Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

**Informatika Fakultatea**

**Informatika Ingeniaritzako Gradua**

▪ Gradu Amaierako Lana ▪

Konputagailuen Ingeniaritza

# Norabide orotako plataforma gaineko beso artikulatu baten kontrola robotika asistentzialean aplikatzeko

Ainhoa Pato Sanchez

2019ko ekaina

Zuzendaria(k):
Julio Abascal
Xabier Gardeazabal

# Laburpena

Gaur egungo gizarte edadetu batean, non egunerokoan robotak aurkitu daitezken, robotika asistentzialaren premia nabaria egiten da. Lan honetan eremu horretan murgilduko gara bai gaur egungo ikerketa lerroak aztertuz, bai garatuak edo garatzen ari diren robot eta soluzio desberdinak aztertuz, baina baita ere erabiltzaile izango diren pertsonen iritzi, behar eta mugak aztertuz. Hau guztia kontuan hartuta proposatu den eta garatu den aplikazioa azalduko da, baita hau garatzeko lan ingurunea ere, honekin zer lortu nahi den eta nola lortu den zehaztuz. Bukatzeko, egindako lanaren ondorioak eta ikasitakoaren laburpena egingo da.

Konkretuki, lan honen bitartez, oinarri mugikorra eta manipulazio ahalmena duten robotak, motor ahalmenen murrizketa duten pertsonen, bai adinagatik baita desgaitasuna izateagatik, laguntzaile moduan jarduteko posibilitatearen "kontzeptu proba" bat egiten saiatu gara.

# Aurkibide orokorra

# Irudien aurkibidea

**Irudiak**



**Eskemak**



**Programak**

# 1. Sarrera

Gaur egungo gizarteak geroz eta adindun gehiago egoteko joerara doa eta, beste aldetik, jaiotze-tasak historiaren balio minimoetan daude. Honek guztiak gizarte edadetu bat uzten digu non, joera aldatzen ez bada, gazte gutxiago egongo dira pertsona zaharren betebeharrez arduratzeko. Egoera hau konpondu edo behintzat leuntzeko, robotika eta antzeko teknologi berriak erreminta bat izan daitezke.

Ikerlariak ere honetaz konturatu dira eta proposamen ezberdinak egin dituzte. Horien adibidea IoT (*Internet of Thinks*)[1] eta adimen artifiziala[4] erabilita dauden hainbat proposamen ditugu. IoT teknologiak gauza fisikoak eta birtualak konektatzeko ahalmena ematen digunez, adinduen monitorizazioan hobekuntza adierazgarriak eragiteko aukerak ematen ditu. Ingurune fisikoan txertatuta dauden sentsoreak datuak biltzeko erabiliz eta datu hauek prozesatuz (IoT erabiliz) ingurunearen jakituria handitu egiten da, monitorizazio aplikazio ezberdinak garatzeko aukera emanez, eta honekin adinduen bizi-kalitatea modu ezberdinetan hobetzea eraginez. Bestalde, adimen artifiziala erabiliz, pertsonen ohiturak antzeman daitezke, eta ohitura horietaz kanpo dagoen zerbait gertatzen denean, testuinguruaren araberako erantzun onuragarria eman. Hala ere, teknologia honek ez ditu behar guztiak betetzen berak bakarrik, eta sentsore eta roboten elkarlana behar du.

Robotikaren haritik jarraituz, robot ezberdinak proposatu dira adindunen premia ezberdinak asebetetzeko. Behar horiek mota ugarietakoak dira; batetik, mugikortasuna eta behar fisikoak, eta beste aldetik, ez dira ahaztu behar psikologikoak. Behar horiek asetzea oso interesgarria da batez ere adinduak oraindik bere etxean daudenean, honela zaharren egoitza edo etengabeko zaintzearen beharra luzatzea lortuz eta hau, adinduen irizpidetan, aukera hobeagoa[10] da. Bere etxean mantentzeko zeregin desberdinak egiteko gai izan behar dira. Zeregin horiek hiru bloketan sailkatu daitezke:  norberaren zaintza, manipulazio ekintzak eta ekintza aberasgarriak. Norberaren zaintzako zereginetan bainatzea, apaintzea, garbitzea, elikatzea, janztea edo ibiltzea sartu daitezke eta hauetatik problema gehien sortzen dituztenak garbitzea, apaintzea eta bainatzea dira, baina, baita ere, jaikitzea edo aulkietan esertzea, edota ohetan etzan edo altxatzea. Manipulazio ekintzaz hitz egiten dugunean, telefonoa erabiltzea, erosketa egitea, janaria prestatzea, arropa eta etxea garbitu eta zaintzea, medikamentuak eta finantzak kudeatzea eta garraioa erabiltzea esan nahi dugu; horietatik janaria prestatzea, etxea garbitu eta mantentzea, batez ere etxearen kanpoaldea, dira konplikatuenak adindu pertsonentzat. Azkenik, ekintza aberasgarriak deitzen diogu sozializatzeari, sozietatean parte hartzeari, gauza berriak ikasi eta zaletasunak praktikatzeari; muga fisiko edo teknologien ezagutza mugen erruengatik ekintza hauek betetzeko oztopoak topatzean, frustrazioa sortzen dute. Gaur egun badaude robot batzuk, merkatuan edo garatze prozesuan, premia hauek betetzen dituztenak eta adinduen eguneroko bizitzan euskarri potentziala izateko aukera direnak.

Hala ere, adinduak etengabeko zaintzapean edo zaharren egoitzetan egotea behar dutenean edo beraiek horrela nahiago dutenean, badaude ere aukera desberdinak zaintza horiek hobetu edo errazteko, honela euren bere bizi kalitatea hobetzeko asmoz. Honen adibide izan daiteke zaintzaileari modu telematikoan adinduak zaintzeko aukera ematen duen robota[7][8]. Robot hau korridore eta geletan ipinitako marken bidez orientatzen da eta automatikoki ibilbide bat jarraitzen du. Bere ibilbide horretan egoera ezohiko bat detektatzen badu (adibidez, pertsona bat lurrean etzanda) erabiltzaileari abisatzen dio (zaintzaileari) eta honek robota kontrol manualera pasa dezake. Modu honetan erabiltzaileak kontrol haptiko baten bidez kontrolatu dezake robota urrutitik. Adindu eta zaintzaileei bizitza errazago egitea lortzen duen beste robot bat Pearl[5] da. Robot honek adinduei oinarrizko eta beharrezko eginkizunak egitea gogorazten dio, hala nola, elikatzeaz, edateaz, medikazioa hartzea eta bainura joatea, baita pertsona horren ingurunean gidatzea ere. Pearl zaharren egoitzetan probatuta izan da baina etxeetan ere bere laguntza ezarri daiteke.

Komentatu diren baliabide guzti hauek adindun pertsonen independentzia luzatzea eta zaintza beharrak geroz eta beranduago iristea lortzea dute helburu. Egin den lan honek ere norabide horretan lagundu eta ekarpen berriak gehitzea du azken mugatzat. Hala ere, adinduen eta erabiltzaileen onurarako (zaintzaileen kasuan), robotak ez dira pentsatu behar gailu funtzional edo praktiko moduan bakarrik, azken finean pertsonarekin batera elkar eragiten bai dute hainbat momentuetan. Horren ildotik, hainbat garatzaile robotak sozialagoak egiten saiatzen ari dira, adibidez, Hobbit[2]-en kasuan. Hobbit lehen aipatu diren bezalako robot funtzionala da eta larrialdi egoerak detektatzen ditu. Besteak beste objektu txikiak beso artikulatu baten bidez hartu ditzake. Nahiz eta funtzionalitate asko izan, garatzaileak laguntza sozialagoa egiten saiatu dira, horretarako hainbat funtzionalitate gehituz, hala nola, elkar zaintza non, erabiltzaileak jakin gabe, robota bere portaera aldatzen joaten den honekin harreman estuagoak eraikiz. Honen adibide da, esaterako, erabiltzailearekiko elkarrizketa modua aldatzen joaten dela, mesedeak bueltatuz, pro-aktiboa izaten eta bere presentzia erabiltzailearen gertu mantentzen duela. Beste funtzionalitate sozial bisualagoa bere buruko "begiak" sentimenduen arabera aldatzea izan daiteke. Hauez gain, funtzionalitate gehiago ditu, hala nola, keinu antzematea, posizio antzematea, etab.

Beste aldetik, azken honengatik, badago eztabaida etiko bat non aditu, ikerlari, baita gizartearen parte bat ere kezkatuta dauden robotak pertsonen tokia kendu al dutelako. Baina adinduei roboten pertzepzioari buruz galdetzen bazaie[9], garbi gelditzen zaie robota lagungarria izan daitekeela baina ezin dutela pertsonen harremana ordezkatu edo bakardadearen soluzioa izan, baizik eta beste pertsonekin elkarrekintza hobetu edo behar sozial batzuk bete.

# 2. Helburuak eta metodologia

## 2.1 Helburuak

Lan honek bi helburu ditu: proiektu handiago baten oinarria izatea (horretarako eskuliburu bat idatzi da, A. eranskinean jaso dena) eta erakutsi erabilitako robota edo antzekoek adinduen, zaintzaileen, mugimendu murriztuko pertsonei eta hauen familiei bizitza erraztu eta hobetzeko ahalmena dutela. Beti ere kontuan hartzen da erabilitako robota ez dela produktu final bezala egokia, izan ere ez da ekonomikoki jasangarria edozeinentzat, baina bere ezaugarriak erabilita horretara eraman daitekeela frogatu nahi da.

Honetaz aparte, helburu pertsonalak ere badaude, hala nola, robotak programatzen eta erabiltzen ikasi eta esperientzia hartu, ikerketa talde baten funtzionamendu mekanikak eta honen batera lan egiten ikasi eta, azkenik, ikerkuntza memoriak (hau bera bezalakoa) idazten eta defendatzen ikasi edo hobetzea.

## 2.2 Metodologia

Lan metodologiari dagokionez, aplikazioa eta lanaren garapena inkrementala izan da, hau hainbat faseetan banatuz, bai kodea idazterakoan bai ikasitakoan. Honen eredu da kodearen diseinu modularra, garatu diren hiru moduluak programa nagusi batean batuz. Gainera modulu hauek (hauen funtzioek) geroago berrerabili daitezke beste aplikazio batzuk garatzeko.

### 2.2.1. Lanaren garapenaren faseak

Aipatu diren fase horiek bost hauek izan direla esan dezakegu, nahiz eta, batzuetan paraleloki bat baino gehiago egin, adibidez garapena eta memoria idaztea edo ikastaldia zein lan prozesuaren fase guztietan egin den.

#### 2.2.1.1. Ikastaldia

Hasieran, hainbat gauza ikasi behar izan dira robota maneiatu baino lehen: ROS, C++[3], robotaren funtzionamendu eta ezaugarriak, etab. Horretarako hainbat dokumentazio[6] irakurri behar izan nuen. Hala ere, fase hau ez da beste faseak hasterakoan amaitu, izan ere beste faseak egiteko ere gauzak ikasi behar izan dira: besoa mugitzeko ROS-eko liburutegiak, robotaren besoaren funtzionamendu eta ezaugarriak; objektuak detektatzeko dauden modu desberdinak eta horretarako liburutegiak; nabigazioa egiteko modu ezberdinak eta ROS-eko mezuak eta azkenik, memoria hau egiteko metodo eta arauak.

#### 2.2.1.2. Besoa mugitzea

Hasieran, GRAL (Gradu Amaierako Lana) honek besoa mugitzea bakarrik zuen helburu gisa eta horretarako hainbat aukera esploratu dira. Lehendabizi, aginte bat erabiliz plataforma mugitzeko iadanik garatuta zegoen aplikazio batean oinarrituta, modu berean besoa ere mugitzeko ahalmena ematen zuen aplikazioa garatzen saiatu zen. Baina, azkenean, aplikazio hau alde batera utzi zen ikusirik bazegoela horrelako aplikazioa, robotaren saltzailea eskainita. Hala ere, garapen prozesu hori esker ROS eta bere funtzionamendua hobeto ulertzea ahal izan zen, geroago egindako aplikazioan mekanikak (argitaratzaile-harpidedun mekanika, *publisher-subscriber* ingelesez) eta mezuak erabiliz.

Gero, besoa mugitzeko, zuzenean besoaren ardatz bakoitzari balioak ematera probatu zen eta honekin demo bat garatu. Demo horrek azkeneko aplikazioa egiten duen antzeko zerbait

egiten du: zama eramateko plataforman zegoen objektu bat hartu, metro bat aurrera ibiltzen zen eta bere parean dagoen pertsona bati ematen dio objektua, momentu honetan beste objektu bat eman dezakegu eta segundo batzuen ondoren emandako objektua berriro plataforman uzten du; bukatzerakoan besoa bere posizio erlaxatuan uzten du eta atzera bueltatzen da (metro bat) hasierako posizioan utziz.

Azkenik, besoa mugitzeko aukeretan bilatuz, *rviz*-en (ROS-ek robotak hiru dimentsioetan bistaratzeko tresnaren) *Moveit!*[15] plugin-a topatu nuen, zein robotaren besoa posizio batetik bestera joateko egin behar duen ibilbidea kalkulatu eta simulatzen duen. Hau ikusita, konturatu ginen bere alderantzizko zinematika[20] (*inverse kinematics* ingelesez) erabili zitezkeela beste aplikazio batean liburutegi moduan. Aukera hau izan da aplikazioan inplementatu dena eta hurrengo atal batean azalduko dena.

#### 2.2.1.3. Objektuen detekzioa

Objektuen detekzioa egiteko modu desberdinak daude ere: koloreak erabiliz, markak, kode desberdinak, eredu bat entrenatu objektuaren hainbat argazki ezberdinekin,... baina, zuzendari eta zuzendari-kidearekin hitz egin ondoren, QR kodeak erabiltzea erabaki zen. Behin hau erabakita, eta jakinik bazeudela QR kodeak detektatzeko liburutegiak, ROS-ekin batera lan egin zezakeen liburutegi bat bilatu zen eta ViSP liburutegia topatu zen.

Kinect-arekin lan egin behar zenez, honen informazioa bilatu zen, bai software bai hardware aldetik eta hainbat proba egin ziren: bere kamera ezberdinak probatuz, rviz-ekin ROS mezu desberdinak nola bistaratzen ziren, etab…

Behin kamera miatuta zegoela ViSP liburutegia ROS-ekin batzeko modulua probatu zen, bere tutorialak pixkanaka aldatuz QR kodea detektatzeko Kinect-a erabiliz. Honela, zati hau aparte programatu eta probatu zen. Hasieran, proba horiek 4,5 zentimetroko kuboekin egiten ziren, baino ikusita ez zuela ondo detektatzen, txikiegiak zirelako, 6 zentimetroko kuboekin probatu zen (ikusi 4.1. irudia). Gainera, neurri hau hobeagoa zen pintzaren ireki-itxiera distantzia-tartegatik. Behin bakarka ondo funtzionatzen zuela frogatu zenean aplikazioan beste modulu bezala sartu zen, behar ziren aldaketak eginik.

#### 2.2.1.4. Nabigazioa

Robotikan nabigazioa deitzen zaio robota bere ingurunean mugitzeko ahalmenari. Hau egiteko hainbat modu daude baina erabiliena ingurunearen mapa bat egitea da. Hau da, adibidez, etxean ditugun robot aspiragailuak erabiltzen dituenak. Hasieran hau ere erabiltzea pentsatu zen, izan ere, ikerketa taldean egondako ikasle batek hau egin zuen bere GRAL (Gradu Amaierako Lana) bezala eta beraz dagoeneko garatuta zegoen. Hala ere, nahiz eta saiatu ez zen lortu ondo integratzea aplikazio honekin eta beste soluzio[18] bat bilatu zen.

Soluzio hau izan da aplikazioaren azken bertsioan erabilitakoa, aplikazioaren behar eta ezaugarrietara eraldatuz geroago azalduko den moduan. Hau ondo joateko hainbat proba egin dira eta hainbat eraldaketa. Beste moduan egitea espero zenez eta hau dagoeneko garatuta zegoela jakinik, hau izan da garatutako azken partea, behin beste guztian ondo funtzionatzen zuela egiaztatuta zegoela.

