İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ORTAK MİMARİLİ YAPILAR İLE İNSANSIZ ARAÇLARDA VERİ İLETİŞİMİ

YÜKSEK LİSANS TEZİ

Okan BOSTAN

Mekatronik Mühendisliği Anabilim Dalı

Mekatronik Mühendisliği Programı

EKİM 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ORTAK MİMARİLİ YAPILAR İLE İNSANSIZ ARAÇLARDA VERİ İLETİŞİMİ

YÜKSEK LİSANS TEZİ

Okan BOSTAN
(518101032)

Mekatronik Mühendisliği Anabilim Dalı

Mekatronik Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Hakan Temeltaş

EKİM 2015

İTÜ, Fen Bilimleri Enstitüsü'nün 518101032 numaralı Yüksek Lisans Öğrencisi **Okan BOSTAN**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"ORTAK MİMARİLİ YAPILAR İLE İNSANSIZ ARAÇLARDA VERİ İLETİŞİMİ"** başlıklı tezini aşağıdaki imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Prof. Dr. Hakan Temeltaş** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Yrd.Doç.Dr. Sıddık Murat Yeşiloğlu** İstanbul Teknik Üniversitesi	.....
	Yrd.Doç.Dr. Özgür Turay Kaymakçı Yıldız Teknik Üniversitesi	.....
		.....

Teslim Tarihi : **11 Eylül 2015**
Savunma Tarihi : **07 Ekim 2015**

Aileme,

ÖNSÖZ

Yaptığım çalışma ve araştırmalar süresinde bana yol gösteren ve destekleyen tez danışmanım Prof. Dr. Hakan Temeltaş'a, robotik laboratuarındaki çalışmalarımda bana yardım eden Onur Şencan'a ve son olarak hayatım boyunca desteğini ve sevgisini esirgemeyen aileme teşekkürü borç bilirim.

Ekim 2015 Okan BOSTAN

İÇİNDEKİLER

Sayfa

KISALTMALAR

ACTIVE : Applied Cognition and Training in Immersive Virtual Enviroments
ARM : Advanced RISC Machines
HTTP : Hyper Text Transfer Protocol
IST : Institute for Simulation and Training
IGVC : Intelligent Ground Vehicle Competition
IP : Internet Protocol
JAUS : Joint Architecture for Unmanned Systems
OCU : Operator Control Unit
RA : Reference Architecture
RISC : Reduced Instruction Set Computer
ROS : Robot Operating System
SSH : Secure Shell
SMB : Server Message Block
SAE : Society of Automative Engineers
TCP : Transmission Control Protocol
UCF : University of Central Florida
UDP : User Datagram Protocol

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

ORTAK MİMARİLİ YAPILAR İLE İNSANSIZ ARAÇLARDA VERİ İLETİŞİMİ

ÖZET

Verinin bir ağ üzerinde herhangi bir kaynaktan bir hedefe yol alabilmesi için gerekli olan en önemli ölçüt ağı oluşturan tüm elemanların aynı dili konuşması ya da aynı kurallar ailesine uyarak haberleşmeleridir. Ortak protokol kullanılmasındaki amaç diğer istemcilerin tek bir yazılım kullanarak birbirinden farklı siteler açabilmesi, tek yazılımla birden çok dosya paylaşımına bağlanabilmeleridir. Bilgisayar sistemlerinde en yaygın olarak kullanılan kurallar bütünü ve protokol TCP/IP (Transmit Control Protocol / Internet Protocol)'dir. Robotik sistemlerde de kullanılan bilgisayar ve hesaplama sistemleri TCP/IP protokolünü kullanarak robot ile Kontrol Ünitesi ve ya operatör arasında veri iletişimini sağlayabilirler. Bu çözümde, haberleşme veri yapısının her robot geliştirici için farklı ve birbiriyle uyumsuz olması ile farklı kaynaklardan çıkan insansız sistemler için geliştirme ve değiştirmenin sadece tasarlayan kişi tarafından yapılabilmesi başlıca sorunları temsil etmektedir. JAUS (Joint Architecture for Unmanned Systems) referans modeli, bu eksiklikleri gidermekte ve robot sistemlerinde esneklik, erişilebilirlik ve güvenilirlik sağlamaktadır. JAUS dünya üzerinde birçok farklı üreticinin ve firmanın ürettiği değişik tipteki insansız araçların birlikte çalışabilir (interoperability) olmasını garantileyen bir dizi standartları belirleyerek üreticilere ve geliştiricilere bu desteği vermiştir. Birden fazla üretici ve araştırmacı birbirinden farklı platformlarda ve farklı fiziksel donanımlarda kendi insansız sistemlerini geliştirmektedir. JAUS farklı üreticilerin geliştireceği insansız sistem parçalarını ve bileşenlerini standartlaştırır, değişik tipte donanım ve yazılımların birbiri ile iletişim kurmasına olanak sağlar. JAUS sayesinde bu gibi heterojen bilgisayar sistemleri birlikte çalışabilmektedir. Ek olarak insansız sistemleri kendi bünyemize dâhil etme ve çalıştırma operasyonları daha basitçe ve hızlıca yapılabilmektedir. Ayrıca Jaus TCP/IP kullandığı için farklı bir veri transfer kanalı ya da donanımına ihtiyaç duymamaktadır. JAUS özellikle askeri anlamda yapılan görevlerde aynı anda birden fazla insansız sistemi yönetmeyi de sağlamaktadır. Sistemimizi mantıksal olarak JAUS yazılımına tanımlayarak sistemin JAUS topolojisi oluşturulur. JAUS topolojisini ve yazılımını sistemimize uygularken kullanılan bazı kavramlar ve seviyeler vardır. Bu kavram ve açıklamalar detaylı olarak JAUS Referans Modelinde anlatılmıştır. JAUS topolojisinde bulunan sistem, alt sistem, nod ve bileşenlerin JAUS haberleşme uyumluluklarına göre üç seviyede sınıflandırılırlar. JAUS'un günümüzde takip ettiği referanslara göre çeşitli sürümleri bulunmaktadır. Bu sürümler iki ana gruba ayrılmıştır. JAUS Referance Architecture (RA) ya da diğer bir deyişle referans modeli, JAUS'un ilk çıkan orijinal halidir. JAUS'un bu projesinin geliştirilmesi sürüm 3.3'den sonra durduruldu. Temeli bileşenler arası haberleşme mimarisine dayanmaktadır. SEA-JAUS JAUS'un standartlarının günümüzde kullanılan son güncel halidir. Servis tabanlı bir mimariye sahiptir. JAUS++, JAUS protokolünün nesneye dayalı C++ uyarlaması olan bir açık kaynak kod kütüphanesi projesidir. JAUS++ projesi, mühendisler ve geliştiricilerin

kendi robotik projelerine JAUS protokolünü kolayca dâhil edebilmelerini amaçlamıştır. İlk sürümleri JAUS RA (Referans Modelini) temel almaktadır. İkinci sürümden sonra günümüzde de geçerli JAUS sürümü olan SAE JAUS standartlarına göre geliştirilmiştir.

Bu tezde JAUS mimarisinin bileşenlerinin incelenmesi ve insansız sistemimizi JAUS uyumlu hale getirme adımları konu alınmıştır. JAUS uyumlu hale getirmek için literatürde bulunan kod kütüphaneleri incelenip test edilmiştir. Bu kod kütüphaneleri içinde güncel ve başarılı olanı seçilmiştir. JAUS++ kütüphanesi kullanılarak operatör kontrol ünitesi ve Radxa Rock ARM tabanlı bilgisayar kullanılarak JAUS mesaj alışverişi gerçekleştirilmiştir. JAUS sayesinde insansız sistemler arasında sağlanan iletişim kullanarak birçok görev daha verimli hale getirilebilir. JAUS'un yazılımsal tasarımından ötürü oluşturulan JAUS kodları yeniden kullanılabilirdir. Bu sayede yazılım maliyetleri ve insansız sistem üzerinde çalışan program en uygun düzeyde koşturulabilir. Üretilen robotların uluslararası ölçekte daha iyi yer edinebilmesi için JAUS uyumlu olması her zaman bir artı olacaktır. Bu alışverişin diğer veri alışverişlerine göre avantajları gözlemlenmiştir.

