

**İstanbul Teknik Üniversitesi Robotik Arama Kurtarma Ekibi**

**Görev Raporu**

**Görev Konusu:** Yazılım Ekibi Araştırma ve Uygulama Görevleri

**Hazırlayan:** Ömer Er

**Teslim Tarihi:** 24.10.2022

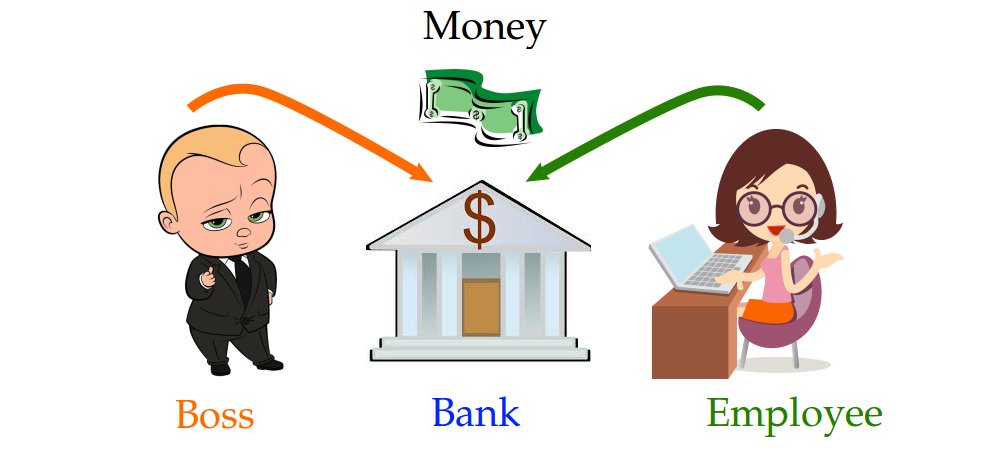
**Robot Operating System ( ROS ) Araştırması**

**ROS’u Tanıma Araştırması**

**ROS nedir?**

ROS (Robot Operating System, Robot İşletim Sistemi), 2007 yılında Willow Garage firması tarafından Stanford Üniversitesi Yapay Zekâ Laboratuvarında yapılan çalışmaların sonucunda oluşturulan bir robotik yazılım platformudur. Farklı bir deyiş ile robotlar için açık kaynaklı bir meta-işletim sistemi olarak tanımlanan ROS, temelde bir işletim sisteminden beklenilen donanım katmanının soyutlanması, donanım seviyesindeki cihazların kontrolü, sık kullanılan fonksiyonalitenin hazır biçimde kodlanması, işlemler arasındaki mesajlaşmanın sağlanması ve paket yönetimi konularını robot sistemleri için gerçekleştirmesi nedeniyle robot işletim sistemi olarak adlandırılmıştır. ROS, her biri özel işlevsellik sağlayan 2000’den fazla pakete ayrılmıştır. ROS için desteklenen işletim sistemi Ubuntu’dur. ROS’u kullanmak için Ubuntu’ya ROS yüklemek gerekir.

Aşağıdaki senaryo ROS’un nasıl çalıştığını anlatıyor.



Patron çalışanına maaş ödemek için parayı bir bankaya yatıracak. Çalışan ise maaşını bankadan alacak. Bu kısaca ROS’un mimarisidir.

ROS yayıncısı( Publisher) patrondur. (Turuncu)

ROS abonesi ( Subscriber ) çalışandır. ( Yeşil)

ROS konusu (Topic) bankadır. ( Mavi)

ROS mesajı ( Message ) paradır. (Siyah)

ROS düğümleri ( Node(s) ) patron ve çalışandır.