#### 2.2.1.5. Memoria idaztea eta defentsaren prestakuntza

Memoria idazteko, erabilgarri zegoen eredua antzeratu da formatu arauak errespetatzen saiatuz. Hala ere, atalak zerbait aldatu dira lan mota honekin gehiago bat etortzen zen eredu batera edo horren helburuarekin. Memoriaren idazketa ere modu inkrementalean egin da, bertsio desberdinak garatuz zuzenketak edo hobekuntzak eginik.

Honetaz gain, lana hasi zenetik egunkari moduko bat idazten joan da, azkeneko memoria hau idazteko momentuan egin den guztia bilduta egoteko intentzioarekin. Hala ere, sarrera eta ondorioak idazteko, GRAL (Gradu Amaierako Lana) honi dagokion eremuan, gaur egungo egoera aztertu da hainbat ikerketa artikulu irakurriz.

# 3. Lan ingurunea

Atal honetan erabili den lan ingurune fisiko (hardware) zein logikoa (software) deskribatuko da, elementu desberdinak deskribatuz eta bere erabilera justifikatuz. Honekin ere, hurrengo aplikazioen garatzaileak edo aplikazio hau errepikatu nahi duten garatzaileentzat argibide moduan funtzionatzeko.

## 3.1 Kuka youBot robota

Kuka youBot[12] robota (3.1 irudian ikusi daiteke) norabide orotako plataforma eta 5 ardatzeko besoa du, beso honen bukaeran pintza bat izanik. Besoa robotaren ordenagailura lotzeko EtherCAT kablea erabiltzen du eta korronteaz elikatzeko robotaren 24V-ko korronte irteeratako bat erabiltzen du. Gainera zama eramateko plataforma bat du, honek objektuak eramateko ahalmena emanik. Hala ere, badago aukera zama eramateko plataforma jarri ordez, beste beso bat ipintzea. Besoak eta honen pintzak objektuak manipulatzeko gaitasuna ematen dio; plataformak, aldiz, desplazatzeko ahalmena ematen dio. Horregatik, aplikazio orotarako aproposa da, izan ere, ikerkuntzarako pentsatuta dago.


3.1. Irudia. Kuka youBot robota

Gure aplikazioan aspektu eta prestazio guzti hauek erabili dira beso bakar bat ipiniz, robot honen (eta antzekoen) gaitasunak erakusteko asmoz.

## 3.2. Microsoft Kinect (Xbox 360 bertsioa)

Robotak ez du ingurunea antzemateko gaitasunik, beraz, robotari erantsi daitezkeen gailu eta sentsore ezberdinen bitartez lortu beharko genuen hau. Teknologia desberdinak daude honetarako: infrasoinuak, laser izpiak, bluetooth, etab. baina Microsoft Kinect-a erabiltzea aukeratu da teknologia ezberdinak konbinatzen dituelako eta beraz, proiektu ezberdin orotarako erabili daiteke (ikusi 3.2 irudia). Kinect-a Xbox kontsolarako pentsatutako gailua da, jokoetan ibiltzeko kontroladore moduan. Lehenengo bertsioa Xbox 360 kontsolarekin atera zen 2010 urtean eta urte bat beranduago Windows ordenagailuekin ibiltzeko. Hau erabiltzaile interfaze natural bat da zein keinuen, ahozko aginteen, objektu eta irudien antzematea egiteko kapaz den. Hau guztia egin ahal izateko RGB kamera, sakontasun sentsorea (infragorri izpietan oinarrituta), matrize orotako mikrofonoa eta prozesadore pertsonalizatua ditu; guzti hau 3D gorputzen

mugimendua, aurpegi eta ahots antzematea bezalako aplikazioak egiteko ahalmena ematen dio.


3.2. Irudia. Microsoft Kinect Xbox 360-rako.

Nahiz eta hasieran sakontasun sentsorea erabili den plataformaren nabigaziorako, azkenean gure aplikaziorako RGB kamera eta zuri-beltzean bakarrik erabili da, honi irudi zuzenketarako iragazkia ipiniz. Gure kasuan gailua robotaren aurrealdean jarri da (4.2. irudia ikusi), honela nabigazioa eta objektuak ikusi eta detektatu aldi berean egiteko gai izateko. Bere momentuan, beste aukera batzuk eztabaidatu ziren objektuak detektatzeko, adibidez beste kamera bat ipintzea besoaren bukaeran baino ikusirik batekin bakarrik nahikoa zela honela utzi zen. Gainera, aukera honetatik joanda kamera besoaren posizioaren arabera aldatuko zen eta beraz, objektu bat detektatzerako orduan bere posizioa determinatzeko besoaren posizioa ere kontuan izan beharko zen. Kinect-a oinarriaren aurrealdean ipiniz objektua detektatzeko momentuan soilik oinarriaren posizioa kontuan izan behar dugu eta hau sinpleago egiten du oinarria mugitzea objektuaren posizioraino.

Kinect-a datuak partekatzeko USB konektorea du eta hau problemarik gabe robotaren USB portuan konektatu daiteke baina USB portua honek ematen dion korrontea ez da nahikoa gailua modu hobenean funtzionatzeko eta honegatik loki elektrikoarekin ere etortzen da behar duen korronte adina lortzeko. 4.3. irudian robotarekin konektatu ahal izateko egin behar izan diren konexioak ikusi daitezke, izan ere, elikadura konektorea moldatu egin da robotaren konektorera robotak ez duelako etxean erabiltzen den konektorea korrontearen irteeran eta korronte hori 24V-koa delako. Konexio hauek honela ipintzeak ikerketaren beharretara erantzuten du, produktu final batean konexio guztiak ezkutatuta edo robotean txertatuta egongo lirateke.

## 3.3. ROS

ROS[14] (*Robot Operating System)* nahiz eta izenean sistema eragilea dela adierazi, garapen ingurune bat da, garatzaileari hainbat tresna eta liburutegiekin hornituz robotak goi mailako lengoaietan programatzeko ahalmena ematen diolarik. Honen agerpena baino lehen, robot bakoitzerako software espezifikoa garatu behar zuten garatzaileek eta robot batekin egindakoa bestelakoekin berrerabili nahi bazen, hasiera hasieratik hasi behar zen, bere liburutegi, garapen ingurune eta sistema eragile desberdina erabiltzen ikasiz. ROS-en erabilera ikerkuntza robotikoan gora egitea ez da arrazoi gabea, izan ere, honen onura bat da esandako azken hori gaindituta gelditzen dela, hau da, ez dela jakin behar roboten software eta hardware espezifikoa hau programatzeko, eta beraz, lan honetan erabili da aurreko atalean azaldu den bigarren helburua ahal den gehien bermatu egiten delako, etorkizuneko garatzaileek hasieratik hasi behar gabe.

ROS-ek hainbat tresna eta mekanika desberdinak eskaintzen ditu, hala nola, bezero-zerbitzari eredua jarraitzen duten programak, makinen arteko maisu-morroi eredua eta mezuak buzoietan bidaltzeko mekanika bat. Azken hau izan da aplikazioan gehien erabili dena. ROS-en

buzoi horiek *topic* deitzen dira eta aplikazioek (*node* deiturikoak) buzoi horietan publikatu edo jaso ditzakete mezuak. Mezu horiek mota desberdinekoak dira eta nahiz eta ROS-ekin mota batzuk lehenetsirik badatozen, garatzaileak bere mezu pertsonalizatuak definitu ditzake, nahi duen aplikaziora egokituz. Hala ere, aplikazio honetarako ez dira mezu berriak definitu, erabilitako liburutegiek eskaintzen zituzten batzuk erabili bai dira.

ROS erabiltzeko beste arrazoi bat da, ROS-en konfigurazio orokorrean bi lerro idatziz konputagailu batean exekutatzen ari dena sarearen bitartez beste batekin komunikatzeko ahalmena duela. Konputagailu batek maisu edo zerbitzari moduan funtzionatuko du eta ROS-en muina (*core*) exekutatuko du, bera ere beste aplikazioak paraleloki exekutatu ahal izanik; beste makinak, morroi edo bezero moduan funtzionatuko du ROS muinarekiko eta, azken finean, bi makinetan exekutatzen ari dena batean egingo zen bezala funtzionatuko du, honen latentzia atzerapenak salbu. Hau erabilgarria da gure robotak ez badu aplikazioren bat exekutatzeko behar den kalkulu abiadura, aplikazioaren lan karga banatzen denez.

# 4. Aplikazioa

Aurreko ataletan lanaren helburu orokorrak eta etorkizunean osatu nahi den proiektu handi batez hitz egin da. Dokumentu honetako sarreran adinduen behar eta egunerokotasunean egiten diren atazetatik arazo edo muga gehien direnak ere adierazi dira. Informazio hori kontuan izanik, atal honetan egindako lana azalduko da, garatu daitezkeen aplikazio ezberdinetatik aplikazio bat.

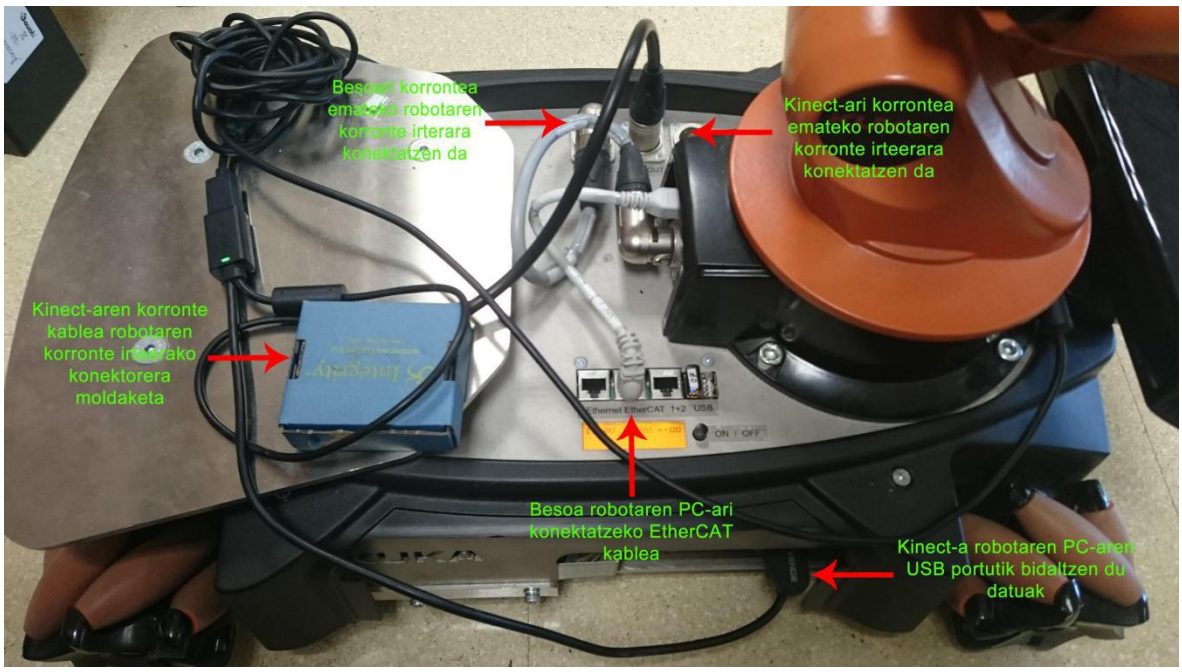
Garatu dugun aplikazio honek adinduei edo mugimendu murriztua duten pertsonei lurrean dauden objektuak hartu eta gerturatzen diote edo beste toki batean utzi. Hasiera batean eragiketa sinplea izan daiteke, baina esan bezala, egindako aplikazio honek oinarri batzuk ipini nahi ditu hemendik hasita eragiketa horiek konplexuago egiteraino, adibidez, etxea txukundu eta ordeinatzea edota erabiltzailearen monitorizazioa egitea.

4.1. Irudia. QR kuboak. Ezkerrean 6cm, eskubian 4.5cm.

Gure eragiketa honen kasuan objektu hori 6cm-ko kuboa da. Kubo hau detektatzeko QR kodeak ipini zaizkio, izan ere, dagoeneko inplementatuta daude hori egiten duten liburutegiak. Kode hauen ordez, badaude objektuak detektatu ahal izateko hauek markatzeko beste metodoak, adibidez, kolore bitartez edota beste motako kodeak (barra-kodeak, adibidez) ipiniz. Hala ere koloreekin egitea, adibidez, traba handi bat du, izan ere, robota antzematen ari denaren artean ezin daiteke egon, detektatu nahi dugun objektutik gain, kolore hori duen beste objekturik, bestela hori ere detektatuko dugu eta erroreak egitera eramango gaitu. Kodeekin egitea, beraz, aukera egokiagoa iruditu zitzaigun eta kodeen artean QR kodea aukeratu zen.


4.2. Irudia. Robota Kinect-arekin.

Gainera, aurreko atalean esan bezala (ikusi *Lan ingurunea*), youBot robotari Kinect bat konektatu zaio, aplikazio honek, zein garatu litezkeen hurrengo aplikazioek ingurunea antzeman ahal izateko eta adibidez, irudien prozesamendua egin ahal izateko edota sakontasun sentsorea erabiliz oztopoak detektatzeko.

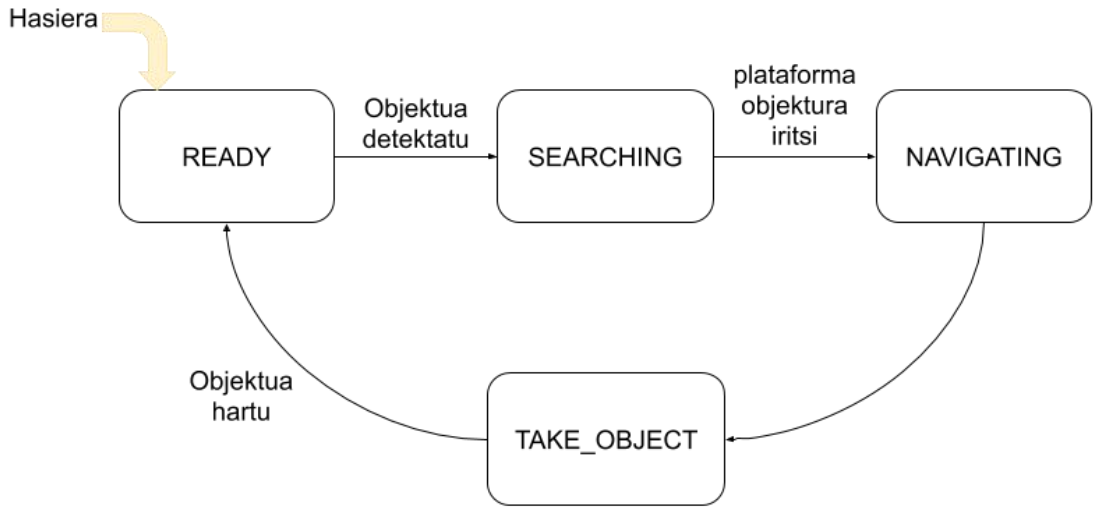



4.3. Irudia. Kinect-aren konexioak.

Egin den aplikazioaren kodea ROS-ekin egin da C++ erabilita gehienbat, nahiz eta Python-ekin ere erabili daitekeen eta erabili diren liburutegi batzuk ere erabiltzen duten. C++-ren aukera hartu da gehienbat C-rekin aurretiko esperientziagatik, jauzi gehiegi ez izateko esperantzarekin.

## 4.1. Funtzionamendu orokorra

Garatutako aplikazioak, esan bezala, mugimendu murriztua duten pertsonei objektuak gerturatzeko helburua du. Honetarako, behar diren hasieraketak egin ondoren, QR kodearen detekzioa egiten da; detekzio horretan, gero azalduko den bezala, objektuaren robotarekiko posizioa lortzen da. Behin objektua non dagoen jakinda, robota bere ondora gerturatzen da eta, azkenik, besoa eta pintza mugituz objektua hartu eta zama eramateko plataforman uzten du.