DATA TRANSMISSION WITH JOINT ARCHITECTURE FOR UNMANNED VEHICLE

SUMMARY

Unmanned systems offers cost effective and high quality work to perform in dangerous or heavy tasks. Also they can perform repetitive tasks with high ratio of success. These benefits of robots can be summarized as mostly in productivity, safety, and in saving time and money. As a result robotic industry produces more unmanned systems to market. Many on these robotic systems are unable to operate and interact with each other. Communication rules are the most needed component for data to travel one source to destination. Also it's recommended that all the computer systems communicate through the same data structure. So that clients could start communication with each other by using the same software. In computer world the most common communication protocol is TCP/IP (Transmit Control Protocol / Internet Protocol). In addition today unmanned systems can use that TCP/IP protocol to communicate between operator and the control unit. Using this TCP/IP communication method developers and researchers create different types of data structures. Between these different unmanned systems the common and simple communication cannot be established. JAUS (Joint Architecture for Unmanned Systems) can solve these limitations and provide all unmanned systems in a domain to communicate with each other and become interoperability. The maintain interoperability to components of a system, predefined rules and standard addressing must be used. JAUS provides a communication framework to reduce life-cycle costs. Reusable components are defined as object orientated method. This method reduces the maintenance costs and allow to develop and improve the unmanned robotic system. As an object oriented written program, it is easy to maintain and modify existing code and improve the software. Also this makes possible to insert the newer technologic hardware into JAUS system. JAUS system designed to be modular and scalable. To integrate our system to support JAUS there are number of terms to understand. These terms are system, subsystem, node, component and instance. A system is a logical grouping of unmanned systems in a domain. In a system all unmanned systems and operator control units that share common applications. A subsystem could able to perform one unmanned task. A subsystem can also provide communication and control abilities. Unmanned ground vehicles, unmanned aerial vehicles that support communication, control abilities could be given as example. A node could support processing capability. In an unmanned system, main processor or the image processing processor, sensor data processor could be given example and as separate nodes of an unmanned system. A component is placed under a node, it provides commands and control. An instance is a piece of different information that is served under a component, a heat value or speed value can named as an instance. Depending on the how JAUS is integrated to our unmanned system there are three levels of integration. In level I integration JAUS compatibility is at subsystem level. Only the communication between two subsystems in a domain environment is done

with JAUS protocol. In addition in level II integration the communication between node manager and subsystem itself is accomplish with JAUS message structure. Lastly in level II JAUS messages are applied at component level. Components are defined as four sectors as IP addressed, beginning with subsystem-ID, node-ID, component-ID and instance-ID. This form is called JAUS address. In every JAUS subsystem there is only one node that is responsible for the whole communication of the unmanned system. This node is called node manager. In addition node manager routes the JAUS communication traffic through the unmanned system itself. Also the components are communicates with each other via the node manager. There are seven JAUS message class. These are Command, Query, Inform, Event Setup, Event Notification, Node Management, and Experimental. These designs are made to easily build and organize the messages. When operator control unit is started, it acts as a subsystem. Operator control unit discovers the other subsystems in the JAUS domain. When connected to subsystem, subsystem send a heartbeat to the operator control unit so that the operator can acknowledge the presence of the unmanned system. It is also not so accepted by some authorities that one common operator control unit to control all types of robots. Because some companies also wants to sell their own operator control unit. But it is a pressure from the governments to increase the usage of JAUS in unmanned systems.

In this Thesis JAUS structure and components are examined. To make JAUS compatible unmanned systems there are some open source projects. These projects are examined and the most optimal and updated version is selected for test section. Using JAUS++ code library is used as experimental purposes to create an Operator Control Unit and an Unmanned System which is Radxa Rock hardware. Radxa Rock is a single board computer by Radxa. It has a quad core processor, it can run android or some Linux distributions. It also got 80 pin headers to connect other sensors or use the GPIO. JAUS++ is created by The ACTIVE Laboratory based on a C++ implementation of JAUS for use in unmanned vehicles. This paper provides an introduction to the library for new developers, and give software architecture overview. After reading this document developers should know how to start using the JAUS++ library within an unmanned systems application. JAUS++ has three main libraries. Core library contains base classes for messages and services. Mobility library contains data structures for Global pose sensor and global waypoint driver. JAUS++ can be run at Linux and windows operating system platforms. The Message class is used to create a JAUS message structure. Current version of JAUS changed its architecture to use a service based framework. In JAUS++ library, Service class is used for message creation and manipulation. JAUS++ divides systems and subsystems into software components. JAUS++ components are designed for especially software reuse. In this research, the same software is used as in the Operator Control Unit software and the unmanned system itself with a little modification. However when we use specific components as Global Pose sensor, hardware implementation could be need to be done. In general because of the Object Oriented design, JAUS++ Component service highly reusable. To maintaining interoperability between the robotic systems and the operator control units is one of the key goals of JAUS++. Because of that the subsystems in JAUS uses same massage set. For instance an operator control unit can control all types of subsystems without any modification. JAUS has compatible with serial, and TCP networks. In general JAUS does not specify the communication method, any of the methods above can be used. JAUS is a communication standard that initiated by the United States Department of Defense.

It is an open and scalable and service based architecture. JAUS also designed to be vehicle platform and hardware and technology independent.

JAUS specifies a hierarchical schema. These members of a schema can be explained as follows. A System contains a sum of Subsystems. A Subsystem is a robotic systems that carries unmanned capabilities such as a robot or an operator control unit. Subsystems consists a sum of Nodes which individual computers can be given as example. On each node one or more Components could be take place. A Component performs a specific function or provides a specific service in our unmanned system. For example an actuator or sensor is explained as components in a JAUS system. To satisfy a complete JAUS compatible unmanned system, The Joint Architecture for Unmanned Systems consists some dedicated elements that act as a whole. The lowest level of abstraction within JAUS is the component. Going up the chain of complexity, a JAUS node consists of multiple components, a subsystem consists of one or more nodes, and a JAUS system consists of one or more subsystems. In nodes there could be more than one instance. This feature supports component redundancy. JAUS specifies the message format so that all robotic developers can improve their codes. JAUS provides a basis for logical interoperability, it reduces cost of support and development. A service is a gate to access a ability of unmanned system.

After using JAUS++ advantages are observed as code reusability, modularity, and interoperability. Even if the technology improves, for instance the connection speed or the processor speed grows, the JAUS software could still be effective as it is, because we define our system into JAUS as software parts. In addition, the JAUS code parts could be reused without any modification as the benefits of object oriented principle of inheritance.

1. GİRİŞ

Haberleşme, günümüzde kullandığımız bilgisayar sistemlerinde temel bir bileşendir. Dünya üzerinde geniş ölçekli bir haberleşme ağı mevcuttur. Şu da bir gerçektir ki haberleşme sistemleri bu kadar gelişmeden önce bilginin dağıtımı ve geliştirilmesi daha verimsiz olmaktaydı. Verinin bir ağ üzerinde herhangi bir kaynaktan bir hedefe yol alabilmesi için gerekli olan en önemli ölçüt ağı oluşturan tüm elemanların aynı dili konuşması ya da aynı kurallar ailesine uyarak haberleşmeleridir. Bu kurallar ailesine protokol denir. Ayrıca iletişim bir etki alanı içerisinde bulunan insansız sistemlerinde önemli bir role sahiptir. İnsansız sistemler arası iletişim yapılabilir hale getirildiğinde insansız sistemler arasındaki birlikte çalışma ve dolayısıyla sistem performansı arttırılabilir [1].En yaygın olarak kullanılan kurallar bütünü ve protokol TCP/IP (Transmit Control Protocol / Internet Protocol)'dir. Bu protokolleri kullanarak iki bilgisayar arasında veri gönderme ve alma işlemleri yapılabilir. Protokol aynı olsa bile gönderip aldığımız anlamlı veriler gene bir kural izlemelidir herkes tarafından anlaşılabilmesi için gereklidir. Örneğin bir sunucudan web sayfası istediğimizde karşımıza anlamlı bir bilgi çıkması için aradaki iletişimin HTTP protokol kurallarına uymalıdır. Aynı şekilde bir dosya paylaşımına bağlanabilmek için SMB protokolünde verilerin sunulması gereklidir. Buradaki ortak protokol kullanılmasındaki amaç diğer istemcilerin tek bir yazılım kullanarak birbirinden farklı siteler açabilmesi, tek yazılımla birden çok dosya paylaşımına bağlanabilmeleridir. Yukarıda verdiğim örneklere benzer olarak robot uygulamalarında da kullanılan bilgisayar sistemleri TCP/IP protokolünü kullanarak robot ile Kontrol Ünitesi ve ya operatör arasında veri iletişimini sağlayabilirler. Bu haberleşmedeki problemler haberleşme veri yapısının her robot geliştirici için farklı olması, farklı kaynaklardan çıkan Robot veya bilgisayar için tasarlayan kişiye hizmet vermesi, uyumsuz olma, belli bir kurala göre yazılmadığı için eksik işlevler içermesi birbiriyle iletişim kuramama olarak sıralanabilir.

JAUS (Joint Architecture for Unmanned Systems) referans modeli, bu eksiklikleri gidermekte ve robot sistemlerinde esneklik, erişilebilirlik ve güvenilirlik

sağlamaktadır. JAUS dünya üzerinde birçok farklı üreticinin ve firmanın ürettiği değişik tipteki insansız araçların birlikte çalışabilir (interoperability) olmasını garantileyen bir dizi standartları belirleyerek üreticilere ve geliştiricilere bu desteği vermiştir [2].

1.1 Tezin Kapsamı

Bu tezde JAUS mimarisinin bileşenlerinin incelenmesi ve insansız sistemimizi JAUS uyumlu hale getirme adımları konu alınmıştır. JAUS uyumlu hale getirmek için literatürde bulunan kod kütüphaneleri incelenip test edilmiştir. Bu kod kütüphaneleri içinde güncel ve başarılı olanı seçilmiştir. JAUS++ kütüphanesi kullanılarak operatör kontrol ünitesi ve Radxa Rock ARM tabanlı bilgisayar kullanılarak JAUS mesaj alışverişi gerçekleştirilmiştir. Bu alışverişin diğer veri alışverişlerine göre avantajları gözlemlenmiştir.

