CMD_ICS; // ICS komutu
    ki->swap[2] = servos[4];
    ki->swap[3] = servos[3];
    ki->swap[4] = servos[2];
    ki->swap[5] = servos[1];
    ki->swap[6] = servos[0];
    ki->swap[7] = 0x1;

    // kullanılan servoları say
    int c, i;
    for (c = 0, i = 0; i < 5; i++)
        for (; servos[i]; c++)
            servos[i] &= servos[i] - 1;

    // her bir servo icin frame hazırla
    for (i = 8; i < 8 + (c * 2); i += 2) {
        ki->swap[i] = (UCHAR) (frame);
        ki->swap[i + 1] = (UCHAR) (frame >> 8);
    }

    ki->swap[0] = i + 1;
    ki->swap[i] = kondo_checksum(ki, i);


    if ((ret = kondo_trx(ki, i + 1, 4)) < 0)
        return i;

    if (ki->swap[1] != RCB4_CMD_ICS)
        kondo_error(ki, "Response was not valid");

    return 0;
}
```

Şekil 5.13. “kondo_send_ics_pos” Fonksiyonu

5.1.3’ ics.c ‘

Bu kütüphane, robotun RCB-4HV kontrolörü yerine yarı-çift yönlü bir devre kullanarak servo motorlara doğrudan bağlanmak üzere kullanılır.[11] Projede üzerinde bir çalışma yapmadığımız için değinilmeyecek.

5.2. RealKondo

Web sunucusu üzerinde bulunan RealKondo sınıfının görevi sunucu ve “libkondo” fonksiyonları arasındaki bağlantıyı sağlamaktır. Bu sınıf sayesinde “rcb4.c” içindeki

fonksiyonları kullanabilmekteyiz. Aşağıda sınıfın kodları gösterilmektedir. Sunucu kodu çalıştığında "realKondo" nesnesi bu sınıftan türetilmektedir.

```
class RealKondo(object):
    def __init__(self):
        self.max_wait = 50 * 1000000
        self.ki = KondoInstance()
        ret = kondo_init(self.ki)
        print(self.ki.error)

    def setServoPos(self, id, frame):
        ret = kondo_set_servo_pos(self.ki, int(id), int(frame))