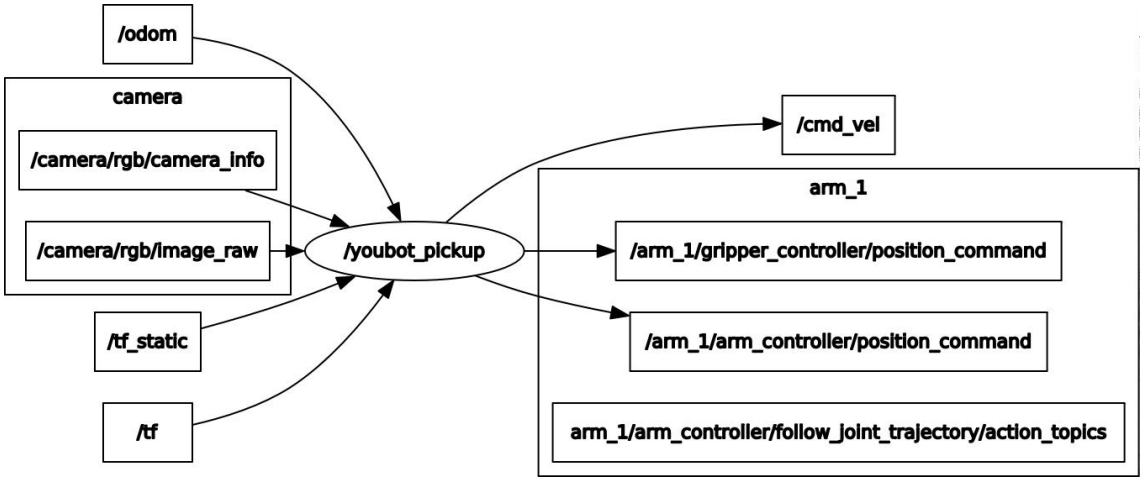



4.1. Eskema. Programa nagusiaren egoera automata.

Programa nagusiak 4.1. eskeman ikusi daiteken egoera automata jarraitzen du, honela nahi den funtzionamendua ordenatuz eta segurtasun pixka bat gehituz. Egoera bakoitzean programa nagusian hau egiten da:

- READY egoera: hasieraketa guztiak egin ondoren, egoera honetan hasiko da programa, baita objektu bat hartzen bukatzen denean ere. Egoera honetan objektua (QR kodea) detektatzeko agindua emango da eta detektatu arte itxarongo da. Detektatzen denean, objektuaren posizioa lortzen da, baina, kameraren araberako posizioa ematen denez, posizio hori robotaren oinarrira transformatu behar da (ikusi *TF liburutegia*), gero objekturaino nabigatzeko. Azken hau ondo bukatzen badu hurrengo egoerara aldatzen da, berriz, objektua ez bada detektatzen edo detektatzerakoan erroreren bat gertatzen bada egoera honetan geldituko gara detektatu eta ondo prozesatu arte.
- SEARCHING egoera: puntu honetan badugu behar dugun objektuaren posizioa eta honetara nabigatzeko agindua ematen diogu robotari, dena ondo ateratzen bada hurrengo egoerara joanez.
- NAVIGATING egoera: suposatuz objektua hartzeko posizio egoki batera iritsi garela, hurrengo egoerara goaz.
- TAKING_OBJECT egoera: egoera honetara zuzenean joango da programa besoa mugitzen hasi baino lehen. Honen arrazoia segurtasunarekin du harremana, izan ere, egoera hau egongo ez balitz, eman daiteke egoera bat non besoak oraindik mugitzen bukatu ez duen eta hurrengo objektua prozesatzen (eta honetara hurbiltzen) hasi dena. Beraz, egoera honekin ziurtatzen gara besoa mugitzen bukatzen denean berriro

automataren hasierako egoerara pasatzen garela, egon daitekeen beste objektu bat hartzeko.




4.2. Eskema. Aplikazioarekin (*youbot_pickup*) komunikatzen diren *topic* ezberdinak.

Moduluak eta liburutegiak deskribatzen diren ataletan gehiago hitz egingo dugu honetaz, baino 4.2. eskeman ikusi daiteke gure aplikazioarekin komunikatzen diren *topic* ezberdinak. Ikusten denez, batzuk taldeetan organizatuta daude eta izenaren lehenengo partea talde honen izena da. Honetaz gain, gezien bitartez ikusi dezakegu zein *topic*-etara harpidetzen den, hau da, informazioa eskuratzen dugun eta baita ere zeinetan argitaratzen dugun informazioa.

Programa exekutatzeko, robotarekin konektatu behar da (ikusi A. eranskinean nola) eta hurrengoa exekutatu:

```
roslaunch youbot_pickup youbot.launch
```

Honekin, behar diren kontroladoreak eta tresnak exekutatuko dira (ikusi *Fitxategi jaurtitzailea* atala informazio gehiagorako) eta programa exekutatzeko prest egongo gara beste terminal batean ondokoa ipiniz:

```
rosrun youbot_pickup youbot_pickup_node
```

## 4.2. Liburutegiak

Gure aplikazioan gauza ezberdinak egiten dira eta horietako bakoitzak ebazteko denbora askoz gehiago beharko genuke, horregatik, beste garatzaileek aurretik egindako eta partekatutako kodeaz (liburutegiak) baliatu gara. Atal honetan erabilitako liburutegi horiek aipatuko dira eta bakoitzaren deskribapen labur bat emango da. Liburutegi bakoitzaren erabilpena, berriz, hurrengo ataletan zehaztuko da.

### 4.2.1. Freenect

Freenect komunitate[13] irekian garatutako kode irekiko Microsoft Kinect-aren kontroladorea da. Komunitate konek modulu ezberdinak garatu ditu, baita programazio lengoai erabilienekin programatzeko liburutegiak (wrappers) ere. Gure kasuan ROS-ekin ibiltzeko kontroladorea[19] erabiltzen dugu. Kontroladore honek, Kinect-aren kamera eta sakontasun sentsorearen datuak ROS-en *topic*-etara pasatzen ditu. Hau ROS erabiltzen duten aplikazioei *topic* horietara harpidetuz Kinect-aren datuak tratatzeko ahalmena ematen dizkio. Izan ere, Kinect dispositiboa ROS-ekin konektatzeko aukera hau izan da liburutegi hau erabiltzeko arrazoia.

### 4.2.2. Youbot manipulation

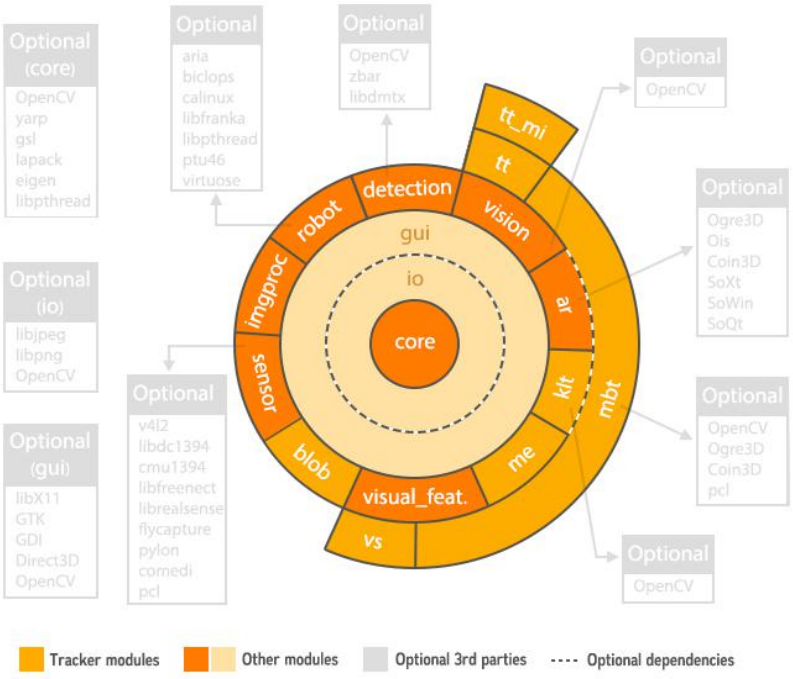
Svenschneider-en[21] Youbot-Manipulation Github biltegiak, robotaren besoa *Moveit!* plugin-arekin ibiltzeko behar diren kodea, konfigurazioa eta jaurtiketa fitxategiak ditu, bai robot fisiko nahiz simulatuarentzat. Hala ere, gure kasuan, kodea erabiliz, besoa alderantzizko zinematikak erabiliz posizio jakin batera mugitzeko erabili da.

Besoa modu ezberdinetan mugitu dezakegu (*Besoa mugitzea* atalean adierazi bezala) baina modu hau seguruagoa da, izan ere, kontuan hartzen da besoaren hasierako posizioa eta honekin besoa plataforma edo beste oztopo batekin talka ez egitea. Honegatik, eta batez ere pertsonekin ibiliko den robotarentzat, oso garrantzitsua da posizio batetik bestera joateko ibilbide planifikazio bat egitea eta hau da, hain zuzen ere, liburutegi hau erabiltzeko arrazoia.

### 4.2.3. ViSP

ViSP[11] (*Visual Servoing Plataform*), liburutegi modular eta plataforma anitza, irudien tratamenduen hainbat teknika erabiltzeko erraztasunak ematen dizkio aplikazio ezberdinen garatzaileei. Besteak beste, erabilgarria izan daiteke robotikan, konputagailu bidezko ikusmenean, errealitate areagotuan eta ordenagailu bidezko animazioetan. Hala ere, guretzako ezaugarri interesgarriena denbora errealeko irudiak prozesatzeko ahalmena da, honela objektu, marka edo kode jakina(k) detektatzeko.

Modularra izanik, nahi ditugun tresnak erabili ditzakegu behar ez ditugunak alde batera utziz. Hala ere, OpenCV eta bestelako liburutegiak ere edukitzea eskatu dezakete, izan ere, 4.3. eskeman ikusten denez, liburutegi horiek erabiltzen dituzte modulu ezberdinetan. Tresna hauek modu erraz batean ROS-ekin elkarlanean erabiltzeko aukera hori izan da liburutegi hau erabiltzeko arrazoi nagusia.



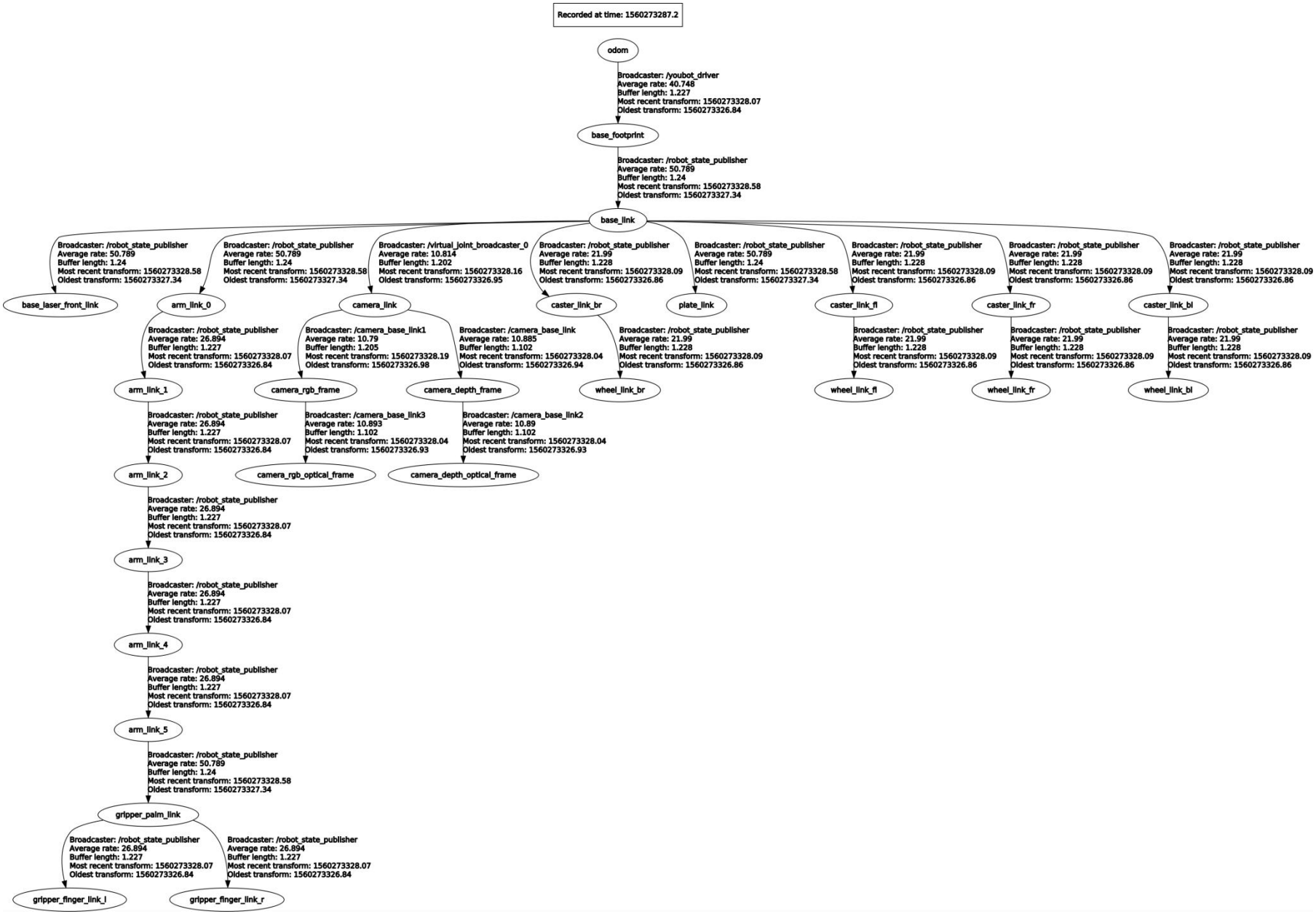

4.3. Eskema. ViSP liburutegiaren modulu ezberdinak[1].

[1] ViSP web orriaren “Software Architecture.” atala, 2019/06/20an eskuratuta: visp.inria.fr/software-architecture/.

### 4.2.4. TF

TF liburutegi oso erabilia eta erabilgarria da, izan ere, nahi dugunaren koordenatuak gorde eta denbora errealean segi dezakegu. Tipikoki koordenatu horiek robotaren elementu ezberdinarenak dira: besoaren ardatzak, pintza, robotaren oinarria, etab. eta liburutegi honek bata bestearekiko eta denborarekiko koordenatuen erlazioak gordetzen ditu. Azken hau egiteko TF elementu bakoitza zuhaitz batean kokatzen da beste elementuen erlazioen arabera eta, garatzaileen beharren arabera, koordenatuak puntuetan, poseetan, bektoretan, etab. transformatu daitezke.

4.3 eskeman ikusi daiteke garatu dugun aplikazioan eraikitzen den eta youBot robota (bere elementuak) definitzen duen TF zuhaitza. Zuhaitz honen erroa odometria da normalean (ikusi *Oinarriaren mugimendua* eta *nav_msgs* informazio gehiagorako) eta gero robotaren oinarria, hemendik oinarrian dauden elementu guztiak ateratzen dira: besoa bere bost ardatzekin eta honen buruan dagoen pintza, zama eramateko plataforma, lau gurpilak eta Kinect-a bere RGB kamera eta sakontasun sentsoreekin. Azken hau guk gehitu dugun dispositiboa denez eta robota jatorrian ez zuenez, bere kokapena oinarrian kokatzeko, kontroladorea publikatzen ez duen TF elementu bat publikatu behar dugu Kinect-a eta oinarriaren lotura moduan funtzionatzeko (elementu hau da 4.4. eskeman *camera_link* elementua). Hau nola egin *fitxategi jaurtitzailea* atalean azalduko da. Ikusten dugunez, elementu bakoitza, elementuaren izena eta beste elementurekiko informazioaz gain, beste motako informazioa ere gordetzen du: aplikazio edo baliabidetik ari garen exekutatzen, zein tamainakoa da transformazio mezuen ilara (*Buffer lenght*), ilararen azkeneko eta lehenengoa mezua (identifikazioen bitartez identifikatuta) zein izan den eta zenbateko abiaduran bidaltzen diren transformazioen mezu horiek (*Average rate*).




4.4. Eskema. Garatutako aplikazioan eraikitzen den TF zuhaitza.

### 4.2.5. ROS mezuak

ROS-ri buruzko atalean gure aplikazioan ROS mezu liburutegiak erabiltzen direla eta mezu horien funtzionamendu orokorraren azalpena eman da. Atal honetan, berriz, gure aplikazioan erabili diren liburutegi zehatzak aipatu eta hauen erabilgarritasuna azalduko da.