2. JAUS

2.1 Amaç

Joint Architecture for Unmanned Systems (JAUS) 1998 yılında Amerikan Savunma Bakanlığı tarafından başlatılan bir projedir. Projenin amacı insansız sistemlerin birbirleriyle ortak iletişim kuralları, kısacası ortak bir dil kullanarak veri alışverişini sağlamaktır. Bu mekanizmanın sağladığı yeniden kullanılabilir komutlar ve mesajlarla birlikte, sistemin daha verimli ve hızlı çalışması amaçlanmıştır. Ayrıca insansız sistemlere, ortak bir noktadan yönetebilme, aynı anda birden çok sistemi yönetebilme, modüler ve yeniden ölçeklenebilir olma, değişen teknolojiye ve donanıma uyumlu olma yeteneklerini eklemiştir. JAUS'un sağladığı katmanlı yapıda, birbiriyle paralel şekilde geliştirme mümkün olduğundan, yazılımların daha çabuk gelişebilmesine olanak sağlamıştır.

Birden fazla üretici ve araştırmacı birbirinden farklı platformlarda ve farklı fiziksel donanımlarda kendi insansız sistemlerini geliştirmektedir. JAUS farklı üreticilerin geliştireceği insansız sistem parçalarını ve bileşenlerini standartlaştırır, değişik tipte donanım ve yazılımların birbiri ile iletişim kurmasına olanak sağlar. JAUS sayesinde bu gibi heterojen bilgisayar sistemleri birlikte çalışabilmektedir. Ek olarak insansız sistemleri kendi bünyemize dâhil etme ve çalıştırma operasyonları daha basitçe ve hızlıca yapılabilmektedir. Ayrıca JAUS TCP/IP kullandığı için farklı bir veri transfer kanalı ya da donanımına ihtiyaç duymamaktadır. JAUS özellikle askeri anlamda yapılan görevlerde aynı anda birden fazla insansız sistemi yönetmeyi de sağlamaktadır.

2.2 JAUS Terimleri ve JAUS Topolojisi

Sistemimizi mantıksal olarak JAUS yazılımına tanımlayarak sistemin JAUS topolojisi oluşturulur. JAUS topolojisini ve yazılımını sistemimize uygularken kullanılan bazı kavramlar ve seviyeler vardır. Bu kavram ve açıklamalar detaylı olarak JAUS Referans Modelinde anlatılmıştır [3]. JAUS Reference Architecture (RA) ya da türkçe karşılığı

olarak referans modeli, JAUS'un ilk çıkan orijinal halidir. Günümüzde geliştirilerek ve adı değiştirilerek SAE JAUS referansı kullanılmaktadır. Temel olarak SAE JAUS servis tabanlı bir mimariyi kullanmaktadır.

2.2.1 Sistem

İnsansız alt sistemlerin mantıksal olarak gruplanmış halidir. Belirli bir ortak görevi olan ve birbirlerine yerel iletişim olarak direk bağlı üyelerden oluşur. Örnek bir sistemin içerisinde JAUS uyumlu Operatör Kontrol Ünitesi (OCU), haberleşmenin sağlanması için gerekli ağ cihazları ve bir veya birden fazla insansız kara aracı bulunabilir.

2.2.2 Altsistem

İnsansız bir işlev yapabilme kapasitesi olan bilgisayar sistemlerine denir. Ek olarak haberleşme, komut işleme özelliğinin de bulunması gerekir. Örnek olarak;

- İnsansız kara araçları
- İnsansız hava araçları
- İnsansız denizaltı araçları
- Operatör Kontrol Ünitesi (OCU)

verilebilir.

2.2.3 Nod

Alt sistem içerisinde İşlem kapasitesine sahip bileşenlere verilen isimdir. Bu işlemci kendine bağlı alt üyelerle haberleşebilmelidir. Nod tam anlamıyla bir servis sağlayabilmelidir. Örnek olarak insansız sistemdeki ana kontrol bilgisayarı, görüntü işlemcisi, hareket kontrol bilgisayarı verilebilir.

2.2.4 Bileşen

İşlemciye bağlı ve işlemciyle veri alışverişinde bulunabilecek sistem parçalarıdır. JAUS hiyerarşisinde en altta bulunan kısımdır.

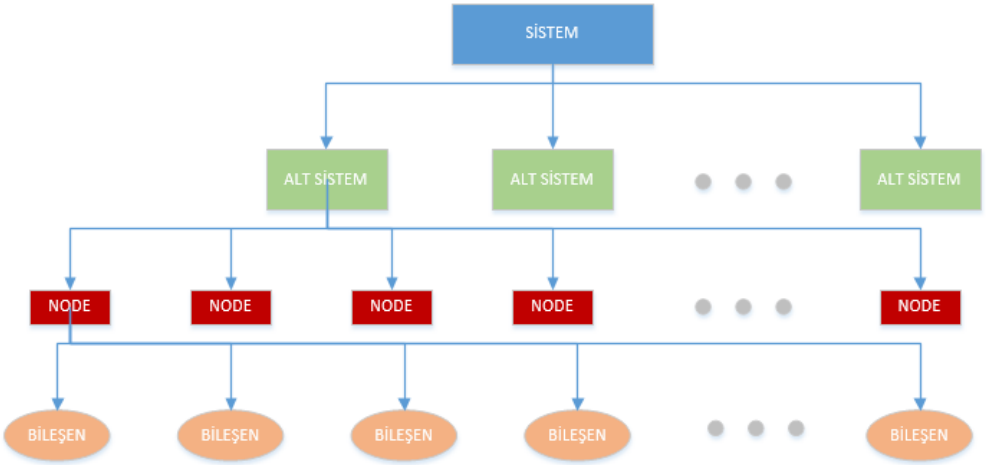
2.2.5 Durum

Bileşene ait yedeklilik veya birden fazla veriyi belirlemek için kullanılırlar. Örneğin bir bileşenin altında birden fazla algılayıcı bulundurulması verilebilir.

2.2.6 Mesaj

JAUS mesaj başlığı ve ilgili verisinden oluşan kavramdır. JAUS haberleşmesinde kullanılır.

Bu terimlere göre sistemimizdeki mantıksal konumlandırılması Şekil 2.1'deki gibidir.



Şekil 2.1: JAUS Sistem Topolojisi

2.3 JAUS Sistem Uyumluluk Seviyeleri

JAUS topolojisinde bulunan sistem, alt sistem, nod ve bileşenlerin JAUS haberleşme uyumluluklarına göre üç seviyede sınıflandırılırlar [4].

2.3.1 Seviye I uyumluluk

JAUS yazılımının alt sistem seviyesinde insansız sistemimize uygulanmış halidir. Alt sistemde bulunan haberleşme kanalıyla JAUS iletişim kurallarının işletilmesidir. Sistemde geri kalan diğer iletişim trafiği JAUS uyumlu olmayan haberleşme biçimleri veya sistemi tasarlayanlar tarafından belirlenen kuralları içerebilirler. Uygulaması en temel uyumluluk seviyesidir.

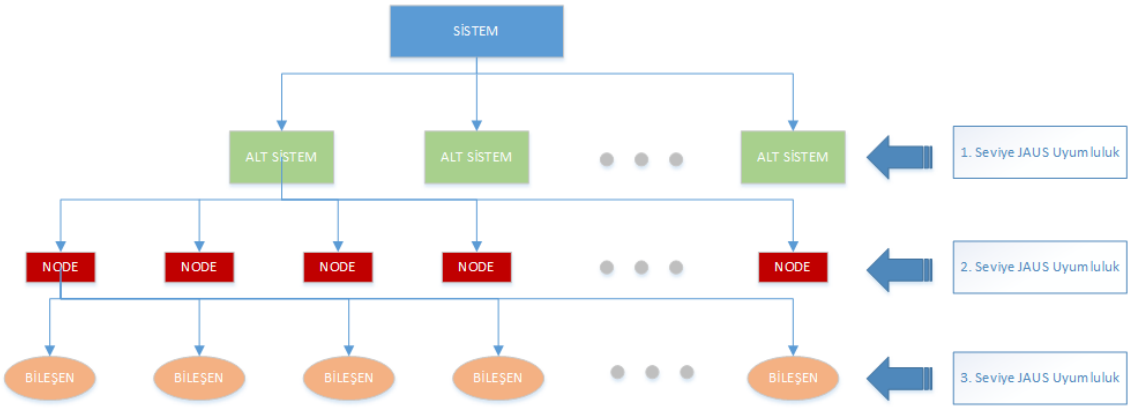
2.3.2 Seviye II uyumluluk

Seviye 2 uyumluluk modunda nod seviyesine kadar JAUS mesajlarının iletildiği uyumluluk seviyesidir.

2.3.3 Seviye III uyumluluk

Seviye 3 uyumluluk modu JAUS mesajlarının bileşen seviyesinde iletildiği uyumluluk modudur. Sistemimizdeki bileşenlerin nod yöneticileriyle JAUS protokolü üzerinden haberleşmesi gerekmektedir. JAUS göreceli olarak yeni bir yazılım olduğu için henüz üçüncü seviye bileşenli sistemler bulunmamaktadır.

Bu seviyelerin göre sistemimizdeki mantıksal konumlandırılması Şekil 2.2'deki gibidir.



Şekil 2.2: JAUS Uyumluluk Seviyeleri

2.4 JAUS Mesaj Yapısı

JAUS uyumlu veri alışverişi için TCP/IP iletişim kurallarıyla haberleşmeyi gerçekleştirirken belirli kurallar ve sabit başlık bilgileri içermelidir [5]. Mesaj yapısı Çizelge 2.1'de belirtilmiştir.