**Robotikte hangi amaçla kullanılır?**

Robotlar her zaman gerçekte çalışmayabilir, bu durumda simülasyon araçlarına ihtiyaç duyulur. ROS, Rviz ve Gazebo gibi birçok harika araca sahiptir. Gazebo ile çevreye bazı fiziksel kısıtlamalar bile eklenebilir, bu nedenle simülasyonu ve gerçek robot çalıştığında sonuç hemen hemen aynı olur. Doğrudan bilgisayarda bir drone ile bir odayı 3D olarak haritalandırabilir ve bu çok fazla zaman kazandırabilir. Simülasyon araçları ayrıca eğitim amaçlı veya belirli bir ortamda test yapmak için sahip olunmayan diğer robotları görmeyi ve kullanabilmeyi sağlar. ROS, birden çok ROS yöneticisiyle çalışabilir. Bu, her biri kendi ROS sistemine sahip birçok bağımsız robotunuz olabileceği ve tüm robotların kendi aralarında iletişim kurabileceği anlamına gelir. ROS’un çekirdek tabanı fazla yer ve kaynak gerektirmez. Çekirdek paketleri hızlı bir şekilde kurulabilir ve birkaç dakika içinde başlanabilir. Ayrıca, ROS Raspberry Pi 3 kartları gibi yerleşik bilgisayarlarda da kullanılabilir. Böylece çok fazla sorun yaşanmadan kolayca yeni bir projeye başlanabilir.

**ROS Elemanlarını Öğrenme Araştırması**

**ROS workspace, nodes, topics, services, master-slave structure, parameter server**

ROS’ta kod yazmaya başlamadan önce bir çalışma alanı (**workplace**) oluşturmak gerekiyor. Çalışma alanı, ilgili ROS kodu parçalarını depoladığınız bir dizi klasördür. ROS’taki çalışma alanlarının resmi adı catkin çalışma alanlarıdır. Çalışma alanı organizasyonda ve iletişimde ROS’un kendi üretmiş olduğu çözüme uygun dosya ortamı oluşturacaktır.

Bir ROS **düğümü (node)** temel olarak hesaplama yapan bir işlemdir. Uygulamanın içinde çalışan yürütülebilir bir programdır. Birçok düğüm yazılabilir ve bunlar paketlere koyulabilir. Düğümler bir grafikte birleştirilir ve ROS konuları, hizmetleri vb. kullanılarak birbirleriyle iletişim kurar. 2 düğüm aynı ada sahip olamaz. Aynı düğümün birden çok örnekleri çalıştırılmak istendiğinde, ada bir önek veya sonek eklenmeli veya bunlar anonim olarak bildirilmelidir. Düğümler kod karmaşıklığını azaltır. Düğümler ayrıca büyük bir hata toleransı sağlar. Düğümler yalnızca ROS aracılığıyla iletişim kurduğundan, doğrudan bağlantılı değildir. Bir düğümün çökmesi diğer düğümlerin çökmesine neden olmaz. ROS dilden bağımsızdır. Bunun anlamı, Python’da bir düğüm, C++ da başka bir düğüm yazılabilir ve her ikisi de sorunsuz bir şekilde iletişim kurabilir.

**Konular(topics),** düğümlerin mesaj alışverişinde bulunduğu otobüsler olarak adlandırılır. Konular, bilgi üretimini tüketiminden ayıran anonim publish/subscribe olma semantiğine sahiptir. Genel olarak, düğümler kiminle iletişim kurduklarının farkında değildirler bunun yerine, verilerle ilgilenen düğümler ilgili konuya abone olur; veri üreten düğümler ilgili konuya yayımlar. Bir konunun birden fazla yayıncısı ve abonesi olabilir. Konular tek yönlü, akışlı iletişim için tasarlanmıştır. Her konu, kendisine yayınlamak için kullanılan ROS mesaj türüne göre yazılır ve düğümler yalnızca eşleşen bir türe sahip mesajları alabilir.

**ROS hizmeti(service**), bir istemci/sunucu sistemidir ve senkrondur. İstemci bir istek gönderir ve yanıt alana kadar engeller. ROS hizmetleri sadece hesaplamalar ve hızlı eylemler için kullanılmalıdır. Bir service, bir isim ve bir çift mesaj ile tanımlanır. Bir mesaj istek, bir mesaj cevaptır. Örneğin c++ için rosccp kitaplığı ve Python için rospy kitaplığı kullanılarak, ROS düğümleri içinde doğrudan service istemcileri ve sunucular oluşturulabilir. Bir service sunucusu yalnızca bir kez var olabilir, ancak birçok istemcisi olabilir. Temel olarak, sunucu oluşturulduğunda hizmette oluşturulacaktır.