    def playMotion(self, motion_id):
        print ("Playing Kondo Motion %", motion_id)
        ret = kondo_play_motion(self.ki, motion_id, self.max_wait)
    def hareketEt(self, pos):
        #Calismiyor
        ret = kondo_hareket(self.ki, pos)
        if ret < 0:
            sys.exit(self.ki.error)
    def close(self):
        ret = kondo_close(self.ki)
        if ret < 0:
            sys.exit(self.ki.error)
```

Şekil 5.14 "RealKondo" Sınıfı

Sınıfın kurucusu olan "__init__" metodu içinde "libkondo4" kısmında anlattığımız KondoInstance nesnesi oluşturulmaktadır. Böylece robota bağlanmış olunur. Ayrıca "rcb4." İçindeki "kondo_play_motion" fonksiyonun bir parametresi olan "long timeout" için belirttiğimiz "max_wait" süreside bu kurucu içinde bir değişkene atanır. Bunu kütüphanenin kendi örneklerini inceledikten sonra 50 sn olarak kullanmaya karar verdik. Bu değeri değiştirmek için sunucuyu başlatmadan önce RealKondo sınıfının "__init__" metodu modifiye edilebilir.

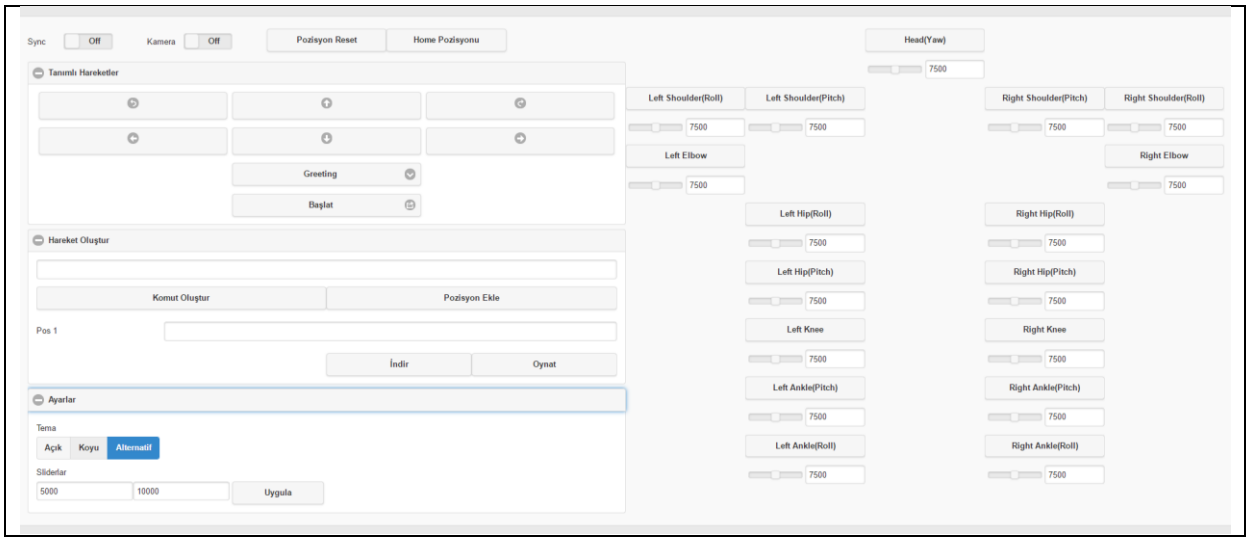
Sınıfın bir diğer metodu ise "playMotion" sayesinde robotun "kondo_play_motion" fonksiyonuna erişim sağlanılmaktadır.

Bir diğer metod olan "setServoPos" üzerinden robot üzerindeki 17 adet servo motoru tek tek kontrol edebilmekteyiz. Bu metod "rcb4.c" içindeki "kondo_set_servo_pos" fonksiyonunu hareke geçirir. Bu fonksiyonsa gidicek "byte array"ı düzenleyen fonksiyon olan "kondo_send_ics_pos" fonksiyonunu çalıştırır.

"close" metodu ile kullanılan KondoInstance nesnesi tamamen yok edilir.

5.3. Kullanıcı Arayüzü

Kullanıcının web arayüzünde yapabileceği aksiyonlardan bahsedilecek. Bu aksiyonun arayüz kısmında hangi kodu tetiklediği, tetiklenen kod parçasının sunucuya kullanıcının seçtiği parametreleri nasıl aktardığı, sunucunun gelen bu parametrelere nasıl tepki verdiği ve hangi 'libkondo4' fonksiyonunu kullandığı üzerinde durulacaktır.



Şekil 5.15. Kullanıcı Arayüzü

Yukarda görülen olan arayüz robotu hareket ettirmekte kullanılmaktadır.

Ekranın sağ kısmında 17 adet kaydırıcı(slider) bulunmaktadır. Bu kaydırıcılar vasıtasıyla robot üzerinde bulunan 17 adet servo motoru anlık olarak kontrol edebilmekteyiz.

Ekranın sol kısmında ise birden fazla aksiyona olanak sağlayan kısımlar bulunduğu için teker teker anlatılacak.

Sync: "On/Off" mantığına göre çalışmaktadır. Eğer "On" ise kullanıcı anlık olarak servo motorları hareket ettirebilmektedir. "Off"durumunda ise hareket ettiremez. Bunu eklenmesinin sebebi hem kendi özgün programı olan "HeartToHeart4" benzer bir sistemin bulunması hem de güvenlik sebebiyle.

Kamera: Robotun başı üzerinde bulunan "Pi Camera" sayesinde sunucudan arayüze anlık görüntü aktarımını sağlamaktadır. "On/Off" mantığına göre çalışmaktadır.

Reset: Kullanıcı ekranın sağ kısmında bulunan kaydırıcılar üzerinde bir değişiklik yapabilir ve daha sonra en başta bulunan duruma getirmek isterse diye eklendi. Esas amacına "Hareket Oluştur" kısmı açıklanırken değinilecektir.

Home Position: Robotun tüm motorlarının başlangıçta bulunduğu değer "7500" dür. Bu butona tıklandığında sunucudan robota tüm motorlarının bu noktaya geri getirilmesi komutu verilmesi istenir.

Kayıtlı Hareketler: Daha önce değindiğimiz gibi robotun üzerine kendi programını kullanarak hazır olarak elimizde bulunan bir takım hareket bilgilerini yüklemiştik. Arayüzün oynatmak istediğimiz hareketi seçerek sunucuya iletebilmekteyiz. Sunucuda gelen hareket adına göre 'libkondo4' üzerinden denk gelen fonksiyonu çağırarak robotun istenilen hareketi yapmasını sağlar.

Hareket Oluştur: Tam anlamıyla bitmemiş olmakla beraber 'Komut Oluştur' kısmı tam olarak doğru çalışmaktadır.

Bu kısımdaki amaç kullanıcıya robota seri hareket yaptırmasına olanak sağlamaktır. Bu seri hareket in yapılması için robota sahneler gönderilir. Her sahnede ekranın sağ tarafındaki 17 adet servo motor kontrol kısmından ayarlanabilir. Tüm bu işlemler yapılırken "Sync", "Off" durumunda olmalıdır. İlk sahne hazırlandıktan sonra "Komut Oluştur"Butonuna tıklanırsa robota gidecek "byte"lar hazırlanmış olur.

"Pozisyon Ekle" butonu sayesinde istenen sayıda sahne eklenebilir.