Çizelge 2.1: JAUS mesaj başlık yapısı.

Bilgi NO:	Tanımı	Tip	Boyut(byte)
1	Mesaj Özelliği	Unsigned Short	2
2	Komut Kodu	Unsigned Short	2
3	Hedef Durum ID	Byte	1
4	Hedef Bileşen ID	Byte	1
5	Hedef Nod ID	Byte	1
6	Hedef Altsistem ID	Byte	1
7	Kaynak Durum ID	Byte	1
8	Kaynak Bileşen ID	Byte	1
9	Kaynak Nod ID	Byte	1
10	Kaynak Altsistem ID	Byte	1
11	Veri Kontrol	Byte	2
12	Dizi Numarası	Byte	2
-	Toplam Boyut	-	16

2.4.1 Mesaj sınıfları

JAUS mimarisinde yedi adet mesaj sınıfı tanımlanmıştır [6]. Mesaj tiplerini organize ederek mesaj tiplerine göre filtreleme yapma ve daha detaylı araştırma yapabilmek için tasarlanmıştır. Çizelge 2.2'de mesaj sınıfları tanıtılmıştır.

Çizelge 2.2: JAUS mesaj sınıfları.

Mesaj Sınıfı	Adres Aralığı
Komut (command)	0000h – 1FFFh
Sorgu (query)	2000h – 3FFFh
Bilgilendirme (Inform) (command)	4000h – 5FFFh
Olay Kurulum (Event Setup) (command)	6000h – 7FFFh
Olay Bilgilendirme (Event Notiication)	8000h – 9FFFh
Nod Yönetim (Node Management)	A000h – BFFFh
Deneysel (Experimental)	C000h – CFFFh

2.5 JAUS Sürümleri

JAUS'un günümüzde takip ettiği referanslara göre çeşitli sürümleri bulunmaktadır. Bu sürümler iki ana gruba ayrılmıştır.

2.5.1 JAUS RA

JAUS Reference Architecture (RA) yada diğer bir deyişle referans modeli, JAUS'un ilk çıkan orijinal halidir. JAUS'un bu projesinin geliştirilmesi sürüm 3.3'den sonra durduruldu. Temeli bileşenler arası haberleşme mimarisine dayanmaktadır [7].

2.5.2 SAE JAUS

SAE-JAUS JAUS'un standartlarının günümüzde kulanılan son güncel halidir. Servis tabanlı bir mimariye sahiptir. Geliştirilen projenin şu an kullanılan kural seti ve belgeleri aşağıdaki gibidir.

AS5669 – JAUS Transport Standard: Haberleşme katmanında paket yapısını ve adreslemesini açıklar [8].

AS5710 – JAUS Core Service Set: Temel hizmet bileşenlerini ile iligli açıklamaları içerir [9].

AS6009 – JAUS Mobility Service Set: Bileşenlerin haberleşmesinde kulanılan kuralları açıklar [10].

2.6 JAUS Açık Kaynak Kütüphaneleri

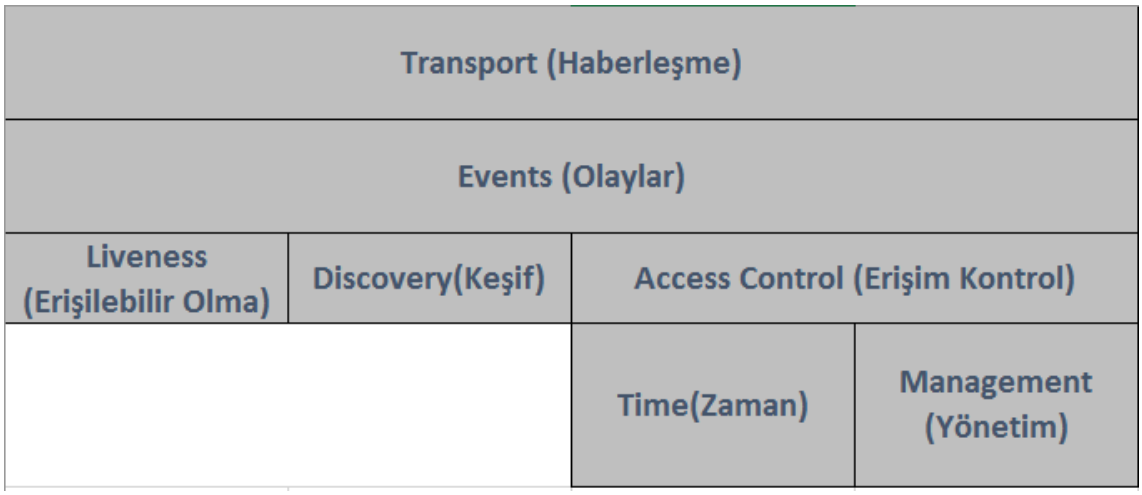
JAUS uygulaması yaparken kod kütüphanemizi kendimizin oluşturabileceğimiz gibi literatürde bulunan açık kaynak JAUS kütüphaneleri de kullanabiliriz. Bu kod kütüphaneleri uyguladığı JAUS sürümlerine göre çeşitlilik göstermektedir. Bu projede deneyler JAUS++ kod kütüphanesi kullanılarak yapılmıştır. Kütüphanenin kullanımı bir sonraki bölümde anlatılacaktır. Aşağıdaki Çizelge 2.3'te en yaygın kullanılan JAUS açık kaynak kod kütüphaneleri listelenmiştir.

Çizelge 2.3: JAUS Açık Kaynak Kod Kütüphaneleri.

Projenin Adı	Uyguladığı JAUS Sürümü
OpenJAUS	JAUS RA 3.3
JAUS Tool Set	SAE JAUS
Junior Middleware	SAE JAUS
JAUS++ v1.x	JAUS RA 3.3
JAUS++ v2.x	SAE JAUS
RI-JAUS	JAUS RA 3.3

2.7 Temel JAUS Servis Yapısı

SAE-JAUS servis tabanlı bir yapıdadır. Bu temel servisler sayesinde insansız sisteminizin mantıksal yapısını çıkarmak ve yönetmek kolaylaşmıştır. Bu servislerin yapısı Şekil 2.3'te gösterilmiştir.



Şekil 2.3: JAUS Servis Yapısı

2.7.1 Transport (haberleşme) servisi

TCP, UDP ve Seri JAUS portlarını ve mesaj başlık yapısını, mesaj tipini, hedef ve kaynak adreslerini tanımlar. JAUS haberleşmesi için varsayılan haberleşme portu UDP 3794'dir.

2.7.2 Events (olaylar) servisi

Belirli bir periyodda ve ya değişim olduğu zaman bilgilendirme yapılmasını sağlayan hizmettir. Örneğin robot hareket ettiğinde operatöre bilgilendirme gitmesini events (olaylar) servisi sağlar.

2.7.3 Liveness (erişilebilirlik) Servisi

Bu servis operatöre ve diğer kullanıcılara insansız sistemin erişilebilir ya da erişilemez olduğu bilgisini sunar.

2.7.4 Time (zaman) servisi

Time(zaman) servisi ile insansız sistemdeki zaman değişkenini öğrenmemizi sağlar.

2.7.5 Discovery (keşif) servisi

Bu servis etki alanındaki insansız sistemleri keşfetmeye yarar. Keşif yapıldıktan sonra insansız sistem ile ilgili yapılandırma ve uygun olan servisleri sunar.

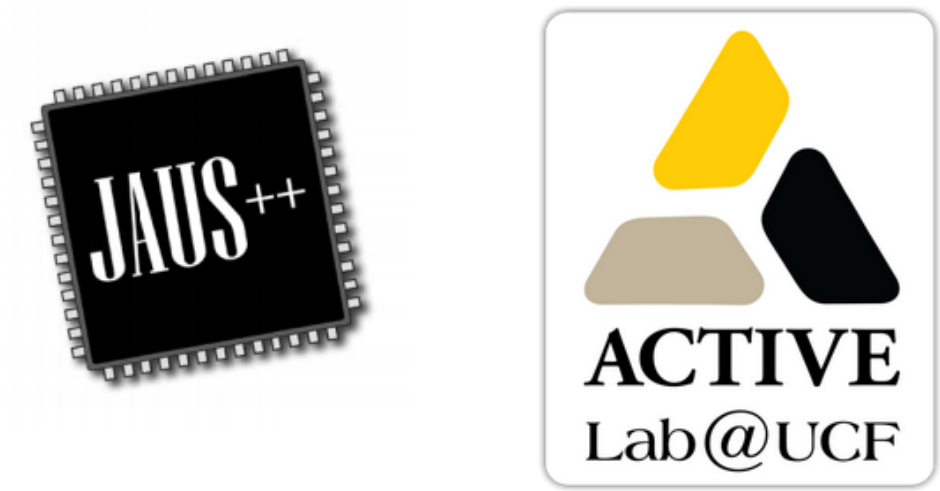
2.7.6 Management (yönetim) servisi

Management (yönetim) servisi ile inansız sistem üzerinde temel işlemleri gerçekleştirmemizi sağlar. Hazır olma, beklemede olma, acil kapanma, yeniden başlatma işlemleri yapılır.