**ROS Master** sistemindeki diğer düğümlere adlandırma ve kayıt hizmetleri sağlar. Yayıncıları ve aboneleri konulara ve hizmetlere göre izler. Master’ın rolü, bireysel ROS düğümlerinin birbirini bulmasını sağlamaktır. Bu düğümler birbirlerini bulduktan sonra birbirleriyle eşler arası iletişim kurarlar ve Parameter Server sağlar. ROS Master ve Slave API’leri, ROS için konu ve hizmetlerin kullanılabilirliği ve ayrıca bağlantı aktarımı anlaşması hakkındaki bilgileri yönetir. XML-RPC kullanılarak uygulanırlar ve çeşitli dil uygulamalarıyla gelirler. Döndürülen değerler [statusCode, statusMessage, value ] şeklinde listelenirler.

statusCode (int): Yöntemin tamamlanma koşulunu gösteren bir tam sayı.

statusMessage (str): Dönüş durumunu açıklayan insan tarafından okunabilir bir mesaj.

value (anytype) : Dönüş değeri, bireysel API çağrıları tarafından ayrıca tanımlanır.

**Parametre sunucu ( parameter server),** ağ API’leri aracılığıyla erişilebilen, paylaşılan, çok değişkenli bir sözlüktür. Düğümler, çalışma zamanında parametreleri depolamak ve almak için bu sunucuyu kullanır. Yüksek performans için tasarlanmadığında en iyi şekilde yapılandırma parametreleri gibi statik, ikili olmayan veriler için kullanılır. Araçların sistemin yapılandırma durumunu kolayca inceleyebilmesi ve gerekirse değiştirebilmesi için küresel olarak görüntülenebilir olması amaçlanmıştır. Parametre Sunucusu, XMLRPC kullanılarak uygulanır ve ROS Master’ın içinde çalışır; bu, API’sine normal XMLRPC kitaplıkları aracılığıyla erişilebilir olduğu anlamına gelir. Parametreler, normal ROS adlandırma kuralı kullanılarak adlandırılır. Bu, ROS parametrelerinin, konular ve düğümler için kullanılan ad alanlarıyla eşleşen bir hiyerarşiye sahip olduğu anlamına gelir.

**roslaunch**, birden fazla ROS düğümünü SSH aracılığıyla yerel olarak ve uzaktan kolayca başlatmak ve ayrıca Parametre Sunucusunda parametreleri ayarlamak için kullanılan bir araçtır. Birçok ROS paketi, birlikte çalıştırılabilecek “başlatma dosyaları” ile birlikte gelir. Roslaunch’ın kod API’si çok kararsızdır ve doğrudan kullanılmamalıdır. ROS’un başlatma işlevlerini desteklemek için birkaç grafiksel araç mevcuttur.

rqt\_launch (very experimental) başlatma dosyalarını çalıştırabilir.

rqt\_launchtree başlatma dosyalarını etkileşimli olarak incelemeye olanak sağlar.

roslaunch\_to\_dot, başlatma dosyası ağacını bir grafiğe dönüştürür ve bir nokta dosyasına kaydeder.

**YAML** bir işaretleme dili değildir. Temel olarak YAML; JSON VE XML’in bir üst kümesi olarak düşünülebilir. Aynı amaca hizmet eder, ancak insan tarafından daha rahat okunabilir ve kolay bir şekilde yazılabilir. ROS parametre YAML dosyalarını ROS catkin alanında nereye koyulacağının konusunda özel bir kural yoktur. ROS Parameter Server tarafından desteklenen herhangi bir tür kullanılabilir.

**CMakeLists.txt** dosyası, yazılım paketleri oluşturmak için CMake derleme sistemine girdidir. Herhangi bir CMake uyumlu paket, kodun nasıl oluşturulacağını ve nereye kurulacağını açıklayan bir veya daha fazla CMakeLists.txt dosyası içerir.