```
function komutMaker(servos_used, servo_count, values) {
    var servos = [0, 0, 0, 0, 0];

    for (id_no in servos_used) {

        var divQuot = parseInt(servos_used[id_no] / 8);
        var divRem = servos_used[id_no] % 8;
        servos[4 - divQuot] += 1 << divRem;
    }
    var swap = [];
    swap[1] = 0x10;
    swap[2] = servos[4];
    swap[3] = servos[3];
    swap[4] = servos[2];
    swap[5] = servos[1];
    swap[6] = servos[0];
    swap[7] = 0x1;
    var i;
    var value_index = 0;
    for (i = 8; i < 8 + (servo_count * 2) ; i += 2) {
        swap[i] = values[value_index]%256
        swap[i + 1] = values[value_index] >> 8;
        value_index++;
    }
    swap[0] = i + 1;
    swap[i] = kondo_checksum(swap);
   var komut = "";
    for (j = 0; j < swap.length; j++) {
        var hexStr = swap[j].toString(16).toUpperCase();
        komut += hexStr + " ";
        //console.log(hexStr);
    }
```

Şekil 5.16 "komutMaker" Fonksiyonu

Ayarlar: Bu ekranda arayüzün teması değiştirilebilir. Sadece görüntüyü değiştirmek amaçlı kullanılmaktadır. Ayrıca ekranın sağındaki kaydırıcıların "minimum" ve "maximum" değerleri tekrar ayarlanılabilir.

5.4 İşlemlerin Detaylı Açıklaması

Kullanıcının sunucuyla dolaylı olarak 'libkondo4' fonksiyonlarıyla etkileşimi iki şekilde olmaktadır. Bu etkileşimlerde farklı 'libkondo4' fonksiyonları çalışmaktadır.

5.4.1 Kayıtlı Hareketler

Sunucu arayüzün "Kayıtlı Hareketler" kısmındaki 'dropdown' menüden istediği hareketi seçer ve "Başlat" butonuna tıklar. Bu noktadan sonra aşağıdaki Javascript kodu çalışır.

```
$("#secilmis_hareket").click(function () {
    var secilen_hareket = $('select[name=selector]').val();
    console.log(secilen_hareket);
    KondoAction("hareket", secilen_hareket);

});
```

Şekil 5.17. "secilmis_hareket" Olayı

```
function KondoAction(komut, hareket_adi) {

    data = {
        "komut": komut,
        "hareket_adi": hareket_adi
    }
    $.ajax({
        type: 'POST',
        url: '/',
        data: JSON.stringify(data, null, '\t'),
        contentType: 'application/json;charset=UTF-8',
    })
}
```

Şekil 5.18. "KondoAction" Fonksiyonu

"KondoAction" adlı fonksiyon "hareket_adi" adında bir parametre alır. Bu parametreye göre bir Javascript nesnesi yapılır.

Sayfayı yenilemeden sunucuya bilgi göndermemizi sağlayan bir teknoloji olan "Ajax" kullanılarak bu "hareket_adi" adlı bilgi sunucuya istek(request) olarak gönderilir.

Aşağıda gelen bilgi sonrasında sunucunun yaptığı işlemler gösteriliyor.