3. JAUS++ Kütüphanesinin Kurulması

3.1 JAUS++ Nedir?

JAUS++, University of Central Florida ACTIVE-IST Laboratuvarı tarafından geliştirilen JAUS protokolünün nesneye dayalı C++ uyarlaması olan bir açık kaynak kod kütüphanesi projesidir [11]. Bu proje 2010 yılında başlatılmış olup günümüze kadar geliştirilmeyi sürdürülmüştür. JAUS++ projesi, mühendisler ve geliştiricilerin kendi robotik projelerine JAUS protokolünü kolayca dâhil edebilmelerini amaçlamıştır. İlk sürümleri JAUS RA (Referans Modelini) temel almaktadır. İkinci sürümden sonra günümüzde de geçerli JAUS sürümü olan SAE JAUS AS5669, AS5710, AS6009 standartlarına göre geliştirilmiştir. Bu açık kaynak kod kütüphanesi 2010 yılındaki IGVC (Intelligent Ground Vehicle Competition) yarışmasında hem başarı ile test edilmiş hem de yarışmada ödül kazanmıştır [12].

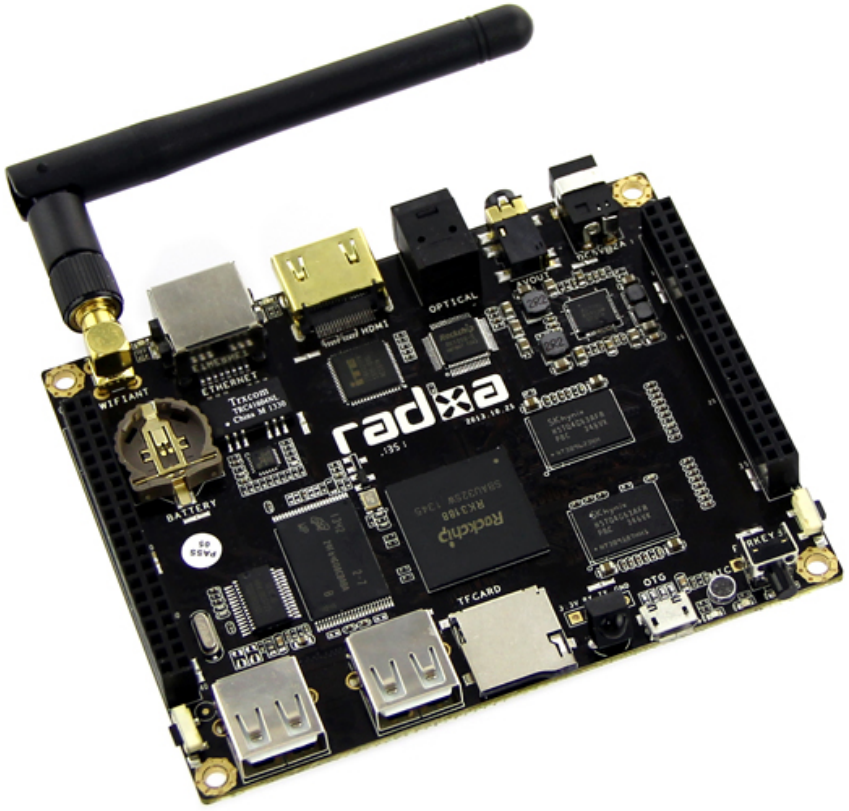


Şekil 3.1: JAUS++ ve ACTIVE-Lab

JAUS++ hem Windows hem de Linux işletim sistemi platformlarında desteklenmektedir. Linux işletim sistemleri için bu projede 5.141203 sürümü, Windows işletim sistemi için de 2.110519 sürümü kullanılmıştır.

3.2 Deney Ortamı

Test ortamı olarak bir adet operatör kontrol ünitesi görevinde Windows 8 bilgisayar, insansız sistem görevinde adet Radxa Rock ARM tabanlı geliştirme kartı olmak üzere 2 adet bilgisayar sistemi kullanılmıştır. Radxa Rock donanımı şekil 3.2'de gösterilmiştir.



Şekil 3.2: Radxa Rock donanımı

3.3 JAUS ++ Linux İşletim Sistemine kurulması

3.3.1 Linux işletim sisteminin hazırlanması

JAUS++ kaynak kodlarını ve programını kurmadan önce Linux dağıtımı üzerinde birtakım ayarlamalar yapılması gerekmektedir. Aşağıdaki bölümlerde Radxa Rock donanımı üzerine kurulan Ubuntu 14.01 işletim sisteminde yapılan ayarlar anlatılmıştır.

3.3.1.1 IP adresinin verilmesi ve DNS ayarları

Bilgisayara tek IP üzerinde erişebilmek için statik IP yapılandırması yapılmalıdır. Aşağıdaki komutlar bu işlemi yapar:

```
$ sudo cat /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:

source-directory /etc/network/interfaces.d

auto eth0
iface eth0 inet static
address 160.75.5.201
netmask 255.255.255.0
network 160.75.5.0
broadcast 160.75.5.255
gateway 160.75.5.254
hwaddress ether de:ad:be:ef:52:01
```

3.3.1.2 Güvenlik duvarı ayarları

Güvenlik duvarı yapılandırması için öncelikle iptables-services isimli iptables-services paketi sisteme yüklenmelidir.

Aşağıdaki komutlar bu işlemi yapar:

```
$ sudo apt-get install iptables
Reading package lists... Done
Building dependency tree
Reading state information... Done
iptables is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 6 not upgraded.
```

Güvenlik duvarı üzerinde aşağıdaki temel yapılandırma bulunmalıdır. Projenin ihtiyaçlarına göre daha sonradan güvenlik duvarı en sondaki DROP kuralının üstüne izin satırları girilerek genişletilmelidir. Güvenlik açısından Radxa Rock donanımız için SSH ve JAUS'un haberleştiği UDP 3794 portlarına izin verip diğer portları kapatmalıyız.

Aşağıdaki komutlar bu işlemi yapar:

```
$ iptables -F
$ iptables -A INPUT -i lo -j ACCEPT
$ iptables -A INPUT -m state --state ESTABLISHED,\
RELATED -j ACCEPT
```

```
$ iptables –A INPUT –s bcksrv1.cc.itu.edu.tr –j ACCEPT
$ iptables –A INPUT –p tcp –s 160.75.5.98 ––dport 22 –j ACCEPT
$ iptables –A INPUT –p udp –s 160.75.5.98 ––dport 3794 –j ACCEPT
$ iptables –A INPUT –j DROP
```

3.3.2 JAUS++ kurulması

JAUS++ için gerekli kütüphaneler kurulur. Kumanda ve kontrol arabirimleri için gerekli olan wxWidgets paketi, JAUS++ kütüphanesini derlemek için kullanılan cmake paketi ve boost c++ kütüphanesi kurulur.

Aşağıdaki komutlar bu işlemi yapar:

```
$ apt-get install libpng-dev libX11-dev libxtst-dev
$ sudo apt-get install python-wxgtk2.8 python-wxglade cmake
```

JAUS++ projesinin resmi internet sitesinden projenin dosyaları indirilir.

Aşağıdaki komutlar bu işlemi yapar:

```
$ wget http://downloads.sourceforge.net/project/active-ist
/JAUS%2B%2B/5.141203/JAUS%2B%2B-5.141203-src.7z
```

İndirilen sıkıştırılmış dosya açılır.

Aşağıdaki komutlar bu işlemi yapar:

```
$ 7za x /root/JAUS++-5.141203-src.7z
```

CMakeList.txt dosyası /JAUS++-5.141203-src/build/cmake dizinine kopyalanır. /JAUS++-5.141203-src/build/cmake dizin yoluna geçilir ve cmake . komutu çalıştırılır.

Aşağıdaki komutlar bu işlemi yapar:

```
$ cp CMakeLists.txt build/cmake/
```

JAUS++-5.141203-src/build/cmake dizin yoluna geçilir ve derleme işlemi için aşağıdaki üç komut çalıştırılır. Aşağıdaki komutlar bu işlemi yapar:

```
$ cmake .
$ make
$ make install
```