**MoveIt**, hareket planlama ve manipülasyon görevleri için ROS’un en gelişmiş ve esnek kütüphanesidir. En gelişmiş ters kinematik çözücüleri, yol planlama algoritmalarını ve çarpışma algılamayı tek bir birleşik ROS arabiriminde birleştirir.

**Arm Navigation ( Kol Navigasyonu):** MoveIt’in temel özelliklerinden biridir. Kolun son efektörü kullanıcı tanımlı bir yere koyması iç gerekli yörüngeleri oluşturmaktır.

**Gazebo** bir 3D simülatördür, ROS ise robot için ara yüz görevi görür. Her iki sonucu birleştirmek, güçlü bir robot simülatörü ile sonuçlanır. Gazebo ile robotlar, engeller ve daha birçok nesne ile bilgisayarda 3 boyutlu bir senaryo oluşturulabilir. Gazebo ayrıca aydınlatma, yerçekimi, atalet vb. için fiziksel bir motor kullanır. Zor veya tehlikeli senaryolarda robota zarar vermeden robot değerlendirilebilir ve test edilebilir. Çoğu zaman, tüm senaryoyu gerçek robotta başlatmak yerine bir simülatör ile çalıştırmak daha hızlıdır. Gazeboyu kurmak için birden fazla seçenek vardır. Gazebo iki bölüme ayrılmıştır. Sunucu kısmı tüm fiziği ve dünyayı hesaplarken, istemci Gazebo için grafiksel ön yüzdür. Bu nedenle bilgisayarda performanstan tasarruf etmek için tüm testleri grafik ara yüz olmadan da gerçekleştirilebilir.

**RVIZ**, birçok mevcut konu türü için eklentileri kullanarak birçok bilgiyi görselleştirmeye izin veren bir ROS grafik ara yüzüdür. Robotun kendi dünyasına ilişkin algısını ( gerçek veya simüle edilmiş ) görmeyi sağlar. RVIZ’in amacı, bir robotun durumunu görselleştirmeyi sağlamaktır. Robotun ortamında neler olup bittiğinin doğru bir tasvirini oluşturmaya çalışmak için sensör verilerini kullanır.

“rviz shows you what the robot thinks is happening, while Gazebo shows you what is really happening” from Morgan Quigley ( one of the original developers of ROS) in his book Programming Robots with ROS.

İkisi arasındaki fark yukarıda söylendiği gibidir “rviz size robotun ne olduğunu düşündüğünü gösterirken, Gazebo size gerçekte ne olduğunu gösterir.”

**Çarpışma Önleyici Algoritmalar Araştırması**

Çarpışmadan kaçınma, çarpışma algılamaya kıyasla, olası bir çarpışma algılandıktan sonra yapılan şeydir. Olası çarpışmalar, hem nesnenin hem de robotun şekline ve boyutuna, ayrıca nesnenin ve robotun konumu ve hareket projeksiyonuna bağlı olacaktır. Bir engelden kaçınmaya çalışırken, yüzleşilmesi gereken birçok karar vardır. Mevcut kararlar, robotun engelden kaçınmak için alabileceği mevcut yollara bağlıdır. Bu yollar, robotun engeli fark etmesiyle veya tüm engellerin konumlarının bilinmesi durumunda harita yüklendikten sonra belirlenebilir. Kullanılacak “doğru “ yolu bulmak, hangi algoritmanın kullanıldığına ve bununla ilişkili ayrıntı düzeyine bağlı olarak zahmetli olabilir.

**Path Planning**

Yol planlama problemi, robotun hareketinin herhangi bir noktasında çalışma alanındaki herhangi bir engelle çarpışmaması şartına bağlı olarak, bir robotu belirli bir başlangıç konfigürasyonundan istenen bir hedef konfigürasyonuna götürerek sürekli bir hareketin nasıl bulunacağıdır. Mobil robotlar için yol planlamasını uygulamak için farklı yaklaşımlar tanıtıldı. Ortama, sensör tipine, robot yeteneklerine vb. uygun olabilecek şekilde yaklaşımlar. Çarpışma olmaması ve en kısa yol gibi belirli kriterleri karşılaması gereken bir yol oluşturmak için başlangıç ve bitiş noktası arasında yol planlaması gereklidir.