```
@app.route('/', methods=['POST', 'GET'])
def hello_world():
    if request.method == "POST":
        try:
            hareket_adi = request.json['hareket_adi']
            realKondo.playMotion(motions[hareket_adi])
        except Exception as e:
            print (e.message)

    return render_template('tasarim.html')
```

Şekil 5.19. "hello_world" Fonksiyonu

Öncelikle gelen "istek" metodunun "POST" olup olmadığını kontrol eder. Öyleyse eğer Python da yaratmış olduğumuz bir sözlük(dictionary) nesnesi içinden istenilen hareketin "index" numarası bulunur.

```
motions = {
    "Greeting": 0,
    "Home Position": 1,
    "Wave": 2,
    "HipHipHipHurray": 3,
    "Chagrined": 4,
    "Headstand": 5,
    "Clap": 6,
    "10 Claps": 7,
    "Rythm Claps": 8,
    "Push-Ups": 9,
    "One Legged Bend": 10,
    "Bunny Hop A": 11,
    "Bunny Hop B": 12,
    "Stand-Up Stomach": 13,
    "Stand-Up Back": 14,
    "Safewalk Forward": 15,
    "Safewalk Backward": 16,
    "Safewalk Left": 17,
    "Safewalk Right": 18,
    "Quickturn Left": 19,
    "Quickturn Right": 20,
    "Regular Walk Forward": 21,
    "Regular Walk Back": 22,
    "Regular Walk Left": 23,
    "Regular Walk Right": 24,
    "Kick Ball Fwd Left": 25,
    "Kick Ball Fwd Right": 26,
    "Kick Ball Side Left": 27,
    "Kick Ball Side Right": 28,
    "Kick Ball Backwd Left": 29,
    "Kick Ball Backwd Right": 30
}
```

Şekil 5.20. "motions" Dictionary

Bu index numarası ise RealKondo sınıfından türetilmiş bir nesne olan "realKondo" nesnesinin "playMotion" metodunun parametresi olur.

Bu işlemler sonunda "kondo_play_motion" fonksiyonu çalışır ve robot istenilen hareketi gerçekleştirir.

5.4.2 Tek Motor Kontrolü

Projenin bir diğer amacı olan tek tek tüm servo motorları kontrolünü sağlamak üzere 'web soket' kullanılmıştır.

Web soket, tek bir TCP bağlantı üzerinden istemci ile sunucu arasında full-duplex iletişimi sağlayan bir iletişim protokolüdür.

Kullanıcı motorları anlık olarak kontrol etmek istiyorsa "Sync" in durumu "On" da olmalıdır.

Herhangi bir slider hareket ettirildiğinde aşağıdaki Javascript kodları çalışmaktadır.

```
$(".changeEvent").change(function () {
    var mySlider = $(this).find("input");
    var degistirilecek_textbox = mySlider.attr("id").split("_")[1];
    var servo_id = mySlider.attr("id").split("_")[2];
    var frame = $(mySlider).val(); //deger bulunuyor
    $("#text_" + degistirilecek_textbox + "_" + servo_id).val(frame);
    if (myToggle() == "On") {
        gonder(servo_id, frame);
    }
});
```

Şekil 5.21. "changeEvent" Olayı

```
function gonder(servo_id, frame) {
    if (sock) {
        //var komut = "seri_hareket";
        var servoIdAndFrame = { "komut": "tek_motor_kontrol" , "servo_id": servo_id, "frame": frame }

        sock.send(JSON.stringify(servoIdAndFrame))
        console.log("Sent: " + servoIdAndFrame);
    } else {
        console.log("Not connected.");
    }
}
```

Şekil 5.22. "gonder" Fonksiyonu

Kaydırıcılardan herhangi birinin değeri değiştiğinde, tüm hepsinin ortak sınıfı olan ".changeEvent" sınıfı üzerinde tanımlamış olduğumuz "change" olayı gerçekleşir. Bu olayın iki amacı vardır.

1- Kaydırıcının yanında bulunduğu "Textbox" un değerini kaydırıcının yeni değerini vermek.
2- Hangi motorun değerinin değiştirildiği bilgisini(servo_id) ve değişim miktarını(frame) alarak, "gonder" fonksiyonunu bu parametrelerle çağırmak.

"gonder" fonksiyonu gelen bilgileri alarak öncelikle hali hazırda bir soket bağlantısı olup olmadığını kontrol eder. Soket bağlantısı varsa "servo_id" ve "frame" değişkenlerini tutan bir nesne yaratır. Daha sonra soket kullanarak bu bilgiler sunucuya iletilir.

Gelen bu bilgiler sunucuda bulunan aşağıdaki kod parçasını çalıştırır.