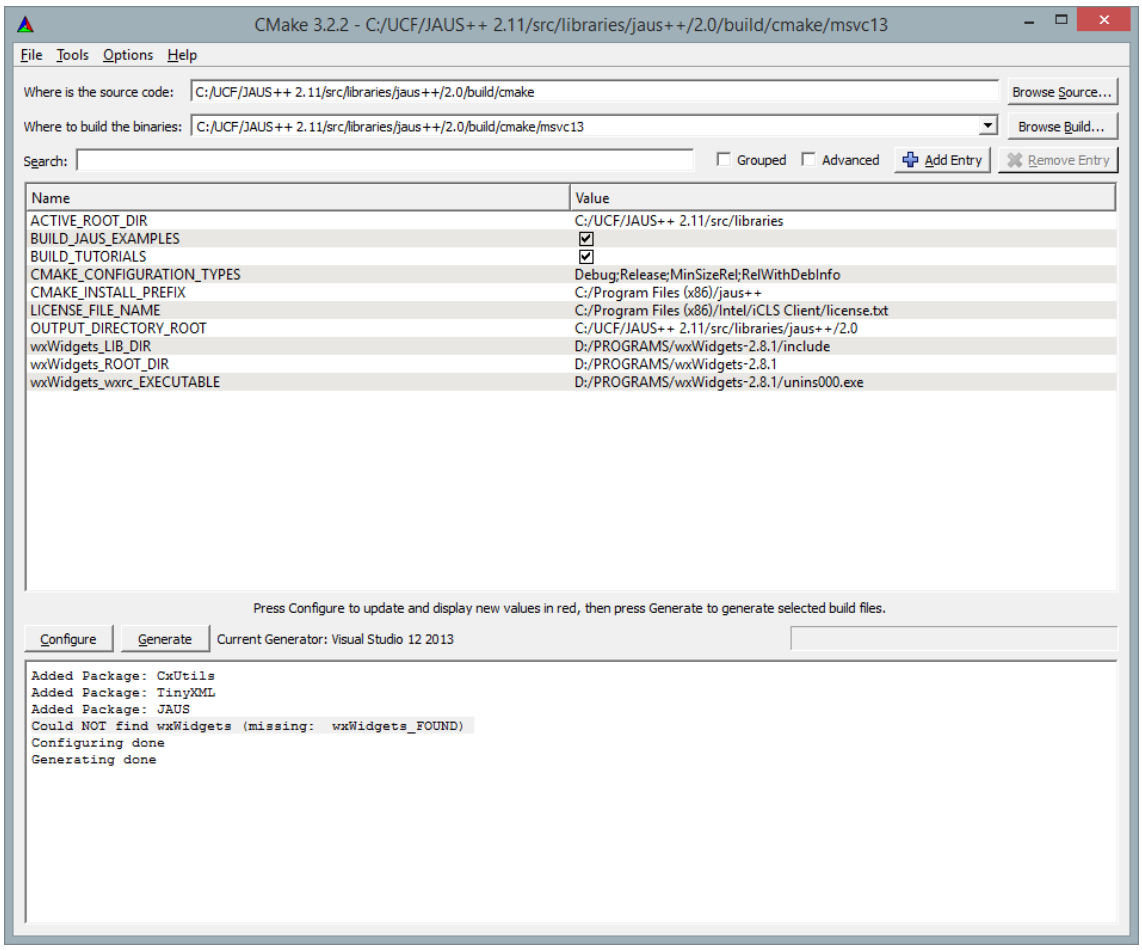
Derlenen kütüphane dosyalarının sistem tarafına dahil edilebilmesi için ld.so.conf dosyasına /usr/local/lib/active satırı eklenir. Değişikliğin geçerli olabilmesi için sudo ldconfig komutu çalıştırılır.

Aşağıdaki komutlar bu işlemi yapar:

```
$ echo /usr/local/lib/active > /etc/ld.so.conf
$ sudo ldconfig
```

3.4 JAUS ++ Windows İşletim Sistemine kurulması

JAUS++ Windows sürümü 2.110519 kullanıldı. Bu programı kurduktan sonra "C:/UCF/JAUS++2.11" altında JAUS++ kütüphane dosyaları oluştu. Windows işletim sistemi c++ kod derlemede Visual Studio 2013 kullanılmıştır. JAUS++ kütüphanesini Visual Studio 2013 programına entegre edebilmek için CMAKE programı kullanılmalıdır. CMAKE programı da kurulduktan sonra çalıştırılır. Gelen ekranda "Where is the source code" kısımına "C:/UCF/JAUS++ 2.11/src/libraries/-jaus++/2.0/build/cmake" yazılır. Daha sonra "Where to build the binaries:" kısmına da Visua Studio dosyalarının oluşturulacağı dizin girilir. Bu kutuya "C:/UCF/JAUS++ 2.11/src/libraries/jaus++/2.0/build/cmake/msvc13" yazılır. Şekil 3.3'de programın yapılandırılması gösterilmiştir.

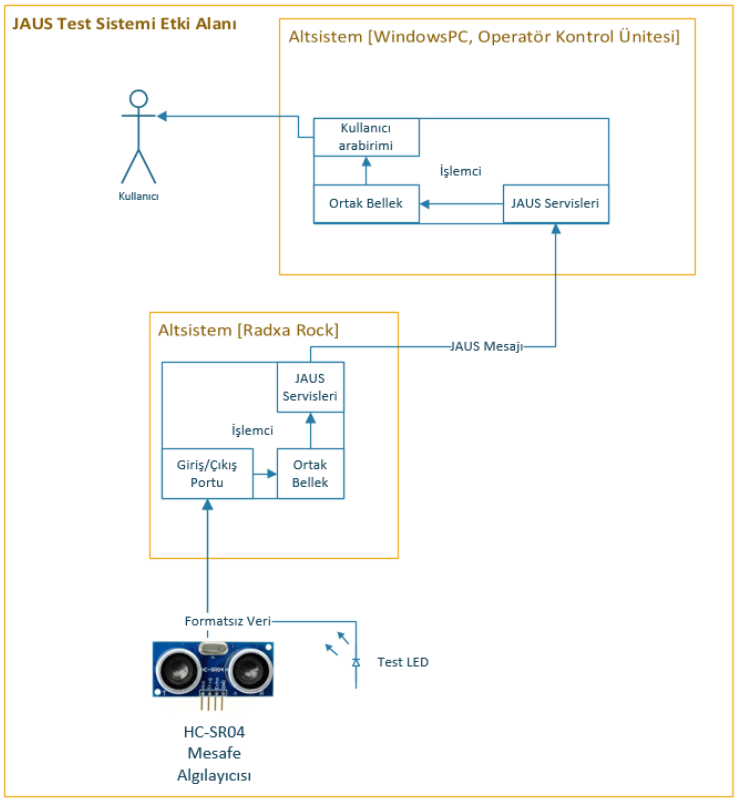


Şekil 3.3: CMAKE yapılandırma ekranı

Bu ayarlarla "Configure" ve "Generate" tuşlarına basılarak Visual Studio için gerekli dosyaların oluşması sağlanır. Ardından Visual Studio'da JAUS++ kütüphanesini kullanabilecek ayarlar tamamlanmış olur.

3.5 JAUS Veri İletişiminin Test Edilmesi

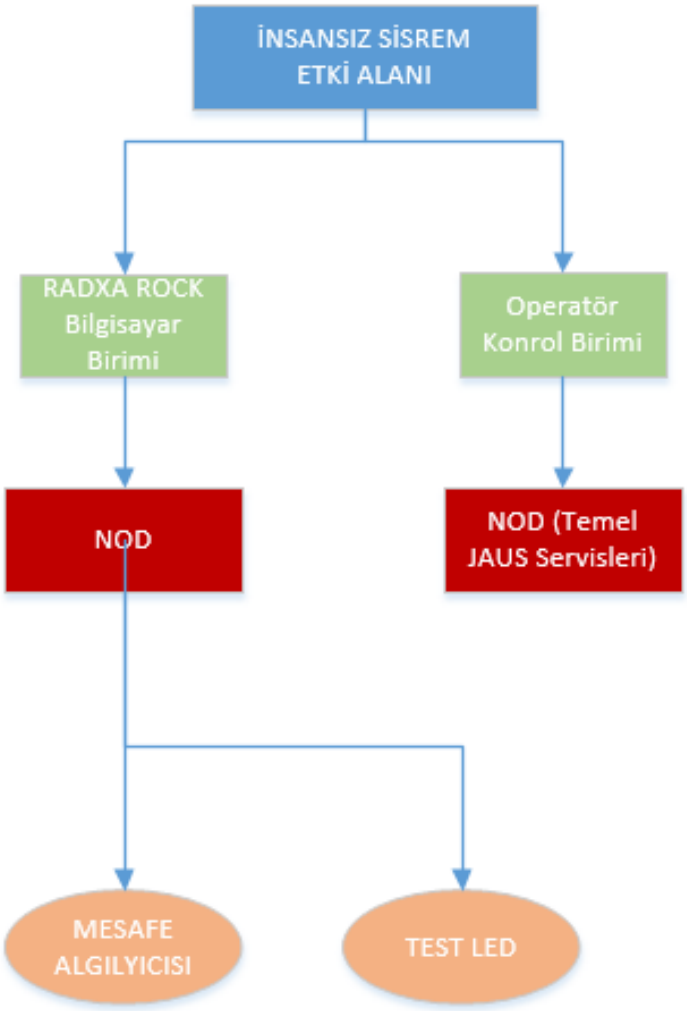
Yukarıdaki adımlarla birlikte test donanımlarımıza JAUS++ kütüphaneleri kurulmuştur. Seviye 1 uyumluluk derecesinde test donanımlarına JAUS yapılandırılacaktır. Bu donanımların insansız sistemimizin işlem birimleri olduğunu varsayacaktır. JAUS mesaj akışının ve formatsız veri akışının tanımı Şekil 3.4'de gösterilmiştir.



Şekil 3.4: Test yapılandırması şema

Radxa Rock yapılandırmasında 1 adet ultrasonik mesafe algılayıcısı ve 1 adet test led kullanılmıştır. Bu algılayıcı JAUS sisteminde bileşeni temsil etmektedir. Test ledi sistemdeki uygulayıcıyı temsil etmektedir. Test sistemimize giriş/çıkış portundan bağlanmıştır.

Algılayıcıyı ayrı bir işlem birimine bağlamadan direk olarak alt sistemimize bağladığımız için nod ve alt sistem aynı donanıma karşılık gelmiştir. Şekil 3.5'de JAUS mantıksal yapısı gösterilmiştir.