#### 4.2.5.1. geometry_msgs

ROS-eko liburutegi arrunta. Hau mezu geometriko arrunt ezberdinak definitzen ditu, hala nola, bektoreak, puntuak edo poseak. Hauen artean interesgarriak dira Twist motako mezuak oinarria mugitzeko abiadurak adierazteko eta Pose motakoak, bai besoaren posea determinatzeko momentuan bai QR kodearen posizioa determinatzeko eta honetara gerturatzeko.

- **Twist** motako mezuak hiru dimentsioko bi bektore ditu barnean, non bat abiadura lineala adierazten duen eta bestea abiadura angeluarra (errotatzeko). Bektore hauen hiru dimentsioak, hiru dimentsioko koordenatuak egokitzen zaizkio (x, y, z) eta beraz, balioak ematerakoan koordenatu horretan linealki edo egingo dugun errotazioaren abiadura adierazten dugu.
- **Pose** motako mezuak ere bi elementu dituzte baino honetan bi elementu horiek objektu baten orientazioa eta posizioa adierazten dute. Posizioa liburutegi honen (geometry_msgs) Point motakoa da, mota honek hiru dimentsioko puntu bat erakusten du (x, y, z). Orientazioa adierazteko koaternioia (Quaternion mota) erabiltzen da, zenbaki konplexuen hedadura bat. Hau lau dimentsiotako bektore espazioa osatzen du (x, y, z, w).

Mezu hauek erabili dira, batetik, ROS-ko muinarekin datozelako eta bestetik erabiltzen diren beste liburutegiekin parekotasunak dituztelako edo batetik bestera eraldatu daitezkeelako, adibidez, Pose mota hau eta TF liburutegiaren Pose mota edota Transform mota, gure interesen arabera. Twist mezuak erabiltzea, berriz, *Besoa mugitzea* atalean azaldu den aginte baten bidez plataforma urrutitik mugitzeko aplikaziotik dator, izan ere, mezu mota hau zen aplikazio horretan erabiltzen zirenak.

#### 4.2.5.2. brics_actuator

Erabili den beste mezu definizio liburutegi bat[22]. Honek robotaren besoa mugitzeko erabili diren mezu motak definitzen ditu eta espresuki robot honentzako idatzita dago saltzaileen eskutik. Besoa mugitzeko, liburutegiaren mezuen artean **JointPosition** erabiltzen dugu, zein bere barruan **JointValue** eta **Poison** motako elementuak ditu. JointValue mota robotaren elementuei balioak emateko erabil daiteke, kasu honetan besoen ardatzak. Ardatz bakoitzari izen bat esleitu behar zaio (joint_uri) eta balioak zein unitatean adierazten diren definitu, gainera denbora markak esleitu daitezke (timeStamp). Horretaz gain, Poison mota mezuaren informazio gehiago ematen digu.

Liburutegi hau erabiltzea robotaren kontroladorearekin zetorren aplikazioei erreparatuta erabaki da. Izan ere, besoa mugitzen hasi baino lehen aplikazio horiek garatu den aplikazioaren adibide bezala erabiltzeko aztertu eta probak egin izan ziren.

#### 4.2.5.3. nav_msgs

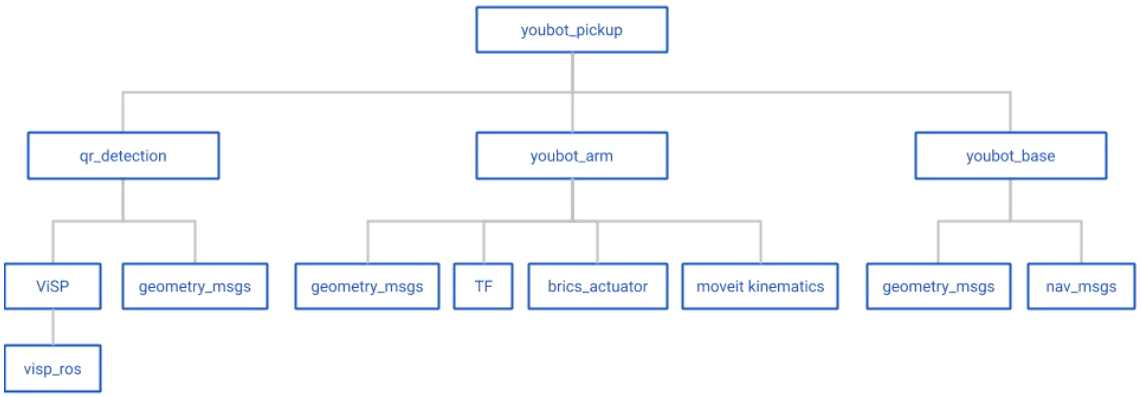
Mezu hauek ere ROS-en arrunten barruan daude eta nabigaziorako erabiltzen diren mezuak dira, hala nola, odometria mezuak, egingo den ibilbidea markatzeko mezuak, etab. Gure kasuan, odometria (**Odometry**) mezuak bakarrik erabili ditugu (ikusi *Oinarriaren mugimendua*) oinarriaren posea eta abiadura estimatzeko. Mezu hauek buruko bat dute eta horretaz gain

elementu semearen identifikazioa eta posea eta abiadura adierazten duten geometry_msgs mezu motak (Pose eta Twist). Baino, azken bi hauei, kobariantza gehitzen zaio 6x6 dimentsiotako matrize baten moduan, non hiru dimentsiotako balioak adierazten diren (x, y, z) baino baita ere ardatz bakoitzarekiko errotazioa.

Ikusten denez odometria mezu hauek TF-rekin zerikusia dute, izan ere, TF zuhaitzean odometria da erroa (ikusi 4.4. eskema) eta deskribatu dugun Odometry mezuetan honen semea adierazi behar da. Hau honela izanik, eta kontuan izanda geometry_msgs-ko Pose mezuak TF mezuetara bilakatu daitezkeela, mezu mota hau erabili da.

## 4.3. Moduluak

Kodea hiru moduluetan banatu da garapena errazteko eta hauek etorkizuneko garapenetan erabiltzeko aukera emateko helburuarekin. 4.5. eskeman ikusi daitekeenez, modulu hauen artean ez dago dependentziarik, beraz, bakarka erabili daitezke, aplikazioaren araberako erabilpena emanik. (Ikusi A. eranskina proiektu honetan garatu den kodearen berrerabilpenari buruzko argibide gehiagorako).




4.5. Eskema. Modulu ezberdinen dependentzia grafoa.

Modulu hauetako bakoitza C++ klase bat da, honela programazio lengoaia honen objektuei orientatutako aldea esplotatuz, aplikazioaren oinarri modularrari indarra ematen lagunduz eta berrerabilpenari indarra emanik.

### 4.3.1. QR detekzioa

Modulu honek QR detekzioa egiten du aurreko atalean azaldutako ViSP liburutegia erabiliz. Moduluak objektu bat du bere barruan eta hau sortzerakoan kameraren informazioa eta datuak (irudia) zein *topic*-etik etorriko den ezartzen du. Hau egiteko, 4.6. eskeman ikusten den bezala, ViSP eta ROS bitartekari bezala funtzionatzen duen ViSP-en modulua erabiltzen da (*visp_ros*).


4.6. Eskema. QR detekzioan liburutegien funtzionamendua.

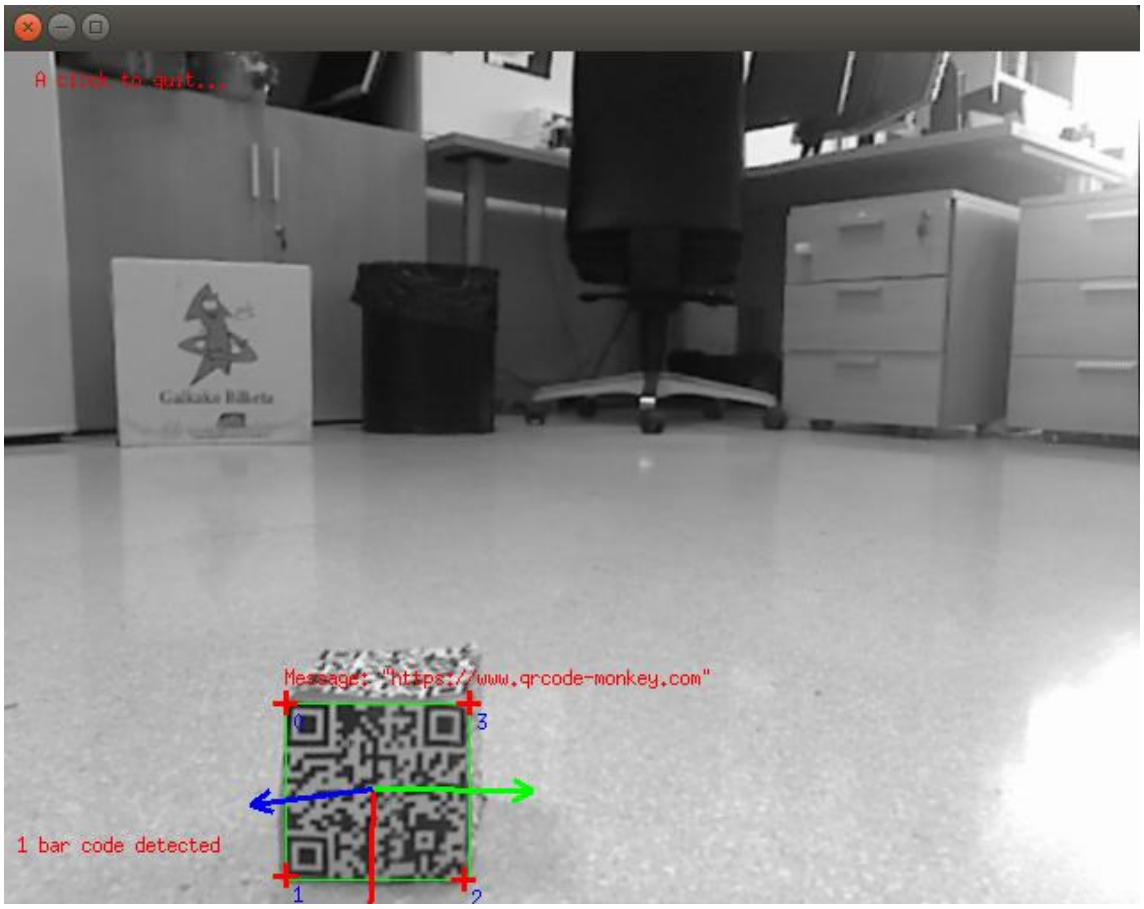
Modulu honetaz aparte, 4.3. eskeman ikusten diren moduluetatik *detection* modulua erabiltzen da QR kodea detektatzeko eta *vision* modulua (objektuaren) posizioa kalkulatzeko. Azken hau egiteko, kodearen eredu bat eman behar zaio funtzioari, izan ere, QR kodea eta kameraren arteko distantzia determinatzeko, kameraren irudiaren tamaina erreferentzia

moduan erabiliz QR kodearen tamaina kalkulatzen da, kodea okupatzen duen pixel kopuruaren arabera. Ereduan definitzen den QR kodearen tamaina hori erabiliz eta irudian okupatzen duena erabiliz distantzia kalkulatzen da. Beraz, bi eragiketa egiten dira: pixel kopuruak metro unitatetara pasa eta ereduan agertzen den tamaina irudian detektatutakoarekiko distantzia kalkulatzea. Eredu hau ondo definituta egotea oso garrantzitsua da beraz. 4.1. programan ikusi daiteke eredu hau puntuen lista bat dela, non hiru dimentsioko koordenatuak ematen diren eta, koordenatu horiek, kodearen erdiko puntutik aldeetara dagoen distantzia adierazten dutela.

```
point.push_back(vpPoint(-0.03, -0.03, 0));
point.push_back(vpPoint(0.03, -0.03, 0));
point.push_back(vpPoint(0.03, 0.03, 0));
point.push_back(vpPoint(-0.03, 0.03, 0));
```

4.1. Programa. QR kodearen hiru dimentsiorako eredua.

Azkenean, QR kodea detektatzen bada 4.4 irudian agertzen den antzekoa irudikatuko zaigu, non QR kodea lauki batean sartuta egongo den, bere ertzak markatu eta zenbaturik eta azkenik QR posea hiru dimentsioak markatzen dituzten gezi batzuekin irudikatuko da. Irudian ere ikusi daiteke, nahi izango bagenu QR kodearen mezua eskuratu dezakegula eta, nahiz eta aplikazio honetarako ez dugun erabili, etorkizuneko aplikazioentzat interesgarria izan daiteke edota beste objektu mota izango bagenu, bata bestearekin desberdin ahala izateko modu erraz bat izan litekeen.



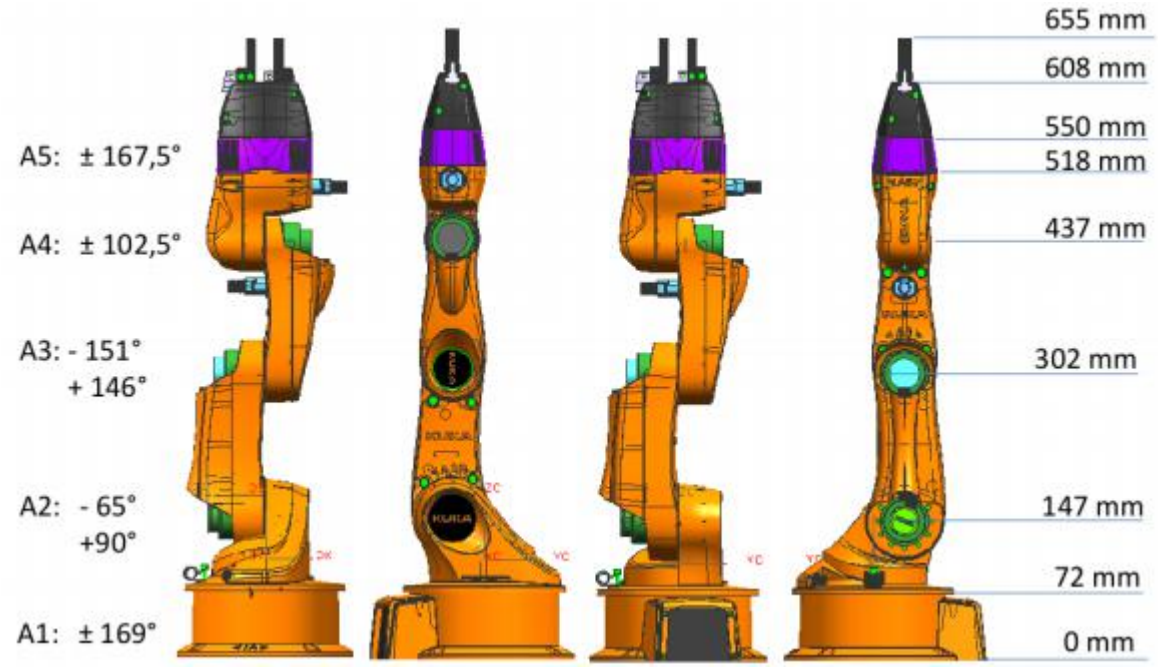

4.4. Irudia. QR detekzioa.

### 4.3.2. Beso mugimendua

Besoa mugitzeko beste modulu hau garatu da. Moduluaren barnean lau posizio aurredefinitu dira: objektua hartu baino lehen besoa honetara gerturatzeko posizioa, objektua hartzeko posizioa, objektua zama eramateko plataforman uzteko posizioa eta besoaren posizio erlaxatua; hauen balioak modulua hasieratzerakoan (klasearen objektua sortzean) ezartzen dira. Besoak pose desberdin hauek egiteko, moduluaren klasearen kanpotik deitu daitezken funtzioak inplementatu dira, honela kodearen berrerabilpena ere bultzatuz.

Besoaren posizio (edo pose) hauek nola lortu zehazki A. eranskinean aurkitu daiteke baina hemen gainetik azalduko dugu. Hasteko, besoaren ezaugarri bakoitza ezagutu behar ditugu (hauek 4.5. irudian ikusi daitezke), bai ardatz bakoitzak noraino biratu dezakeen baita besoa osatzen duten pieza bakoitzaren neurriak. Ardatz bakoitzak balio ezberdin jakin bat edukiko du posizio ezberdinetarako, baina horretaz gainera lehenengo ardatzetik azkenerainoko (TF) koordenatuak beharko ditugu ardatzaren balio hori besoaren pose batera bihurtzeko. Gero, behin baino gehiagoetan aipatu den alderantzizko zinematiketan (ikusi *Youbot manipulation* liburutegia) erabiliko du informazio hau besoaren posizio batetik bestera joateko ibilbidea kalkulatzeko.