```
class EchoServerProtocol(WebSocketServerProtocol):

    def onMessage(self, payload):
        message = payload.decode('utf8')
        message = json.loads(message)
        komut = message["komut"]
        if komut == "tek_motor_kontrol":
            servo_id = message["servo_id"]
            frame = message["frame"]
            print servo_id + frame
            realKondo.setServoPos(servo_id, frame)
        elif komut == "hareket_oynat":
            print "Henuz Eklenmedi"

        self.sendMessage(payload)
```

Şekil 5.23. "EchoServerProtocol" Sınıfı

Twisted kütüphanesi kullanılarak kodlanan bu protokolün amacı gelen bilgileri ayıklamak ve robotu hareket ettirecek kodları tetiklemektir.

Gelen değişkenlerle, RealKondo sınıfının "setServoPos" metodu çağrılmaktadır. Bu method ise "libkondo4" kütüphanesinin "kondo_set_servo_pos" fonksiyonunu tetikler.

Bu fonksiyon gelen "servo_id" ye göre "servos" adında 5 baytlık bir dizi hazırlar. Oluşan bu dizi tamamiyle verilen "servo_id" ye bağlıdır. Hazırlanan bu servos dizisi ise "frame" ile beraber "kondo_send_ics_pos" fonksiyonunu tetikler. Bu fonksiyon ise hareket etmesi istenilen servo motoru istenilen miktarda hareket ettirir.

6. SONUÇ ve ÖNERİLER

6.1. Sonuç

Sonuç olarak proje kullanıcıya 3 önemli özellik sunmaktadır.

1- Robot üzerindeki kayıtlı hareketleri oynatabilme,
2- Tek tek tüm servo motorları kontrol edebilme.
3- Arayüzün sağ tarafındaki motorlarlara verilen değerlere göre RCB-4HV ye gidebilecek bir komut oluşturulabilir. Bu kısım kendi programıyla aynı komutu üretmektedir.

6.2. Olumsuzluklar

Olumlu yanlarının yanı sıra, tüm denemelerimize rağmen sahnelerden kurulu özel bir hareket oluşturup, robotun bu hareketi yapmasını sağlayamadık.

Bunu başarmak için izlenen yol şöyleydi. Kendi programı olan HeartToHeart4 üzerinden bir hareket oluşturduk ve bunu robota gönderdik. Robota aşağıda gösterilen şekilde göndermektedir.

Bizde bunu "libkondo4" üzerinden taklit ederek yapmak istedik. Öncelikle her sahne için bir komut oluşturucak bir sistem tasarladık. Kendi programı ile kıyaslayınca bu kısımda bir hata gözükmemektedir.

```
function komutMaker(servos_used, servo_count, values) {
    var servos = [0, 0, 0, 0, 0];

    for (id_no in servos_used) {

        var divQuot = parseInt(servos_used[id_no] / 8);
        var divRem = servos_used[id_no] % 8;
        servos[4 - divQuot] += 1 << divRem;
    }
    var swap = [];
    swap[1] = 0x10;
    swap[2] = servos[4];
    swap[3] = servos[3];
    swap[4] = servos[2];
    swap[5] = servos[1];
    swap[6] = servos[0];
    swap[7] = 0x1;
    var i;
    var value_index = 0;
    for (i = 8; i < 8 + (servo_count * 2) ; i += 2) {
        swap[i] = values[value_index]%256
        swap[i + 1] = values[value_index] >> 8;
        value_index++;
    }
    swap[0] = i + 1;
    swap[i] = kondo_checksum(swap);
   var komut = "";
    for (j = 0; j < swap.length; j++) {
        var hexStr = swap[j].toString(16).toUpperCase();
        komut += hexStr + " ";
        //console.log(hexStr);
    }
```

Şekil 6.1. "komutMaker" Fonksiyonu

Kullanıcı bu sahneleri sunucuya gönderince sunucu bir "for" döngüsü içinde bizim eklemiş olduğumuz bir fonksiyon olan "hareketEt" fonksiyonuna bu komutu gondermektedir. Hareket et fonksiyonu aşağıdaki gibi çalışmaktadır.

Kendi eklediğimiz kısım başta olmak üzere robota esas veriyi iletecek olan kısmı "kondo_send_ics_pos" fonksiyonundan aldık.