Şekil 3.5: Test yapılandırması mantıksal şema

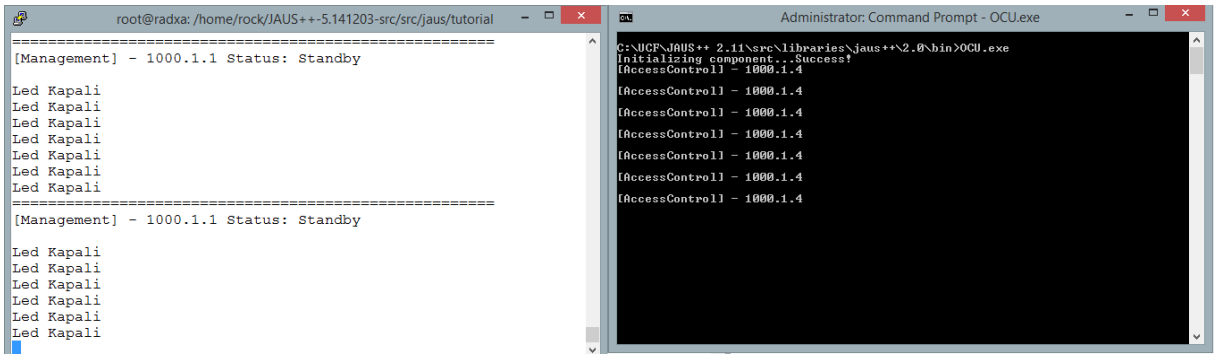
Test ortamındaki insansız sistemimizdeki bileşenler olan ultrasonik algılayıcı ve led kontrol programı python dili ve radxa rock için bulunan python kütüphanesi kullanılarak yazılmıştır. JAUS desteği eklemek için C++ programlama dilinde JAUS++ kütüphanesi kullanılmıştır.

Kütüphanenin kullanımı ile ilgili geniş bilgilendirme projenin dokümanlarından yararlanılmıştır. [13] Operatör Kontrol Birimi ise benzer şekilde Windows sistem üzerinde komut arabirimi kullanılarak geliştirilmiştir. Radxa rock ve kontrol ünitesi ile ilgili kodlama EK-1 ve EK-2'de belirtilmiştir.

Radxa Rock platformunda kod dosyası derlenirken aşağıdaki komut kullanıldı.

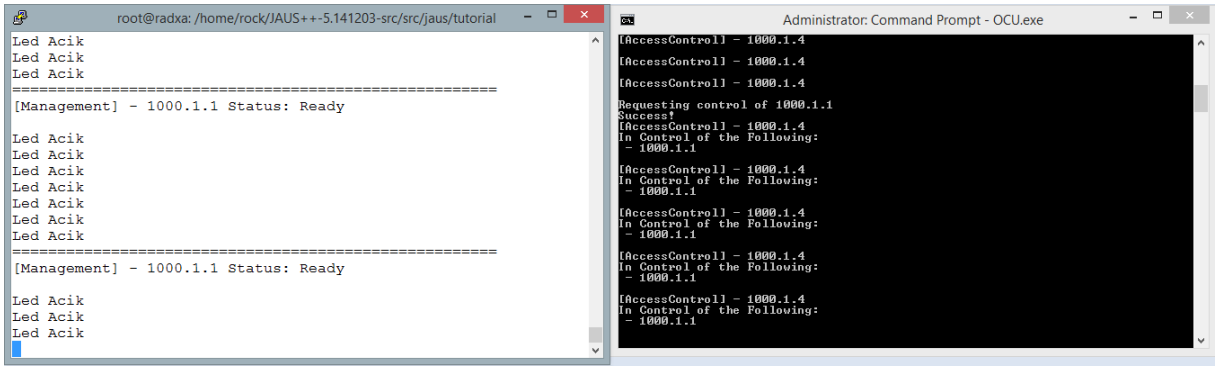
```
$ g++ -Wall RadxaRock.cpp -o RadxaRockJAUS
-L/usr/local/lib/active -lJAUSCore -lCxUtils
-I/usr/local/lib/active -lboost_system
-lJAUSExtras -lJAUSMobility -lJAUSEnviroment
```

Operetor kontrol birimi Windows ortamında Visual Studio 2013 kullanılarak derlendi. Derlenen programları çalıştırdığımızda ise Şekil 3.6'deki çıktılar alınır. Başlangıçta led kapalıdır.



Şekil 3.6: JAUS Haberleşmesi başlangıcı

JAUS mesajları operatör kontrol ünitesinden Radxa Rock bilgisayarına ulaştığı zaman led açılır ve bağlantı kurulmuş olur. Şekil 3.7'de operatör kontrol ünitesi tarafından JAUS altsisteminin kontrolü ele alınır. Bu işlemden sonra led'in yandığı da gözlemlenir.



Şekil 3.7: JAUS haberleşmesi gerçekleştirilme durumu

4. SONUÇ VE ÖNERİLER

Sonuç olarak üretilen insansız sistemlerin ve araçların birbirleriyle haberleşebilmesi için aynı dili konuşmaları önemli bir ölçüttür. JAUS referans modeli sayesinde günümüzde üretilen JAUS uyumlu insansız araçlara belirli bir protokol üzerinden erişmek mümkün kılınmıştır. Mevcut TCP/IP altyapısı ve protokollerini kullanarak ağ üzerinden erişimi en yüksek seviyede oluşturulmuştur. JAUS yazılımı sadece haberleşmeyi kolaylaştırmamış olup modüler yapısı sayesinde sisteme ekleme ve çıkarma yapılması halince yeni donanımların kolaylıkla uyarlanmasına olanak sağlamıştır. Bu mekanizmanın sağladığı yeniden kullanılabilir komutlar ve mesajlarla birlikte, sistemin daha verimli ve hızlı çalışması sağlanmıştır. JAUS'un sağladığı katmanlı yapıda, birbiriyle paralel şekilde geliştirme mümkün olduğundan, yazılımların daha çabuk gelişebilmesine olanak sağlamıştır. Ek olarak haberleşme yazılımının hazır olarak sunulması sayesinde harcanan süre en aza indirilmiştir. JAUS sayesinde insansız sistemler arasında sağlanan iletişim kullanarak birçok görev daha verimli hale getirilebilir. JAUS'un yazılımsal tasarımından ötürü oluşturulan JAUS kodları yeniden kullanılabilirdir. Bu sayede yazılım maliyetleri ve insansız sistem üzerinde çalışan program en uygun düzeyde koşturulabilir. Üretilen robotların uluslararası ölçekte daha iyi yer edinebilmesi için JAUS uyumlu olması her zaman bir artı olacaktır. Gelecek çalışma olarak ROS (Robot Operation System) JAUS yeteneklerinin birleştirilmesi önerilir. Robotik alanında sıklıkla kullanılan işletim sistemi olan ROS'a JAUS eklentisi, kod geliştiricilerin işini kolaylaştıracaktır.

KAYNAKLAR

[1] **ÇAYIRPUNAR, O.** (2009). Çoklu Robot Sistemlerinde Robotlar Arası Haberleşme ve İşbirliği Kullanılarak Arama Verimliliğinin Artırılması.

[2] (1991). IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries, *IEEE Std 610*, 1–217.

[3] **YAKIN, I. ve F., K.** (2009). Developing JAUS Compliant Communication Infrastructures for Command and Control of Unmanned Systems through MDSD.

[4] **WAGNER, C. ve ROWE, S.** (2011). An Introduction to the Joint Architecture for Unmanned Systems(JAUS).

[5] **JAUS** (2007). JAUS Message Definition, *Volume 2 Part 2, Version 3.3*.

[6] **CROUSE, J.** (2011). The Joint Architecture for Unmanned Systems: A Subsystem of the Wunderbot 4.

[7] **JAUS** (2007). JAUS Reference Architecture, *Volume 2, Part 1, Version 3.3*.

[8] **SAE-International**, (2007), AS5669: JAUS Transport Specification, `http://standards.sae.org/as5669/`.

[9] **SAE-International**, (2008), AS5710: JAUS Core Service Set, `http://standards.sae.org/as5710/`.

[10] **SAE-International**, (2009), AS6009: JAUS Mobility Service Set, `http://standards.sae.org/as6009/`.

[11] **UCF-ACTIVE-IST**, JAUS++, `http://active-ist.sourceforge.net/jaus++.php`, alındığı tarih: 2015.

[12] **IGVC**, Intelligent Ground Vehicle Competition, `http://www.igvc.org/`, alındığı tarih: 2015.

[13] **UCF-ACTIVE-IST**, JAUS++ Kütüphane Dosyaları, `http://active-ist.sourceforge.net/jaus++/html/files.html`, alındığı tarih: 2015.