**PRM Algorithm**

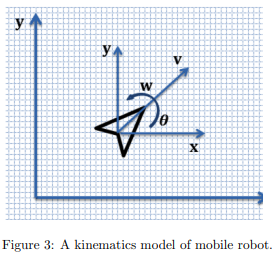
Olasılıksal yol haritaları (PRM), bir robotun çarpışmasız hızının bağlanabilirliğini yakalamak ve birçok serbestlik derecesi ile yol planlama problemlerini çözmek için etkili bir araçtır. PRM, robotun konfigürasyon alanı hakkında tam bilgiye sahip olduğumuzda, bu alandan rastgele örnekler alıp bunları test ettiğinden, boş bir alanda olup olmadıklarını ve ardından bu örnekleri bağlamak için yerel planlayıcıyı kullandığında kullanılır. Bundan sonra, başlangıç ve hedef yapılandırmaları eklenir ve başlangıç ve hedef yapılandırmaları arasında bir yol belirlemek için genellikle grafik arama algoritması ( A\* Algoritması) uygulanır.

**Bug Algorithm**

Hata algoritması, yerel bir yol planlama algoritmasıdır. Hata algoritmaları, bir mobil robot çevre hakkında sınırlı bilgiye sahip bir hedefe doğru hareket ederken en yakın engeli algılamak için sensörler kullanır. Robot, engeli hedef noktaya doğru terk etmek için algoritma kriterlerini yerine getirmek için belirli bir koşul bulana kadar engeli dolaşırken, algoritma engel sınırını hedefe doğru kılavuz olarak kullanır.

**Trajectory Data**

Yol planlama algoritması (PRM veya Bug), daha önce de belirtildiği gibi, robotun başlangıç noktasından hedef noktasına kadar tam bir yörüngesini belirler, PRM algoritması bilinen ortam içindir, ancak Bug bilinmeyen ortam için bir keşif algoritmasıdır.



**System Description**

**Kinematics of the Robots**

Çok hareketli robotların davranışlarını kontrol etmek için her zaman kinematik analize ihtiyaç vardır. Her robotun iki boyutlu bir düzlemde iki tekerlekten oluştuğu ve robotların düzlemde kaymadan hareket ettiği var sayılırsa yukarıdaki şekilde gösterildiği gibi mobil robot ve doğrusal hareketin kinematik modeli, (x,y) Kartezyen koordinatlarında şeklinde temsil edilebilir.

**Çarpışma önleme bir robot için neden önemlidir?**

Highway Safety Enstitüsü’ne göre, çarpışma önleme sistemlerine sahip araçlar daha az kazaya ve yaralanmaya bağlı araba kazalarına karışıyor. Bu güvenlik teknolojisi ölümcül araba kazalarını ve yaralanmaları azalttığını kanıtlıyor. Çarpışma önleyici sistemlere sahip araçlar, çarpışmaları önceden tahmin edebilir ve kazaları önlemek için direksiyonu otomatik olarak düzeltebilir veya fren uygulayabilir. İleri çarpışma uyarı sistemleri gibi bazı teknolojiler birkaç yıldır insanlar tarafından konuşuluyor, ancak çoğunlukla üst düzey lüks otomobillerde bulunuyorlar. Bu sistemlerin, sıklıkla görülen ciddi yaralanma ve ölümlerle sonuçlanan birçok araba kazasını önleyebileceğini belirlendi.

Çarpışma önleme güvenlik sistemleri, bir aracın etrafında olup bitenleri izlemek için bir dizi kameraya, lazere, sensöre ve kısa veya uzun menzilli radara bağlıdır. Bu güvenlik sistemleri diğer araçları, trafik sinyallerini, yol işaretlerini ve ayrıca motosikletleri, bisikletleri ve yayaları algılar ve görür. Arabadan veya sürücüden bir tür eylem isteyen bir bilgisayar sistemine bilgi girerler. Sürücünün dikkatini çekmek için bilgisayar, yanıp sönen gösterge paneli ışığı, bir dizi bip sesi, sürücü emniyet kemerinin çekilmesi veya direksiyon simidinde bir titreşim gibi uyarılar verebilir. Sürücü yanıt vermezse, daha gelişmiş sistemler fren sistemini etkinleştirir. İleri çarpışma uyarı sistemleri, önlerindeki araca çok yaklaştıklarında sürücüleri sesle uyarır. Ayrıca çarpışma etkisine hazırlanmak için fren sistemini önceden şarj ederler. Çapraz trafik uyarı sensörleri, sürücüleri yanlarına yaklaşan araçlara karşı uyarır.