Behin hori kalkulatuta dugunean mezuen bidez (ikusi *ROS mezuak*) besoa eta honen bukaeran dagoen pintza mugitu dezakegu. Inplementatu den moduan, pintzaren bi elementuak batera eta distantzia berdina mugitzen dira, baino badago bakoitza independenteki mugitzeko aukera.




4.5. Irudia. Kuka youBot robotaren besoaren neurriak eta besoaren ardatz bakoitzaren biraketa tarteak[2].

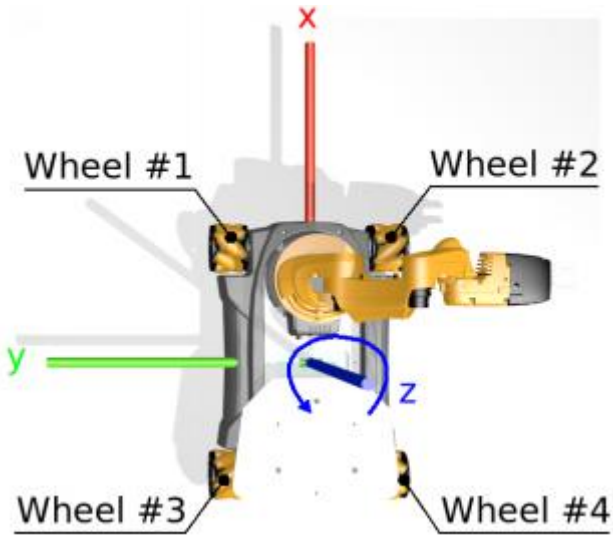
### 4.3.3. Oinarriaren mugimendua

Objektua ondo hartzeko helburuarekin oinarria mugitzeko modulua garatu da. Modulua hasieran orokorragoa egin nahi izan zen: koordenatu (pose) bat emanda, nabigazio paketea erabiliz honetara hurbiltzeko. Hala ere, azaldu den bezala, mapa erabiltzen zuen nabigazio pakete hori ez da lortu ondo konfiguratzea baino etorkizuneko aplikazioetan erabil daitezkeelako kodean[23] mantendu da iruzkinduta. Mapan oinarrituta zegoen nabigazio honek, mapak sortzerako

[2] KUKA youBot User Manual eskuliburutik aterata, 2012ko abenduaren 6an.

momentuan arazoak ematen zituen: mapan agertzen ziren distantziak eta norabideak ez ziren zuzenak eta, nahiz eta ondo eratutako mapa bat eman, puntu batera joateko esaten zenean, robota bere buruari bueltan gelditzen zen, mapan kokatzen saiatuz. Hau konpontzeko, ROS-en wiki-an agertzen diren azalpenak jarraituz, nabigazio paketea berriro instalatu eta konfiguratzea izan daiteke modua. Lan honetan ezin izan da egin denbora faltagatik eta ez zelako lan honen helburuetako bat.

Azkenean garatu den soluzioa zuzenean robotaren gurpilei abiadura bat esleituz mugitzea izan da. Horregatik oso garrantzitsua da 4.6. irudian agertzen diren koordenatu ardatzak, jakiteko, adibidez, aurrera edo atzera joateko *x* ardatzean mugitu behar garela eta biratu nahi dugunean *z* ardatzean angelua mugituko dugula.




4.6. Irudia. Kuka youBot robotaren oinarriaren goiko ikuspegia.[3]

Honetaz gain, helburu posizio bat beharko dugu, gure kasuan objektuaren posizioa robotarekiko izango dena, eta robotaren egungoko egoera nabigazio mezuetako odometriak emango diguna. Odometria deitzen zaio nabigazioaren zehar gurpildun ibilgailuen posizio estimazioari eta robotikan oso erabilia da bere inplementazio eta kalkulu kostu txikiengatik.

Hau guztia dugunean, robota eta objektuaren distantziaren arabera mugituko da robotaren oinarria, objektua hartzeko besoaren posizio aurredefinitua kontuan hartuz. Azken hau kalibratzea izan da denbora gehien behar izan duen partea.

## 4.4. Fitxategi jaurtitzailea

Oraindik ez dugu azaldu eta ROS-en tresna erabilgarrienetakoena dira fitxategi jaurtitzaileak. Fitxategi hauek, behar ditugun liburutegi, kontroladore, aplikazio edo beste fitxategi jaurtitzaileak exekutatzeko ahalmena ematen digu terminalean lerro bakar bat idatziz. Gainera aplikazioan behar diren parametroei balio lehenetsiak zehazteko aukera ere ematen digu, gero exekutatzerakoan aldatu daitezkeenak.

Hala ere, gure kasuan, garatutako aplikazioa ez da fitxategi honetatik exekutatzen, baino exekutatzea posible izango zen; honela egitearen arrazoia berrerabilgarritasun kontua da: etorkizunean beste aplikazioak garatzen direnean, fitxategi hau erabilgarri izan daiteke aplikazio horiek erabiltzen dituzten baliabideak exekutatzeko. Exekutatzen diren baliabideak eta nola 4.2. programan ikusi daitezke, baita fitxategi hauen XML moduko sintaxia. Lehenengo, esan bezala,

[3] KUKA youBot User Manual eskuliburutik aterata, 2012ko abenduaren 6an.

robotaren kontroladorearen fitxategi jaurtitzailea deitu egiten da honek kontroladorea exekutatu dezan, gero TF zuhaitzean elementu bat gehitzeko agindua ematen da, oinarria eta kameraren erlazioarekin eta, azkenik, Kinect dispositiboaren kontroladorearen fitxategi jaurtitzailea kontroladorea exekutatu dezan.

```
<?xml version="1.0"?>
<launch>
   <!-- Kuka youBot drivers -->
   <include file="$(find youbot_driver_ros_interface)/launch/youbot_driver.launch"/>

   <!-- Broadcast static tf for robot root and kinect -->
   <node pkg="tf" type="static_transform_publisher"
name="virtual_joint_broadcaster_0" args="0.28 0 0.07 0 0 0 base_link camera_link 100"
respawn="true"/>

   <!-- Kinect's drivers -->
   <include file="$(find freenect_launch)/launch/freenect_throttle.launch">
       <arg name="rate" value="5"/>
   </include>
</launch>
```

4.2. Programa. *youbot.launch* fitxategi jaurtitzailea.

Fitxategi honetaz gain, beste fitxategi batzuk daude aplikazioaren paketearen barruan, hurrengo garatzaileei interesgarri izan daitezkeenak eta A. eranskinean azalduta daudenak.

## 4.5. Aurkitutako arazoak eta emandako soluzioak

Lan hau egiten hastean, erabilitako robota hutsegiteak eta falta lana garatzeko tresnak falta zitzaion. Hasteko, robota ezin zen korronte elektrikotik deskonektatu eta bateriarekin ibili 5 minutu baino gehiago. Arazo hau zuzendariei luzatu zitzaion eta hauek bateria berria erosi izan behar zuten. Robotak 24V-ko korrontea behar du funtzionatzeko eta bateria robotaren oinarrian dagoen hutsunean finkatzen da, beraz, bateria egokiarekin ematea ez da hain erraza. Azkenean, bateriaren kontua zen eta erositako berria ipintzean ordu bat ordu eta erdi artean iraun dezake arazorik gabe aplikazio hau exekutatzen.

Aurkitu zen beste hutsa zen ez zeudela robotaren manualean agertzen ziren kontroladore eta API guztiak instalatuta eta beraz, hauek instalatu behar izan ziren batez ere bestela ez geneukalako besoa mugitzeko tresnarik. Tresna hauek instalatu eta konfiguratzea denbora dezente eskatzen zuten.

Hasieran, asko lagundu nintzen robotean aurreko ikasleak egindako konfigurazioetan, berak erabilitako tresnak aztertu nituen, berak egindakoa ulertu nahian. Baino, azkenean konturatu nintzen bere konfigurazio horiek sinplifikatu litezkeela eta hau programen abiadurarentzat onura bat izan litezkeela, tresna bakoitzerako terminal ezberdinak aldi berean exekutatu ordez, tresna guztiak komando bakarrean aldi berean exekutatuz. Honetarako fitxategi jaurtitzaile berriak (adibidea, 4.2 programa) idatzi nituen, mota honetako fitxategiak nola idatzi ikasiz.

Orain arte bakarrik hasieran izandako arazoak kontatu dira baino aplikazioa garatzen hasterakoan arazo gehienak etorri ziren. Hasteko, ez geneukan ziur zein aplikazio mota garatu eta aukera ezberdinak pentsatu eta probatu ziren. Aukera horietako batzuk alde batera utzi izan behar ziren, nahiz eta batzuk garatzen hasita zeuden ere. Hala ere, besoarekin zerbait egitea zenez lanaren helburua, horrekin hasi ginen eta honeta ere arazoak egon ziren. Besoa mugitzeko erabiltzen diren pose aurredefinitu horiek eskuratzeko momentua, hain zuzen ere. Hau konpontzeko A. eranskinean agertzen den metodoa aurkitu zen, azterketa eta proba ezberdinen ondoren.

Gero, QR kodea detektatzeko aukera ezberdinak aztertu ondoren, ViSP aukeratu zen, baina aplikazioan integratu baino lehen proba ezberdinak egin ziren. Hasieran, QR kodeak 4,5cm-koak ziren (4,5cm-ko tamaina zuen ere kuboaren alde bakoitza) baina proba ugari ondoren, determinatu zen tamaina horrekin ez zela normalean ondo detektatzen QR kodeak, batez ere kuboa lurrean bazegoen. Gure proposamena lurrean zeuden objektuak hartzea zenez, beste tamainako kuboekin probatu zen, horretarako pintzaren zabaltze tamaina kontuan izanik. Honela, 6cm-ko kuboak erabiltzera pasa zen eta proba ezberdinak egin ondoren, hau da aplikazioan erabiltzen den QR kodearen tamaina.

Azkenik, aipatu den aurreko ikaslea egindako nabigazio aplikazioa, aplikazio honetara integratzea saiatu zen, baina, esan den bezala, honekin arazoak eduki genuen ere. Arazo horiek mapa sortzerakoan zein mapa hori erabiltzerako momentuan agertzen ziren. Azkenik, arazo oso zabala zenez eta beste GRAL (Gradu Amaierako Lana) bat izateko moduko lana zenez, beste metodo bat bilatu zen. Hau da aurkezten den aplikazioan erabiltzen dena eta mapa bat erabili ordez, bakarrik objektuaren posizioa eta robotarekin distantzia erabiltzen da. Honek ez du arazoa guztiz konpontzen, izan ere, beste metodoa (maparekin) oztopoak galarazteko aukera ematen zigun eta azkenean egin den moduan hau ez da egiten.

# 5. Ondorioak

Lan honen hasieran robotika asistentzialaren eremua eta horren inguruko ikerketa ezberdinak tratatu dira, adindunen eremuan zentratuz. Ikerketa horiek kontuan izanik mugikortasun arazoak dituzten pertsonentzat aplikazio bat garatu da eta dokumentu honen bitartez azaldu izan nahi da. Egin dugun aplikazio honekin demostratu nahi izan da mota honetako robotak ere robotika asistentzialean erabili daitezkeela. Nahiz eta oraindik asko dagoen egiteko, proiektu honetan garatutakoa helburu horretara gerturatzeko lehenengo pausuak eman direla uste dugu.

Hasieratik esan da robot hau ez dela produktu final bezala egokia, adibidez arrazoi ekonomikoengatik, baino baita ere kontuan izan behar da mota honetako robotek ezin dutela pertsonekin edo pertsonen inguruan lan egin, ez bazaio segurtasun neurri batzuk inplementatzen behitzat. Egin daitezken beste hobekuntzak dira pertsonekiko interakzioa hobetzea, sozialago eginez edota erabiltzaile interfazeak hobetzea, adibidez, web edo mugikorrerako aplikazio bat eginik robota (eta bere funtzionalitateak) kontrolatzeko. Honetaz gain, objektuak detektatzeko beste moduak esploratu daitezke irudien tratamendua eginez forma anitzeko objektuak detektatzeko[14] edo *machine learning* erabiliz ereduak eraiki objektuak detektatzeko edota QR kodeak erabili ordez beste aukera batzuk[16] erabili. Izan ere, QR kodea detektatzerako orduan argitze arazoengatik batzuetan ez du detektatzen edo asko tardatzen du detektatzen. Kinect-a erabiliz egin daiteken beste gauza batzuk dira aurpegi eta/edo gorputzen detekzioa honela pertsonak segitu ahal izateko eta, adibidez, pertsona bat lurrean aurkitzean abisu bat eman zaintzaile edo familiari. Gainera, lehenago aipatu da nabigazioa egiteko ez dela egin modu konbentzional batean, hau da, ingurunearen mapa bat eginez eta oztopoak detektatuz, ezin izan delako aplikazioarekin integratu. Beraz, oinarriaren nabigazioa asko hobetu daiteke, bai dagoena hobetuz zehaztasun gehiago emanez, bai beste metodoren bat erabiliz. Nabigazioa beste fase aurreratu batera eraman daiteke eta ibilbide aurredefinituak zehaztu adibidez zaharren egoitzetan erroldak egiteko edo etxean dena ondo dagoen detektatu eta modu kontsekuentean jokatu. Beste aldetik, nabigazioa erabiltzea behartzen digu programa urrutiko moduan exekutatzea eta honek, adibidez irudiekin latentzia asko sortzen ditu.

Lan honen garapenaren bitartez gauza asko ikasi direla uste dugu, gehienbat pertsonalki baino baita ere robotika asistentzialean erabilgarri izan daitezkeenak. Pertsonalki modu autonomoago batean lan eta ikasi egin da eta hau lan mundura gerturatu didala uste dut. Beste aldetik, robot erreal batekin lan egitean robotikari buruz asko ikasteko aukera eman dit, robot bat konfiguratzetik, ROS-ekin programatzera. Prozesu honetan dokumentu honetan agertu diren terminoak ikasi dira, hala nola, robot nabigazioa, alderantzizko zinematikak, etab. eta baita ere horiek nola garatzen diren ikasi da. Robotikaren aldetik hurrengo proiektuentzat oinarri batzuk jarri ditugula uste dugu baita lan egin den robotaren potentziala erakutsi ere, guzti honekin garatzaileei lerro honetatik jarraitzea animatzeko.

Orokorrean, nahiz eta aipatu diren hobekuntza guzti horiek eta seguruenik gehiago dauden egiteko, egindako lanarekin gustura gelditu gara, kontuan izanik hasieran besoa mugitzea bakarrik zela gure helburua eta azkeneko emaitza honetaz haratago doan emaitza izanik, zerbait funtzionala aurkeztuz.

# Bibliografia


[1] Azimi, I., Rahmani, A. M., Liljeberg, P., & Tenhunen, H. (2016, 06). Internet of things for remote elderly monitoring: A study from user-centered perspective. *Journal of Ambient Intelligence and Humanized Computing, 8*(2), 273-289. doi:10.1007/s12652-016-0387-y

[2] Bajones, M., Fischinger, D., Weiss, A., Wolf, D., Vincze, M., Puente, P. D., . . . Frennert, S. (2018, 06). Hobbit: Providing Fall Detection and Prevention for the Elderly in the Real World. *Journal of Robotics, 2018*, 1-20. doi:10.1155/2018/1754657

[3] "The C Resources Network." *Cplusplus.com*, www.cplusplus.com/.

[4] Cesta, A., Cortellessa, G., Rasconi, R., Pecora, F., Scopelliti, M., & Tiberio, L. (2011, 02). Monitoring Elderly People With The Robocare Domestic Environment: Interaction Synthesis And User Evaluation. *Computational Intelligence, 27*(1), 60-82. doi:10.1111/j.1467-8640.2010.00372.x

[5] E. Pollack, Martha & Brown, Laura & Colbry, Dirk & Orosz, Cheryl & Peintner, Bart & Ramakrishnan, Sailesh & Engberg, Sandra & Matthews, Judith & Dunbar-Jacob, Jacqueline & E. Mccarthy, Colleen & Montemerlo, Michael & Pineau, Joelle & Roy, Nicholas. (2002). Pearl: A Mobile Robotic Assistant for the Elderly.