EKLER

EK A.1 : Radxa Rock için düzenlenen JAUS++ kodu
EK A.2 : Operator Kontrol Ünitesi için düzenlenen JAUS++ kodu

EK A.1

JAUS++ RadxaRock Kod içeriği

```
1
2  ////////////////////////////////////////////////////////////////
3  ///
4  ///  \dosya RadxaRockJAUS.cpp
5  ///  Bu ornekte kullanilan yontemler JAUS++
6  ///  orneklerinden faydalanilarak hazirlanmistir.
7
8  //JAUS Kutuphaneleri dahil edilir.
9  #include <jaus/core/Component.h>
10 #include <cxutils/Keyboard.h>
11 #include <iostream>
12
13
14 int main(int argc, char* argv[])
15 {
16 /*Bilesen Sinifindan bir bilesen tanimlanir.*/
17 //Bu bilesenin tum Temel Servisleri Icerir
18
19 JAUS::Component component;
20
21 //Kesif servisi kullanilarak
22 JAUS::Discovery* discoveryService = NULL;
23 discoveryService = (JAUS::Discovery*)component._
24 GetService(JAUS::Discovery::Name);
25
26 //Alt sisteme isim verilir. Bilesni olusturmadan
27 //isimlendirme yaplmalidir.
28 discoveryService->SetSubsystemIdentification_
29 (JAUS::Subsystem::Vehicle,
30 "RadxaRock");
31
32 discoveryService->SetNodeIdentification_
33 ("RadxaRockAnaIslemci");
34 discoveryService->SetComponentIdentification_
35 ("TestLed");
36
37 //JAUS sisteminde erisimi olmasi icin
38 //essiz bir numara tanimlanir.
39
40 JAUS::Address componentID(1000, 1, 1);
41
42 // Bilesen devreye alinir..
```

```
43 std::cout << "Bilesen olusturuluyor...";
44 if(component.Initialize(componentID) == false)
45 {
46 std::cout << "Bilesen olusturulamadi. [" << _
47 componentID.ToString() << "]\n";
48 return 0;
49 }
50 std::cout << "Basarili!\n";
51
52 //Ana program blogunda Operator Kontrol Biriminden
53 //gelecek komutlar icin sonsuz dongu olusturulur.
54 JAUS::Time::Stamp displayStatusTimeMs =
55  JAUS::Time::GetUtcTimeMs();
56 while(true)
57 {
58
59 JAUS::Management* managementService = NULL;
60 managementService = (JAUS::Management*)component.GetService_.
61 .(JAUS::Management::Name);
62
63
64 //Kapanma sinyali gelirse programdan cikmasini saglanir.
65 if(managementService->GetStatus() ==
66 JAUS::Management::Status::Shutdown)
67 {
68 // Programdan cik.
69 break;
70 }
71
72 //Eger hazir sinyali gelirse bilesen uzerindeki test LED acilir.
73 if(managementService->GetStatus() == JAUS::Management::Status::Ready)
74 {
75 //Python kodu tetiklenir.
76 //Test amacli oldugundan sadece led acilmistir.
77 std:string command "python /home/rock/ledOn.py";
78 system(command.c_str());
79
80 }
81
82 if(JAUS::Time::GetUtcTimeMs() - displayStatusTimeMs > 500)
83 {
84 std::cout << "================================\n";
85 // Print status of the service.
86 managementService->PrintStatus(); std::cout << std::endl;
87
88 displayStatusTimeMs = JAUS::Time::GetUtcTimeMs();
89 }
90
```

```
91  if(CxUtils::GetChar() == 27)
92  {
93  //ESC tusuna basilirsa da programdan cikiyor.
94  break;
95  }
96
97  CxUtils::SleepMs(1);
98  }
99
100 //Bilesen kapatilir.
101 component.Shutdown();
102
103 return 0;
104 }
```

EK A.2

JAUS++ Operatör Kontrol Ünitesi Kod İçeriği

```
1
2  ///  \dosya OCU.cpp
3
4  ///  Author(s): Daniel Barber
5  ///  Copyright (c) 2010
6  ///  Duzenleyen: Okan BOSTAN
7  ///  Bu ornekte kullanilan yontemler JAUS++ orneklerinden
8  ///faydalanilarak hazirlanmistir.
9
10 //JAUS Kutuphaneleri dahil edilir.
11 #include <jaus/core/Component.h>
12 #include <cxutils/Keyboard.h>
13 #include <iostream>
14
15
16 int main(int argc, char* argv[])
17 {
18     JAUS::Component component;
19
20     // Setup identification info.  For questions about this,
21     // see the previous tutorial(s).
22     JAUS::Discovery* discoveryService = NULL;
23     discoveryService = component.DiscoveryService();
24     discoveryService->SetSubsystemIdentification_
25     (JAUS::Subsystem::OCU,
26     "OCU");
27     discoveryService->SetNodeIdentification_
28     ("Windows8");
29     discoveryService->SetComponentIdentification_
30     ("kontrol");
31
32     //Erisim servisi olusturulur.
33     JAUS::AccessControl* accessControlService = NULL;
34     accessControlService = component.AccessControlService();
35
36     //Test ortaminda RadxaRock makinasinin kontrol edebilmek icin
37     //kod 100 olarak set edilir.
38     accessControlService->SetAuthorityCode(100);
39
40     //Zaman asimi degeri
41     accessControlService->SetTimeoutPeriod(1);
42     //JAUS adreslemesi yapilir.
```

```
43      //(altsistem Nu, Nod Nu, Bilesen Nu) girilir.
44
45      JAUS::Address componentID(2, 1, 1);
46      // Bilesen Olusturulur.
47
48      std::cout << "Bilesen Olusturuluyor...";
49      if(component.Initialize(componentID) == false)
50      {
51          std::cout << "Bilesen olusturulamadi [" << _
52          componentID.ToString() << "]\n";
53          return 0;
54      }
55      std::cout << "Basarili!";
56
57      //Ana program blogunda Radxa Rock
58      //altsistemine komut yollanir.
59      //gelecek komutlar icin sonsuz dongu olusturulur.
60      JAUS::Time::Stamp displayStatusTimeMs = JAUS::Time::GetUtcTimeMs();
61      JAUS::Time::Stamp acquiredControlTimeMs = 0;
62      JAUS::Time::Stamp releaseControlTimeMs = 0;
63      while(true)
64      {
65          JAUS::Management* managementService = NULL;
66          managementService = component.ManagementService();
67          if(managementService->GetStatus() ==
68          JAUS::Management::Status::Shutdown)
69          {
70              // Kapatma sinyali gelirse programdan cikilir.
71              break;
72          }
73
74
75
76          // Let's see if we are in control of any components.
77          JAUS::Address::List controlledComponents;
78          controlledComponents =
79          accessControlService->GetControlledComponents();
80          if(controlledComponents.size() == 0 &&
81          JAUS::Time::GetUtcTimeMs() - releaseControlTimeMs > 3000)
82          {
83              // Kontrol icin Bilesen aranir.
84              JAUS::Address::List availableComponents;
85              JAUS::Discovery* discoveryService = NULL;
86              discoveryService = component.DiscoveryService();
87              availableComponents =
88              discoveryService->GetSubsystem_
89              (component.GetComponentID())_
90              ->GetComponentsWithService_
```

```
91              (JAUS::AccessControl::Name);
92
93              JAUS::Address::List::iterator c;
94              for(c = availableComponents.begin();
95              c != availableComponents.end();
96              c++)
97              {
98                  // Kendi bilesenimizi kontrol etmemek icin.
99                  if( *c == component.GetComponentID() )
100                 {
101                     continue;
102                 }
103                 std::cout << "Denetim istegi "_
104                  << c->ToString() << std::endl;
105                 if(accessControlService->_
106                 RequestComponentControl( (*c) ) == true)
107                 {
108                 std::cout << "Basarili!";
109                 acquiredControlTimeMs = JAUS::Time::GetUtcTimeMs();
110
111                 //Baglanilan bilesene Test amacli
112                 //Ready sinyali yollanir.
113                 component.ManagementService()->Resume((*c));
114
115                 break;
116                 }
117                 else
118                 {
119                     std::cout << "Hata!";
120                 }
121             }
122         }
123
124         //Yukaridaki kod ile alinan denetim birakilir.
125         else if(JAUS::Time::GetUtcTimeMs() -_
126         acquiredControlTimeMs > 3000)
127         {
128             JAUS::Address::List::iterator c;
129             for(c = controlledComponents.begin();
130             c != controlledComponents.end();
131             c++)
132             {
133                 //Bilesene standby sinyali yollanir..
134                 if(component.ManagementService()->_
135                 GetComponentStatus( (*c) ) !=
136                 JAUS::Management::Status::Standby)
137                 {
138                     //Standby komutu.
```

```
139                     component.ManagementService()->Standby( (*c) );
140                 }
141                 std::cout << "Denetim sona erdi ... "_
142                  << c->ToString() << std::endl;
143                 if(accessControlService->_
144                 ReleaseComponentControl( (*c) ) == true)
145                 {
146                     std::cout << "Basarili!\n";
147                     releaseControlTimeMs =
148                     JAUS::Time::GetUtcTimeMs();
149                     break;
150                 }
151                 else
152                 {
153                     std::cout << "Hata!\n";
154                 }
155             }
156         }
157 
158         if(JAUS::Time::GetUtcTimeMs() - displayStatusTimeMs > 500)
159         {
160             // Anlik durum bilgisi
161             accessControlService->PrintStatus();
162             std::cout << std::endl;
163             displayStatusTimeMs = JAUS::Time::GetUtcTimeMs();
164         }
165 
166         if(CxUtils::GetChar() == 27)
167         {
168             break;
169         }
170 
171         CxUtils::SleepMs(1);
172     }
173 
174     // Bilesen kapatilir.
175     component.Shutdown();
176 
177     return 0;
178 }
```

ÖZGEÇMİŞ

Ad Soyad: Okan BOSTAN

Doğum Yeri ve Tarihi: Balıkesir - 02.08.1986

Adres: İ.T.Ü. Bilgi İşlem Daire Başkanlığı Maslak/İSTANBUL

E-Posta: okan.bostan@itu.edu.tr

Lisans: Kontrol Mühendisliği - İ.T.Ü.