```
UCHAR kondo_hareket(KondoRef ki, char * hex) {

    int ret;
    const char s[2] = " ";
    char *token;
    token = strtok(hex,s);
    int p = 0;
    while(token != NULL)
    {   printf(" %s\n", token);
        ki->swap[p] = *token;
        token = strtok(NULL,s);
        p++;
    }

    if ((ret = kondo_trx(ki, p + 1, 4)) < 0)
        return p;


    if (p != 4 || ki->swap[1] != RCB4_CMD_ICS)
        kondo_error(ki, "Response was not valid");


    UINT so = 5;
    return (UCHAR) so;
}
```

Şekil 6.2. "kondo_hareket" Fonksiyonu

Bu kodun robotu hareket ettirmemesinin sebebi bizim yazdığımız C kodunun yetersiz olması olabilir. Ama başka bir yol daha denedik. Bu yol aslında gayet kötü olmakla beraber sadece çalışıp çalışmadığını kontrol etmek içindi.

Bilindiği üzere "libkondo4" içindeki KondoInstance içindeki "swap" unsigned char dizi robota gidecek olan "byte" ların tutulduğu yerdi. Bu "swap" arrayın değerlerini aşağıdaki gibi atamayı denedik.

"ki->swap[0] = 43"

.....

Bu yol her ne kadar yanlış olsa da test amaçlı kullanılmıştır. Aslında "libkondo4" kendi içindeki tüm fonksiyonlara bu şekilde kullanmaktadır.

Bu yöntemle de çalıştıramayınca dönem sonuna geldiğimiz için bu kısım üzerinde çalışmayı durdurduk.

Bu sorunu çözmeye çalışırken çeşitli araştırmalar yaptık. Bu araştırmanın sonucunda "github" ta , "amuthelet" adlı bir kullanıcının kodunu bulduk.[12] Koddan anladığımız üzere kendisi "Blender 3D" adında bir program kullanmaktadır. Bu program aynı zamanda Python dilinde de yazmamızı sağlamaktadır.

Vaktimiz kalmadığı için bu kodu daha detaylı inceleyemedik.

6.3. Alternatif Yollar

Araştırmalarımız sırasında bulduğumuz, “robosavvy” forumunun üyeleri tarafından yazılmış sadece Python dili kullanılarak, robota ulaşılmıştır. Bu kod gayet sade olmakla beraber [13]“libkondo4”kütüphanesinin sağladığı esnekliklerin hiç birini sağlamamaktadır. Programcı isterse “libkondo4”ün tüm özelliklerini Python da yazmayı deneyebilir. Bu denemede “pySerial” kütüphanesi çok önemli bir yer tutmaktadır.

6.4. RCB4 Command Generator

Araştırmalar sırasında bulmuş olduğumuz bir başka program. Bu programı “kondo-robot” kendi sitesinden yayınlamıştır.

Bu program “komut oluşturma” mantığını çözmemişte çok işimize yaradı. Program üzerinde tek tek veya birden fazla motoru seçerek komut üretmek mümkün. Aynı şekilde robota göndermek de mümkündür.

Programın en önemli yanı ise bize “rcb4.dll” dll ini vermesidir. Bu dll sayesinde HeartToHeart4 programının muhtelemen tüm özelliklerini kendi yazacağımız programda kullanabilmekteyiz. Ama bu “dll” Windows ortamında çalışmaktadır haliyle. Raspberry Pi 3 üzerine Windows IOT kurularak belki bu sorun giderilebilir.[14]

KAYNAKLAR

1) http://kondo-robot.com/product/hearttoheart4

2) https://github.com/galatasarayuniversity/libkondo4

3) https://robosavvy.com/store/kondo-khr-3hv-r2-humanoid-robot-kit-exc-battery-and-charger.html

4) http://kondo-robot.com/product/02042

5) https://bitbucket.org/vo/libkondo4/wiki/Home

6) http://flask.pocoo.org/

7) https://twistedmatrix.com/trac/

8) https://github.com/crossbario/autobahn-python

9) https://jquerymobile.com/

10) https://www.w3schools.com/jquerymobile/jquerymobile_grids.asp

11) https://robosavvy.com/forum/viewtopic.php?f=2&t=5877&start=15

12) https://github.com/amuthelet/kondo

13) http://robosavvy.com/forum/viewtopic.php?t=7862

14) http://kondo-robot.com/faq/rcb-4-reference-ver-2_2

ÖZGEÇMİŞ