**Afet sonrası ortamlarda çarpışma önleme neden önemlidir?**

Afet sonrası görevlerde görevlendirilen mobil robotlar, kendilerine atanan güvenlik görevlerini yerine getirmek için bilinmeyen dağınık ortamlarda keyfi olarak hareket edebilmelidir. Robot, bir RGBD sensörü tarafından elde edilen derinlik haritasının bir son işlem adımından türetilen geometrik özelliklerden yararlanarak ortamın çaprazlanabilirlik değerlendirmesi için makine öğrenimi tekniklerinden yararlanır. Ardından, geçilebilir sahneler, robotun önünde rastgele bir yönde çarpışma olasılığı ile değerlendirilir. Ayrıca çarpışma riski olasılığı, Hücresel Otomata dayalı bir yol izleme algoritması ile birleştirilir, böylece engelsiz bir rota tespit edilir.

**Kaynakça**

https://roboticsbackend.com/what-is-ros/

https://www.toptal.com/robotics/introduction-to-robot-operating-system

https://service.niryo.com/en/blog/8-reasons-use-ros-robotics-projects

https://www.aselsan.com.tr/tr/inovasyon/haber-detay/ros-nedir-gunumuzden-ornekler-ve-kullanim-alanlari-3423

https://www.quora.com/What-are-the-advantages-and-disadvantages-if-any-of-Robot-Operating-System-ROS

https://automaticaddison.com/how-to-create-a-ros-workspace/#:~:text=A%20workspace%20is%20a%20set,in%20ROS%20is%20catkin%20workspaces.

https://roboticsbackend.com/what-is-a-ros-node/#:~:text=A%20ROS%20node%2C%20according%20to,%2C%20services%2C%20actions%2C%20etc.

http://wiki.ros.org/Topics

https://roboticsbackend.com/what-is-a-ros-service/#:~:text=A%20ROS%20service%20is%20a,for%20computations%20and%20quick%20actions

http://wiki.ros.org/Master

http://wiki.ros.org/ROS/Master\_Slave\_APIs

http://wiki.ros.org/ROS/Slave\_API

http://wiki.ros.org/roslaunch#:~:text=roslaunch%20is%20a%20tool%20for,processes%20that%20have%20already%20died

https://roboticsbackend.com/ros-param-yaml-format/

http://wiki.ros.org/catkin/CMakeLists.txt#:~:text=The%20file%20CMakeLists.,where%20to%20install%20it%20to

https://medium.com/@jonathansumon/what-is-moveit-ros-a-jump-start-guide-to-moveit-873e0102d7e4

https://nu-msr.github.io/me495\_site/lecture18\_moveit.html#:~:text=MoveIt!%20is%20ROS's%20most%20advanced,a%20single%2C%20unified%20ROS%20interface

https://www.generationrobots.com/blog/en/robotic-simulation-scenarios-with-gazebo-and-ros/#:~:text=Gazebo%20is%20a%203D%20simulator,%2C%20gravity%2C%20inertia%2C%20etc

https://en.wikibooks.org/wiki/Robotics/Navigation/Collision\_Avoidance

http://ceur-ws.org/Vol-1894/vis1.pdf

https://cogburncares.com/collision-avoidance-systems/#:~:text=According%20to%20the%20Institute%20for,accidents%2C%20injuries%2C%20and%20fatalities

https://www.researchgate.net/publication/236166914\_Learning\_the\_Terrain\_and\_Planning\_a\_Collision-free\_Trajectory\_for\_Indoor\_Post-Disaster\_Environments