[6] KUKA youBot User Manual. (2012, 12). Locomotec.

[7] Kaneko, S., & Capi, G. (2014). Human-robot Communication for Surveillance of Elderly People in Remote Distance. *IERI Procedia, 10*, 92-97. doi:10.1016/j.ieri.2014.09.096

[8] Kaneko, S., & Capi, G. (2014). Human-robot Communication for Surveillance of Elderly People in Remote Distance. *IERI Procedia, 10*, 92-97. doi:10.1016/j.ieri.2014.09.096

[9] Lazar, A., Thompson, H. J., Piper, A. M., & Demiris, G. (2016). Rethinking the Design of Robotic Pets for Older Adults. *Proceedings of the 2016 ACM Conference on Designing Interactive Systems - DIS '16*. doi:10.1145/2901790.2901811

[10] Smarr, Cory-Ann & Fausset, Cara & Rogers, Wendy. (2011). Understanding the potential for robot assistance for older adults in the home environment.

[11] *ViSP*, visp.inria.fr/.

[12] "Main Page." *YouBot Wiki*, www.youbot-store.com/wiki/index.php/Main_Page.

[13] "Main Page/Es." *OpenKinect*, openkinect.org/wiki/Main_Page/es.

[14] Malmsten, Paul Oesterle. "Object Discovery with a Microsoft Kinect." *Digital WPI*, digitalcommons.wpi.edu/mqp-all/913/.

[15] "Moving Robots into the Future." *Brand,* moveit.ros.org/.

[16] Neosistec, Mrvlab &. "The Cutting Edge Technology for the Visually Impaired." *NaviLens EMPOWERING the Visually Impaired*, www.navilens.com/.

[17] "Wiki." *Ros.org*, wiki.ros.org/.

[18] "[ROS Q&A] 053 - How to Move a Robot to a Certain Point Using Twist." *The Construct*, 17 Sept. 2018, www.theconstructsim.com/ros-qa-053-how-to-move-a-robot-to-a-certain-point-using-twist.

[19] Ros-Drivers. "Ros-Drivers/freenect_stack." *GitHub*, 22 Nov. 2017, github.com/ros-drivers/freenect_stack.

[20] Sharma, Shashank, and Christian Scheurer. "Generalized Unified Closed Form Inverse Kinematics for Mobile Manipulators With Reusable Redundancy Parameters." *Volume*

*5B: 41st Mechanisms and Robotics Conference*, 2017, doi:10.1115/detc2017-68104.

[21] Svenschneider. “Svenschneider/Youbot-Manipulation.” *GitHub*, 28 Nov. 2016, github.com/svenschneider/youbot-manipulation.

[22] Wnowak. “Wnowak/brics_actuator.” *GitHub*, 16 Oct. 2015, github.com/wnowak/brics_actuator.

[23] aps54. “aps54/youbot_pickup.” *GitHub*, 10 June 2019, github.com/aps54/youbot_pickup.

# Manual de usuario

Kuka youBot

Laboratorio de Interacción Persona-Computador para Necesidades Especiales
Euskal Herriko Unibertsitatea
Universidad del País Vasco
(EHU/UPV)

Manual de usuario para el manejo del robot Kuka YouBot.

Ainhoa Pato

# Índice

# Introducción

Este manual de usuario se ha escrito pensando en los nuevos desarrolladores que trabajarán con el robot Kuka youBot para desarrollar nuevas aplicaciones y herramientas en el futuro. El objetivo del documento es intentar hacer más fácil los primeros pasos de estas personas y dotarlas con herramientas que ya se han explorado o desarrollado anteriormente que se creen son interesantes y de utilidad, tanto como para construir a partir de ellas o para usarlas en momentos puntuales del desarrollo.

Por lo tanto, se ruega a los siguientes desarrolladores que trabajen con el robot, sigan desarrollando, extendiendo y mejorando este manual en nuevas versiones, siempre haciendo mención en el apartado de autor(es) o citando a la bibliografía o referencias a las personas que hayan tomado parte en versiones anteriores.

Espero que este documento te sea de ayuda y te animes a seguir escribiendo, ayudando así de igual manera a los siguientes compañeros.

# Primeros pasos

## Encendido y apagado del robot Kuka youBot

Enchufado el robot tanto con batería o por cable de alimentación directamente, debería aparecer en la pantalla LCD, aunque sin luz, el mensaje "System standby". Si no aparece, el robot no está bien enchufado.

Pulsar una vez el botón ON/OFF hasta que la pantalla LCD se encienda con luz naranja. Saldrá un mensaje de "System startup".

Pulsar de nuevo el botón ON/OFF hasta que aparezca el mensaje en la pantalla LCD de "PC On", soltar inmediatamente. Oiremos cómo se encienden los ventiladores del PC.

Pulsar una tercera vez el botón ON/OFF para encender los motores haciendo parecido que la vez anterior pero esperando a que aparezca "Motor On". Si vamos a usar el brazo, en la parte posterior de éste, donde están los puertos EtherCAT y de alimentación, veremos un botón. Si está encendido en rojo pulsar y soltar para encender los motores del brazo. Se pondrá en verde.

Si después se quieren apagar únicamente los motores del brazo se puede volver a pulsar el mismo botón hasta que sea rojo de nuevo.

Para apagar todos los motores pero poder seguir trabajando con el PC del robot habrá que pulsar de nuevo el botón ON/OFF y aguantar hasta que en la pantalla LCD salga el mensaje "Motor Off". Notarás que con esto el botón del brazo también se apagará, no tendrá ningún color.

Si se quiere apagar el PC del robot pulsar el botón ON/OFF hasta que salga el mensaje "PC off" en la pantalla.

Finalmente, para apagar del todo el robot, dejar pulsado el botón ON/OFF hasta que aparezca "Switch Off" en la pantalla LCD. Con esto debería apagarse también la luz de ésta y aparecer el mensaje del principio ("System standby").

Si tenemos dudas de lo que tenemos encendido/apagado en cualquier momento, además de las señales ya mencionadas (luces, sonidos, etc.), nos podemos fijar en la pantalla LCD. Arriba a la derecha tiene 4 cuadrados. El que está más a la derecha nos dirá si los motores están encendidos o apagados. Si el cuadrado es grande, significa que están encendidos, sin embargo, si el cuadrado es pequeño, los motores están apagados.
El cuadrado que está a la izquierda de éste nos dirá, de igual manera, si el PC del robot está encendido o apagado. Los otros dos cuadrados restantes conciernen a la batería: si está enchufada, si está cargándose o cargada.

# Iniciar sesión

Para iniciar sesión tenemos dos opciones: conectarnos remotamente via *ssh* o directamente iniciar sesión en el PC interno del robot.

Tanto si vamos a utilizar el PC del robot directamente o vamos a conectarnos remotamente los datos de inicio de sesión son los siguientes:

**Usuario:** `youbot`
**Contraseña:** `youbot`

## PC interno del robot

Simplemente conectamos un cable VGA de un monitor en el lado de la base del robot con el conector pertinente y un ratón y teclado en los conectores USB al lado de éste. Se recomienda esta opción al principio o si se van a hacer pruebas en las que el robot no vaya a moverse del sitio.

## Conexión remota

Como ya se ha mencionado, se hace mediante *ssh* por lo que es imprescindible tenerlo instalado en el ordenador que se vaya a usar para conectarse, aunque normalmente viene con el sistema operativo. Tendremos que escribir lo siguiente en una terminal:

```
ssh youbot@192.168.1.234
```

O si queremos usar alguna opción gráfica, es decir, que se nos abran ventanas, tendremos que escribir:

```
ssh -X youbot@192.168.1.234
```

### Conseguir la dirección IP

Puede que cuando te vayas a conectar remotamente te salte un error de que no se encuentra la dirección IP, esto puede ser porque se haya cambiado o simplemente porque el adaptador wifi del robot no está funcionando bien. En estos casos, hay una aplicación que muestra la IP del robot en la pantalla LCD. Tendrás que conectarte al PC del robot (ver el apartado anterior "*PC interno del robot*") y ejecutar en una terminal lo siguiente:

```
cd /usr/local/bin
sudo ./displayIpAddress /dev/ttyACM0 eth1 wlan4
```

Si el error no es por tener la dirección IP cambiada, prueba a hacer ping desde el PC remoto a la dirección IP que aparece en la pantalla. Si al ejecutar el comando anterior te da error puede ser que *wlan4* haya cambiado de nombre, prueba a mirar las interfaces existentes ejecutando *ifconfig* en el PC del robot.

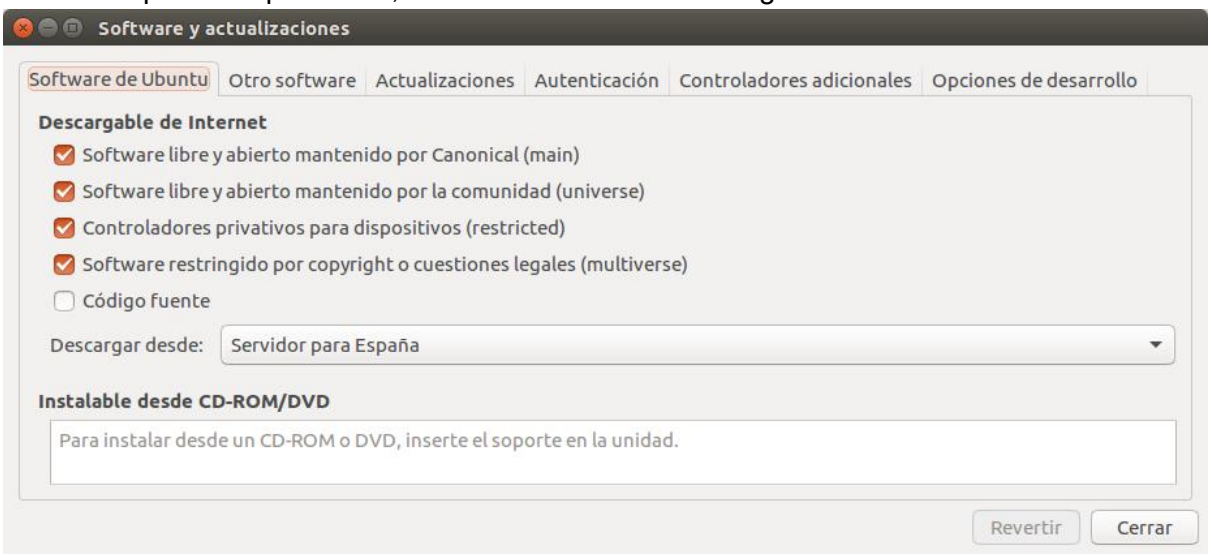
# ROS

## Instalación de ROS en PC remoto

Se recomienda usar la distribución Kinetic de ROS disponible para Ubuntu Willy (15.10), Ubuntu Xenial (16.04 LTS) y Debian Jessie (8.0). Si se tiene instalado unas versiones posteriores a éstas ROS recomienda usar la distribución Melodic. Sin embargo no se ha probado el trabajo realizado en esa distribución y, por lo tanto, no se garantiza si funcionará o no.

### Ubuntu

Primero, comprueba que en tu configuración sobre repositorios están permitidos todos los tipos de repositorios, como se muestra en la imagen.




Una vez hecho esto, configurar tu ordenador para aceptar software de packages.ros.org:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main"
> /etc/apt/sources.list.d/ros-latest.list'
```

Por último, consigue tus claves:

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key
421C365BD9FF1F717815A3895523BAEEB01FA116
```

Ya estamos listos para instalar ROS, pero primero que no se te olvide:

```
sudo apt-get update
```

Hay distintas versiones de ROS, depende de qué vayas a usar y las herramientas que necesites. Nosotros vamos a instalar la versión *Escritorio* que, además de ROS, trae librerías genéricas para robots, rviz y rqt. Para eso ejecutar:

```
sudo apt-get install ros-kinetic-desktop
```

### Debian

Primero debes de configurar tus repositorios Debian para que acepten "contrib" y "non-free".
Después tendrás que configurar *source.list* añadiéndole una línea para que acepte los repositorios de packages.ros.org. Para eso puedes ejecutar la siguiente línea:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main"
> /etc/apt/sources.list.d/ros-latest.list'
```

Una vez hecho esto, antes de instalar ROS tenemos que conseguir las claves para el repositorio.

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key
421C365BD9FF1F717815A3895523BAEEB01FA116
```

Para conseguir la última versión actualizada ejecutar

```
sudo apt-get update
```

Hay distintas versiones de instalador, depende de qué vayas a usar y las herramientas que necesites. Nosotros vamos a instalar la versión *Escritorio* que, además de ROS, trae librerías genéricas para robots, rviz y rqt. Para eso ejecutar:

```
sudo apt-get install ros-kinetic-desktop
```

## Creación y configuración del entorno de trabajo

### Crear workspace (*catkin*)

Se recomienda crear el entorno en una carpeta dentro del directorio *home (~)* de cada usuario. Se puede tener más de un entorno de trabajo, pero por simplificar y para evitar problemas de dependencias es mejor tener uno solo. Se puede crear de la siguiente manera:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
```

Este último comando se usa para compilar los proyectos dentro del entorno pero, en este caso, lo usamos para crear las carpetas que se necesitan para que todo funcione. Al finalizar, la estructura del directorio debería ser la siguiente:

```
~
↳    /catkin_ws
    ↳   /build
    ↳   /devel
    ↳   /install
    ↳   /src
```

## Configuración del entorno de trabajo

En la carpeta *home (~)* del usuario debería de haber un archivo *.bashrc* o *.zshrc.* Edítalo con el editor de texto que quieras y añade lo siguiente:

```
source /opt/ros/kinetic/setup.bash

source /home/user/catkin_ws/devel/setup.bash

# Establecer al PC una dirección IP fija
export ROS_IP=192.168.1.223
# Dirección del Master PC (robot)
export ROS_MASTER_URI=http://192.168.1.217:11311

# Establecer dirección del entorno
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH
```

En la primera línea tendrás que cambiar *kinetic* por el nombre de la distribución de ROS que tengas, si es que tienes otra diferente (como melodic, por ejemplo).
En la segunda línea tendrás que cambiar *user* por el nombre de tu usuario. Esta línea es para no tener que estar referenciando la carpeta en cada terminal que abramos, ya que si no se referencia no ejecutará ningún proyecto.
Las dos siguientes líneas son necesarias para conectar el PC con el PC del robot por WIFI en una red privada.
Por último, la última línea es para que ROS sepa que carpetas tiene que compilar al usar *catkin_make*. Si tienes más de un entorno tendrás que separarlos con ":" (dos puntos) y sin dejar espacios (por ejemplo: *`export ROS_PACKAGE_PATH=~/ros_stacks:$ROS_PACKAGE_PATH`*).

## Inicializar el detector de dependencias (*rosdep*)

Antes de usar ROS, es recomendable que tengas inicializado el detector de dependencias del sistema que trae ROS y así evitar problemas. Ejecuta lo siguiente:

```
sudo rosdep init
rosdep update
```

## Otras herramientas necesarias

Puede que en algún momento necesites estas herramientas, sobretodo si descargas proyectos de otros desarrolladores (de github por ejemplo) que necesites compilar y que no están en la base de datos interna de *catkin*. Para evitar sorpresas y porque estas herramientas son muy utilizadas se recomienda descargarlas con la siguiente orden:

```
sudo apt install python-rosinstall python-rosinstall-generator python-wstool
build-essential
```

## Crear un proyecto o paquete

Un proyecto o un paquete de ROS sería cada aplicación que desarrolles en ROS dentro del espacio de trabajo. Éstos tienen que cumplir varios requisitos: siempre tiene que haber un fichero *package.xml* que tendrá la información para compilar la aplicación y otro fichero llamado *CMakeLists.txt* que usará *catkin* en la fase de construcción, además cada aplicación tiene que ir en una carpeta distinta dentro de la carpeta *src* del espacio de trabajo. Siendo así, esta sería la estructura más simple de un paquete ROS:

```
/src
↳    /nombre_paquete
    ↳   package.xml
    ↳   CMakeLists.txt
```

Es muy importante que estos archivos estén bien configurados respecto a lo que queremos compilar, incluir (dependencias) o instalar (librerías, plugins, archivos lanzadores, ….) porque sino nos puede originar muchos problemas y quebraderos de cabeza.

Además de esto, los paquetes típicamente tienen dentro otras carpetas para distintos propósitos, tales como *src* para el código de la aplicación, *launch* para ficheros lanzadores e *include* para los encabezados de C/C++ ya sean de ésta misma aplicación o de librerías que se usan dentro de la aplicación.

Esto puede ser un poco lioso o engorroso de hacer a mano, por lo que ROS trae una herramienta que te ayudará bastante a la hora de crear un paquete, creando los elementos indispensables mencionados, rellenando los archivos de compilación y creando algunas de las carpetas típicas mencionadas anteriormente. Así pues, si sabes que vas a crear un proyecto llamado *mypackage* que dependerá de las librerías (por ejemplo y siendo estas las más típicas) *std_msgs, rospy* y *roscpp,* ejecutando la siguiente línea

```
catkin_create_pkg mypackage std_msgs rospy roscpp
```

se creará una carpeta llamada *mypackage* que contendrá todo lo mencionado, donde el contenido de package.xml tendrá estas líneas que indican las dependencias

```
<package format="2">
...
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
...
</package>
```

y que podemos customizar a nuestro gusto añadiendo una breve descripción, el nombre del/la creador/a, la licencia, etc. También tendremos el fichero CMakeLists.txt relleno con, de momento, la información que le hemos añadido al crear el paquete y que seguramente haya que modificar si nuestro proyecto se hace más grande.

Una vez hecho esto se recomienda compilar el espacio de trabajo:

```
cd ~/catkin_ws/
catkin_make
```

Si quieres crear un paquete nuevo en el robot la carpeta donde debes ejecutar el comando para crearlo es `/home/youbot/youbot/ros/src` por lo que se recomienda navegar hasta ella (comando `cd`) antes de ejecutarlo. A la hora de compilar hay que ir al directorio `/home/youbot/youbot/ros/`.

# Aplicaciones

El robot tiene en sí varias aplicaciones que se han desarrollado o se han instalado de terceros, como las aplicaciones proporcionadas por el vendedor. En este apartado se detallarán algunas que se consideran interesantes, tanto por su carácter práctico a la hora de manejar el robot, como por ser una herramienta que se pueden aplicar y usar en siguientes aplicaciones y proyectos que se desarrollen.

## Control remoto con mando (*joypad*)

Entre las aplicaciones desarrolladas por los vendedores del robot encontramos esta útil herramienta. Nos sirve tanto para controlar a distancia la plataforma móvil como las articulaciones del brazo y las pinzas. Hay que tener en cuenta que esta aplicación no se puede utilizar mientras hay otra consola (*roscore*) u otra aplicación lanzándose (*roslaunch*). Para ejecutarla tendremos que tener un mando conectado (preferiblemente inalámbrico) y escribir lo siguiente en la terminal:

```
rosrun joypad_control youBot_JoypadControl
```

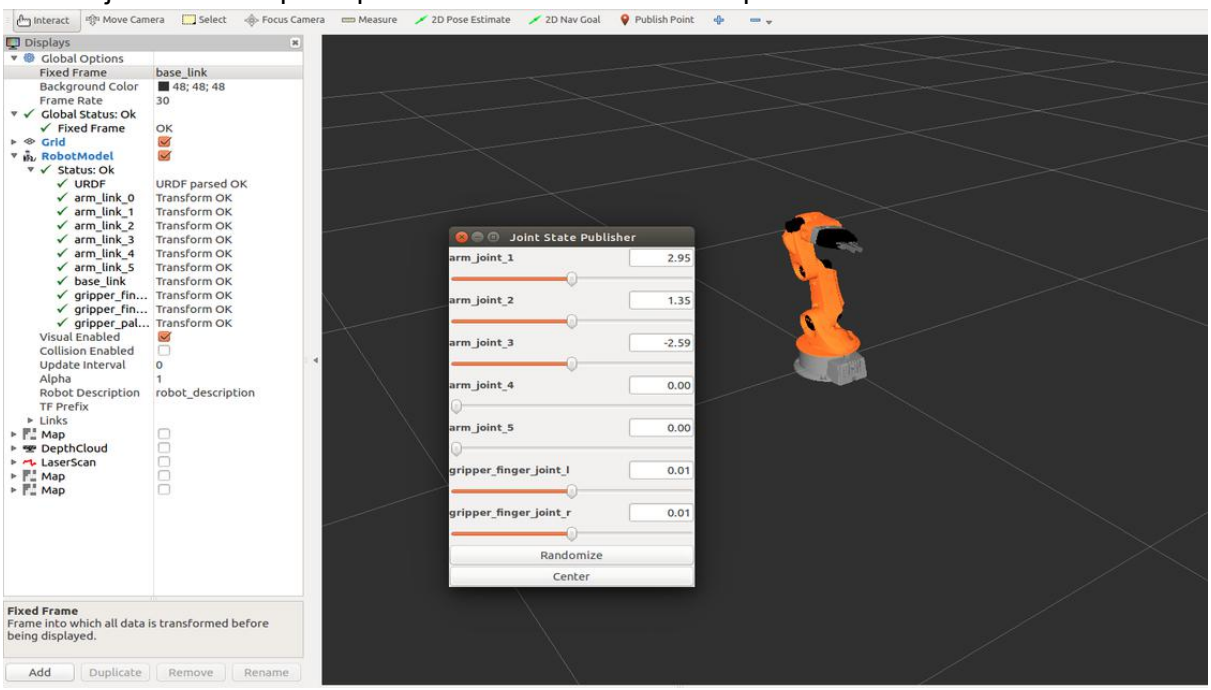
Los controles son los siguientes:



Control Remoto
por mando

## Movimiento del brazo con rviz dando valores

Esta aplicación que viene con los drivers del robot (dentro de *youbot_description*) te puede ayudar en algún punto a conseguir valores que hay que dar a cada uno de los ejes del brazo para que éste se encuentre en una posición determinada.




Puedes ejecutarla de esta manera:

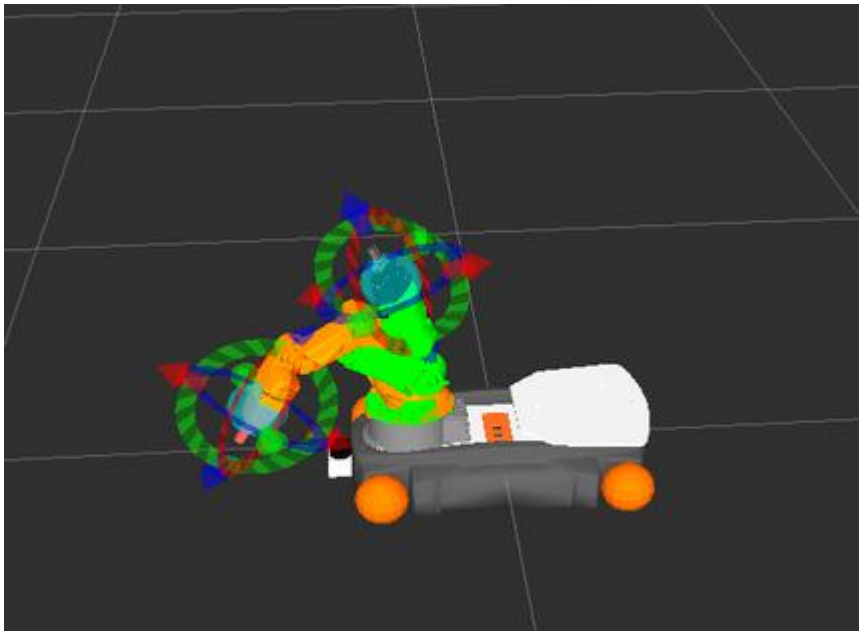
```
roslaunch youbot_description youbot_arm_simulation.launch
```

Ahora solo hace falta que vayas moviendo las flechas en las barras para cambiar de posición los ejes y conseguir la pose deseada. Si quieres volver a la posición inicial dale al botón "Center" y si quieres obtener una posición aleatoria puedes darle al botón "Randomize".

Como habrás notado, esta aplicación no mueve realmente el brazo, sino la visualización del brazo en rviz. Sin embargo, en la siguiente aplicación que se va a mostrar sí que se puede mover el robot real.

## Movimiento del brazo con rviz (Moveit!)

La siguiente aplicación sirve para algo parecido que la anterior. Sin embargo, tienen una diferencia muy importante y es que ésta aplicación te muestra la animación desde una pose a otra, es decir, te calcula los movimientos que va a hacer el brazo para llegar de la pose A a la pose B. A esto se le suele llamar *Inverse Kinematics* (IK), Cinemáticas Inversas en castellano y pueden ser usadas como parte de otro proyecto en el que se quiera mover el brazo.



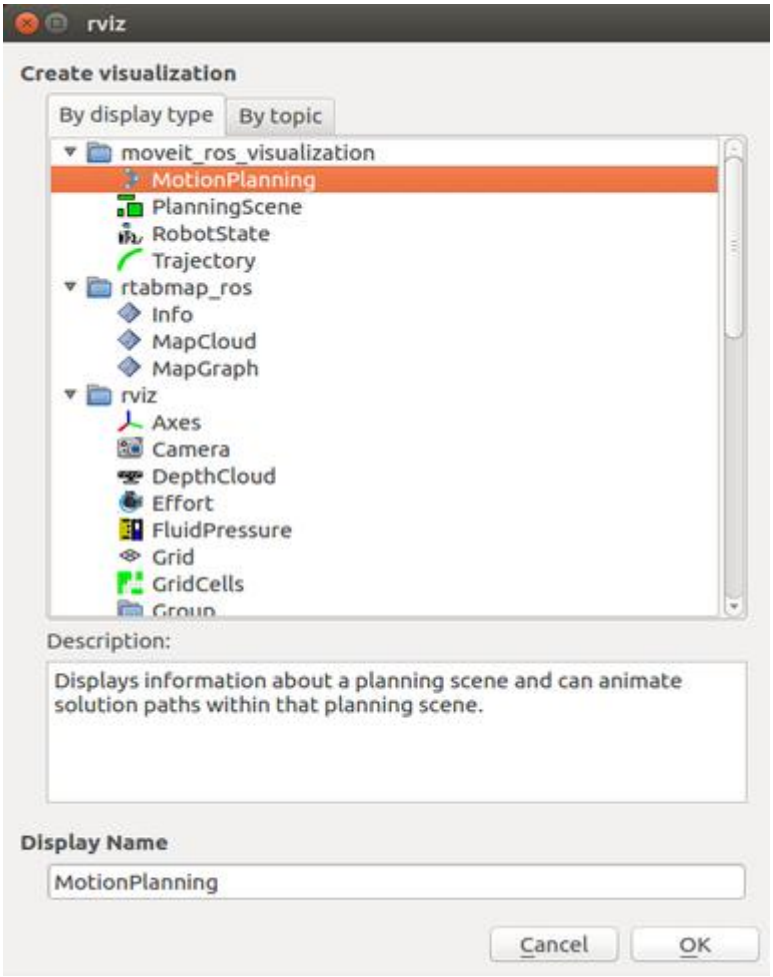
Otra diferencia notable, como se ve en la imagen de arriba, es que la forma de mover el brazo es más orgánica. Puedes coger la cabeza del brazo y ponerla en cierta posición, sin tener que usar menús.
Para ejecutarlo deberás escribir en la terminal:

```
roslaunch youbot_moveit move_group.launch
```

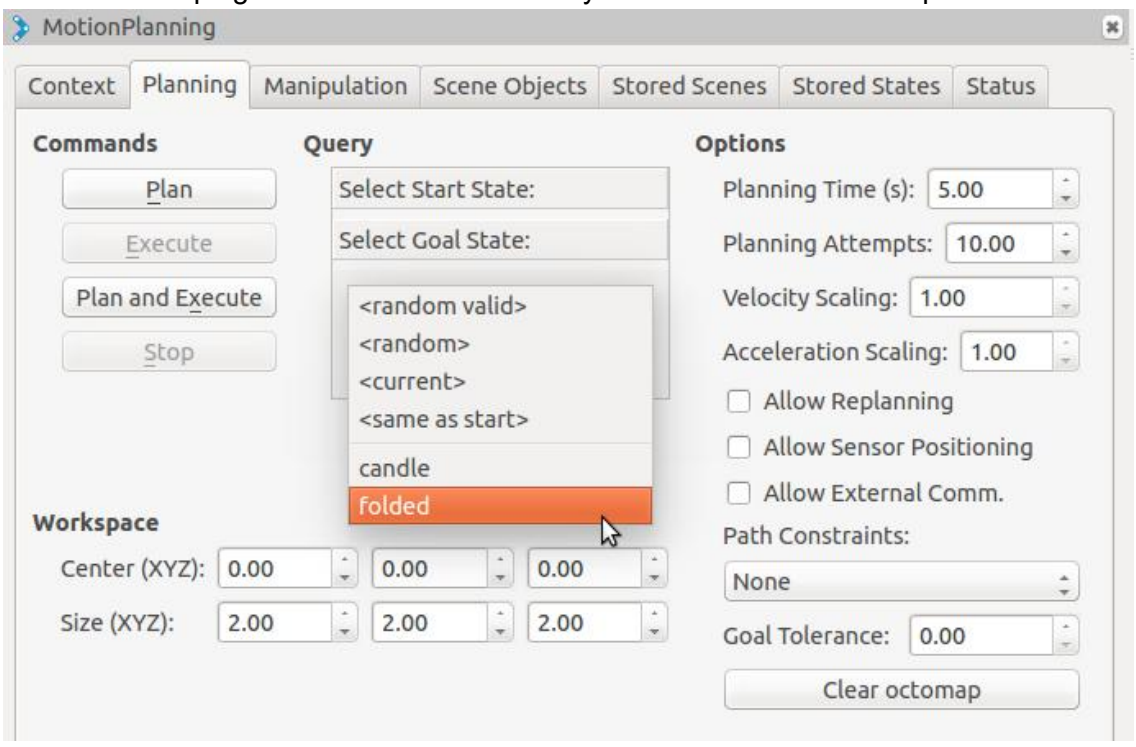
Después, cuando salga el mensaje de que todo está bien, en otra terminal, ejecutar *rviz* de esta forma:

```
rosrun rviz rviz
```

Nos abrirá una nueva ventana con la aplicación *rviz* que se ve también en la anterior aplicación. Para ver el robot tal y como aparece en la primera foto tendremos que darle al botón "Add" que está en la parte de abajo de la sección "Displays" y seleccionar la siguiente opción:

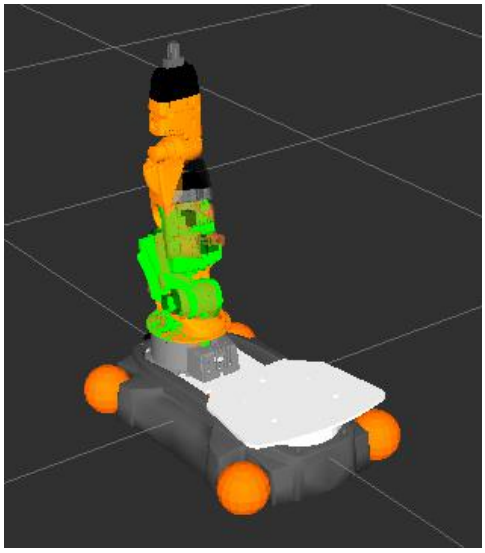



Después de hacer esto nos aparecerá un nuevo menú a la izquierda, debajo de "Displays". Si vamos a la pestaña "Planning" deberíamos encontrar y desplegamos uno de los desplegables en la sección "Query" deberíamos ver estas opciones:



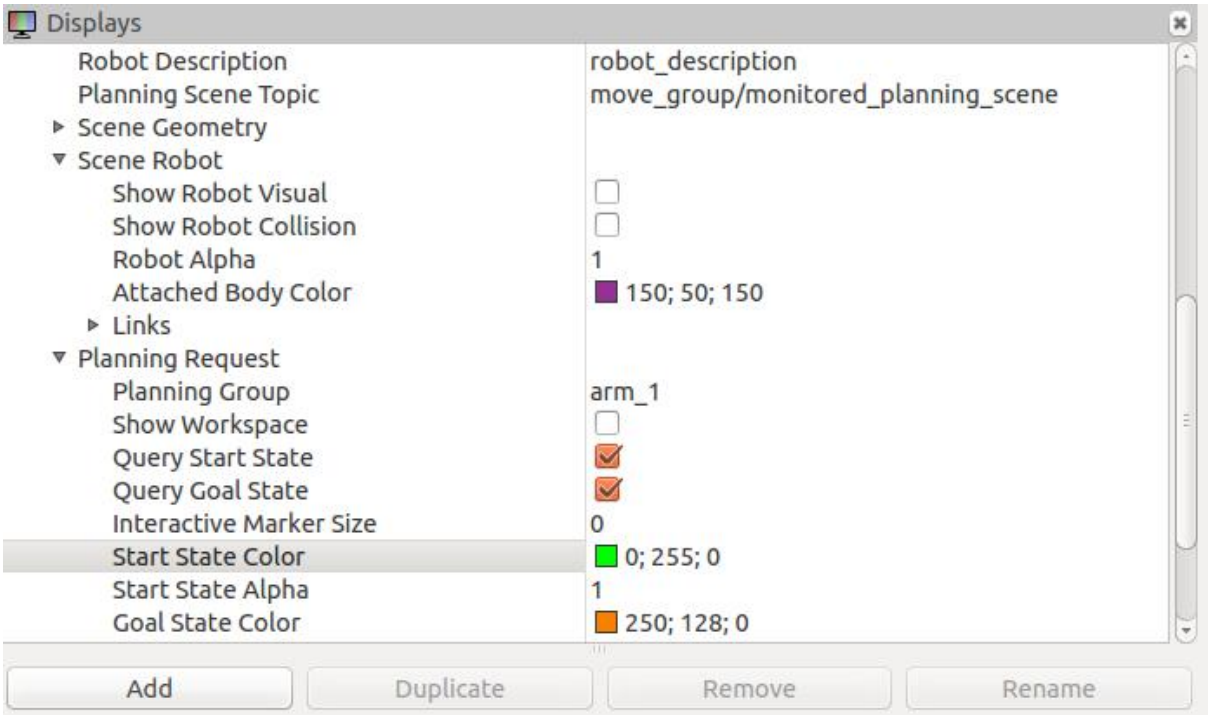
Estas opciones son movimientos o posiciones predeterminadas que podemos poner y son muy útiles por ejemplo para llevar el robot a la posición relajada (*folder*) o si sin querer hemos movido la posición inicial (Start State) ponerlo en la posición que realmente está con la opción *current*. Sin embargo si quieres mover el brazo a una posición que no está predeterminada tendrás que hacerlo con las flechas y ejes que aparecían en la primera imagen de este apartado. Para eso asegúrate de que "Fixed Frame" dentro de "Global Options" en el menú "Displays" es *base_link*.

Cuando ya tengas la posición a la que quieres que se mueva el brazo, dale al botón "Plan" para que se calcule la trayectoria que tiene que seguir el brazo para llegar a esa posición. Verás una animación que mostrará cómo se irán moviendo los ejes hasta llegar la posición deseada.



Si quieres que esto se ejecute en el robot real sólo tienes que darle al botón "Execute". Para hacer este proceso de dos pasos en uno sólo puedes darle al botón "Plan and Execute". Cuando hagas esto, por seguridad, hazlo en un sitio con bastante espacio alrededor.

Sin embargo, puede que tengas dificultades para distinguir la posición original del robot ("Start State") y la posición objetivo ("Goal State"). Para solucionar esto, puedes cambiar las configuraciones en "Displays". Por ejemplo cambiando así dentro de la sección "MotionPlanning":




También hay opción de ejecutar el programa sin usar el robot o sin tener el robot físico escribiendo el siguiente comando:

```
roslaunch youbot_moveit demo.launch
```

En este caso se abrirá directamente rviz con la configuración necesaria para usar el plugin *Moveit!* y podrás hacer pruebas con la visualización sin temer por el robot real.

Si quieres ejecutarlo en tu PC asegúrate de que tienes rviz y el plugin MoveIt! instalado. Para ello puedes dirigirte a: https://moveit.ros.org/install/
Después tendrás que descargarte la configuración específica para este robot:
`git clone https://github.com/svenschneider/youbot-manipulation.git`

## Reutilizar código

Si quieres mover el brazo a distintas posiciones predefinidas, puedes usar las cinemáticas inversas de esta aplicación. En este caso tendrás que incluir la siguiente librerias en tu código

```
#include <moveit/kinematics_base/kinematics_base.h>
#include <pluginlib/class_loader.h>
```

y poner las siguientes líneas para cargarlo como si fuera un plugin,

```
// Instance & initialize IK solver
pluginlib::ClassLoader<kinematics::KinematicsBase>
loader("moveit_core", "kinematics::KinematicsBase");
  try
```

```
  {
      ik_ =
loader.createInstance("youbot_arm_kinematics_moveit::KinematicsPlugi
n");
      ik_->initialize("/robot_description", "arm_1", "arm_link_0",
"arm_link_5", 0.01);
  }
```

que podrás utilizar como si fuera un objeto de una clase, como por ejemplo:

```
ik_->getPositionIK(goal_pose, seed, solution, error_code);
```

## Detección y agarre de un objeto con código QR

Esta aplicación te permite detectar códigos QR con la cámara Kinect y coger un objeto con el brazo. La aplicación tiene características interesantes para poder construir en base a ellas o utilizarlas como parte de otra aplicación. Además al ser modular se puede usar partes de ésta sin tener que preocuparse por dependencias.

El código trae un programa principal que usa los módulos desarrollados para hacer lo descrito anteriormente. Esos módulos son los siguientes:

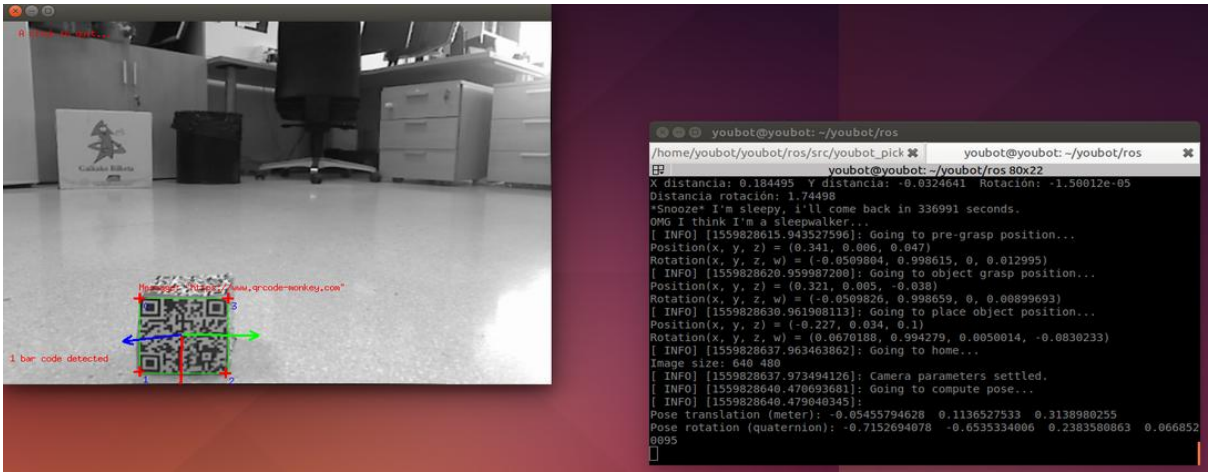
- Detección de QR (`qr_detection`): módulo que, utilizando la cámara RGB de la kinect detecta el código QR. Este código QR tiene que ser de 6x6 centímetros. Además de detectar el código QR también se calcula la pose en la que está, dando así la distancia desde la cámara Kinect al objeto.
- Movimiento del brazo (`youbot_arm`): módulo que tiene varias funciones. Una de ellas calcula la trayectoria que tiene que hacer el brazo para llegar a la posición que le pasemos usando cinemáticas inversas (ver "*Movimiento del brazo con rviz (Moveit!)*"). Otra de las funciones publica las posiciones de los ejes que le pasemos. Para el control de la pinza hay dos funciones: una para abrirla del todo y otra para cerrarla del todo. Si queremos dejar la pinza en una posición intermedia podemos usar la aplicación que publica el valor de la pinza. Por último, hay una función que nos pone el brazo y la pinza directamente en la posición relajada (también llamada *home position,* en inglés).
- Movimiento de la plataforma móvil (`youbot_base`): este módulo es más específico para esta aplicación y con él se consigue mover la plataforma al lado del objeto detectado en base a su posición.

Si quieres probar esta aplicación tendrás que usar el siguiente comando que ejecutará el archivo lanzador (ver *Archivos lanzadores*).

```
roslaunch youbot_pickup youbot.launch
```

Después, en otra terminal, ejecutar la siguiente línea para empezar la aplicación.

```
rosrun youbot_pickup youbot_pickup_node
```




## Reutilizar código

Si quieres reutilizar o usar las funciones del código, en este apartado se darán pistas, tanto de las funciones de cada módulo que se pueden usar o crear como de la utilización de los distintos archivos lanzadores.

### Módulos

Todos los módulos son clases de C++ que dentro contienen un objeto, por lo tanto, si quieres usar las funciones de la clase tendrás que crear un objeto de esa clase. Por ejemplo:

```
YoubotArm* youbotArm = new YoubotArm(n);
```

Se recomienda ver los archivos de encabezado (*.h* o *.hpp*) dentro de la carpeta *include* del proyecto para saber los parámetros de cada función constructora.

#### qr_detection

Si se quiere detectar un código QR con la Kinect, tienes la función `detect()` la cual devuelve `true` si se detecta un QR y `false` si surge algún problema o se le da click a la ventana que abre con la imagen de la Kinect.

```
bool detect()
```

Si además, una vez detectado el código QR quieres encontrar la pose de éste, tienes la función `getPose()`. La función mandará un error si se intenta ejecutar antes de haber detectado un código.

```
geometry_msgs::Pose getPose()
```

Por eso, si únicamente se quiere consultar si se ha detectado un QR o no tenemos la función auxiliar `isQR()` que nos devuelve `true` o `false` dependiendo de si se ha detectado o no.

```
bool isQR()
```

Ten en cuenta que, como se ha dicho anteriormente, el código QR es de 6x6cm y que la posición de éste se calcula en base a su tamaño. Por eso, si quieres usar un QR de distinto tamaño tendrás que editar las siguientes líneas dentro del constructor de objeto de la clase:

```
// 3D model of the QRcode (6cm each side, change this for other metrics)
point.push_back(vpPoint(-0.03, -0.03, 0)); // QCcode point 0 3D coordinates in plane Z=0
point.push_back(vpPoint(0.03, -0.03, 0));  // QCcode point 1 3D coordinates in plane Z=0
point.push_back(vpPoint(0.03, 0.03, 0));   // QCcode point 2 3D coordinates in plane Z=0
point.push_back(vpPoint(-0.03, 0.03, 0));  // QCcode point 3 3D coordinates in plane Z=0
```

Estos puntos representan la distancia desde el centro de la imágen a detectar (en este caso un QR) y sus bordes.

### youbot_arm

Este módulo tiene varias posiciones del brazo predefinidas que se pueden ejecutar con las funciones **goToPregrasp**(), **goToGrasp**(), **goToPlace**() y, quizás la más útil de ellas, **goHome**() que hace que el brazo vaya a su posición relajada. Además también hay funciones para abrir del todo y cerrar la pinza.

```
// Gripper's functions
    void openGripper()
    void closeGripper()

// Arm position's functions
    void goToPregrasp()
    void goToGrasp()
    void goToPlace()
    void goHome()
```

Si se quiere se pueden definir más posiciones o directamente hacer pública la funciones **publishArmValues**() y **publishGripperValues**() para poder usarlas fuera de la clase usando el objeto de esta.

```
void publishArmValues(std::vector<double>& p)
void publishGripperValues(double w)
```

Estas funciones publicarán directamente los valores que le pases. Sin embargo, si quieres usar las cinemáticas inversas (ver *Movimiento del brazo con rviz (Moveit!)*) con nuevas posiciones para el brazo tendrás que seguir los siguientes pasos:

1. En una terminal ejecutar lo siguiente (ver *Movimiento del brazo con rviz dando valores*) y mover el brazo hasta la posición que se quiera. Estos valores serán los ***valores semilla*** (*seed values* en inglés).

   ```
   roslaunch youbot_description youbot_arm_simulation.launch
   ```

2. Sin cerrar lo anterior abrir otra terminal con el siguiente comando. Dale rápido a CTRL+C porque salen muchos mensajes. Quédate con la parte de "Translation" y "Rotation in Quaternion" para meterlo en una

**transformación** (`tf::Transform`) como translación y rotación respectivamente.

```
rosrun tf tf_echo arm_link_0 arm_link_5
```

Para calcular las cinemáticas inversas se usa la función **`moveArm()`** con la información que se acaba de obtener. Esta aplicación llama directamente a **`publishArmValues()`** cuando finaliza.

```
int moveArm(std::vector<double>& seed, tf::Transform& goal)
```

#### youbot_base

En este módulo podrás usar la función **`publishGoal()`** que puede que no se ajuste a la aplicación que buscas pero puedes editar o descomentar líneas para que funcione de otra manera con el paquete de navegación de ROS que ahora mismo no está bien configurado.

```
bool publishGoal(geometry_msgs::Pose& p)
```

### Archivos lanzadores

Los archivos lanzadores te permiten ejecutar varios programas o herramientas a la vez sin la necesidad de abrir una terminal por cada una de ellas, por lo que son una herramienta muy útil y potente de ROS.

Dentro de la carpeta del paquete de ROS (`youbot_pickup`) en la carpeta `launch` encontrarás distintos archivos lanzadores. Aquí tienes una breve descripción de la utilidad de cada uno de ellos:

- `youbot.launch` Como se ha expuesto al principio, este archivo se usa para lanzar los elementos necesarios de la aplicación, tales como los drivers del robot, los de la kinect y la generación del láser de ésta además de un emisor de TF para la posición de la misma.
- `youbot_joy.launch` Hace lo mismo que antes pero además ejecuta la aplicación para manejar la plataforma con el mando.
- `youbot_nav.launch` Este archivo activa también aplicaciones que se usan en la navegación del robot con mapa, tales como `amcl` o `server_map`.

> Si vas a cambiar el archivo lanzador tendrás que hacer `catkin_make install` después para guardar los cambios en el mismo directorio donde se compila normalmente.

# Referencias y más información

- Es recomendable, además de este manual, leer o echar un vistazo al manual de usuario proporcionado por el vendedor (en inglés). (ftp://ftp.youbot-store.com/manuals/KUKA-youBot_UserManual.pdf).
- También puede ser de ayuda el documento de las especificaciones técnicas (en inglés): *Kuka youBot. Operating and Assembly Instructions*.
- Aunque la página del vendedor ya no vende el robot, aún puedes encontrar información interesante en su sección de desarrolladores (http://www.youbot-store.com/developers/) y en la wiki también ofrecida por los mismos (http://www.youbot-store.com/wiki/index.php/Main_Page).
- Si quieres más información de ROS puedes entrar en su wiki donde hay distintos tutoriales desde usuarios principiantes a avanzados e información sobre librerías y herramientas (http://wiki.ros.org/).
- Para más información de *Moveit!* puedes entrar en su web en la que encontrarás información de descarga e instalación, así como tutoriales (https://moveit.ros.org/).
- Si quieres aprender C++ o si tienes alguna duda sobre librerías, funciones o versiones puedes acudir a cplusplus.com (http://www.cplusplus.com/).
- La aplicación para mover el brazo, detectar códigos QR ha sido desarrollada por la alumna Ainhoa Pato, siendo esta parte de su TFG defendido el curso 2018/19. Puedes consultar su memoria en la plataforma ADDI de la Facultad de Informática de la UPV/EHU para más detalles e información. También puedes encontrar el código en el repositorio de github (https://github.com/aps54/youbot_pickup).
- Si se quiere ver este documento en formato digital poner en el navegador la siguiente dirección: http://bit.ly/egokituzyoubotman. No podrás editar pero podrás descargar el documento en distintos formatos editables o imprimirlo.