INSTITUT CATHOLIQUE DES ARTS ET METIERS
Department of Computer Science
Emmanuel ANTO – Mahdi KZADRI - Pierre HENRION

RESEARCH PAPER

AUTONOMOUS HANDLING SYSTEM FOR MOBILE ROBOTS

Design and implementation: Kuka YouBot demonstrator

May 2018

Under the direction of **Ahmed RHIAT**

Summary

Index of images

1. Project description

1.1. Abstract

INCASE is a European Interregional project funded by the Interreg V 2 Seas Programma 2014 – 2020. The project runs from September 2016 to August 2019. We currently focus on Industry 4.0 and its applications. We know from countless projects that the full potential of Industry 4.0 does not unfold until solution concepts holistically encompass the issues facing companies with a high level of competence in all customer-specific topic clusters. With the pooled specialists from a broad range of fields in a dynamic, interdisciplinary Industry 4.0 acceleration units. Our aim is to reveal the result of our research at the outset of the planning phase all added value that they stand to gain from digitization. And then to accompany them with our project expertise. In this way, companies also obtain direct access to KUKA's global ecosystem. A network of highly qualified partners, integrators and suppliers from all areas of the digitized value chain. The added value of integrated, globally interconnected value chains is acknowledged throughout the world. Companies are opening up to the digital transformation in order to successfully compete in the markets of the future. As a thought leader and trailblazer for Industry 4.0, KUKA supplies intelligent solutions for maximizing the flexibility of the entire process chain – from consultancy and conceptual design through to the complete implementation of individual requirements.

In this report, we briefly portray about how a KUKA Youbot can be used with industrial application and with addition of human interactions, how do we communicate with our Youbot. The objective is to create a demonstrator, which shows, how a h*Autonomous handling system for mobile robots*uman factor can arbitrate the communication with Youbot, how effective in daily consumption of the machine, with an industrial point of view.

1.2. Objectives

The automotive industry is undergoing an epochal change. In the age of e-mobility, intelligent vehicles and completely new mobility services, the industry is facing far-reaching changes – from development through to production and logistics. As we walk through into creating a perfect demonstrator, we come across multiple hypothesis, corresponding to integrate human interaction with our Youbot. Now, we do

have solutions that are uncomplicated and highly adapted for the industries. Youbot can be functionally

broken into three elements. The base for navigation, arm and perception.

As we walk through into creating a perfect demonstrator, we come across multiple hypothesis, corresponding to integrate human interaction with our Youbot. Now, we do have solutions that are uncomplicated and highly adapted for the industries. Youbot can be functionally broken into three elements. The base for navigation, arm and perception.

1.3. Hardware used for prototyping

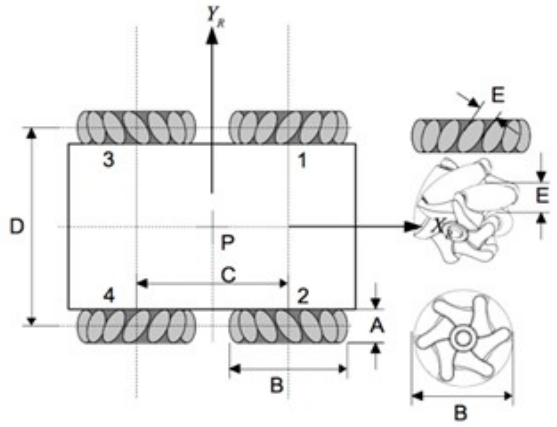


Illustration 1: Youbot Base

Overall Weight : ~20 kg
Permissible Payload : 20 kg
Overall Length : 580 mm
Overall Widht : 380 mm
Overall Height : 140 mm
Minimum velocity : 0.01 m/s
Maximum velocity: 0.8 m/s
Power supply: 24 V
A = 74,87 mm ; B = 100 mm ; C = 471 mm ; D = 300,46 mm ; E = 28 mm

Control setting

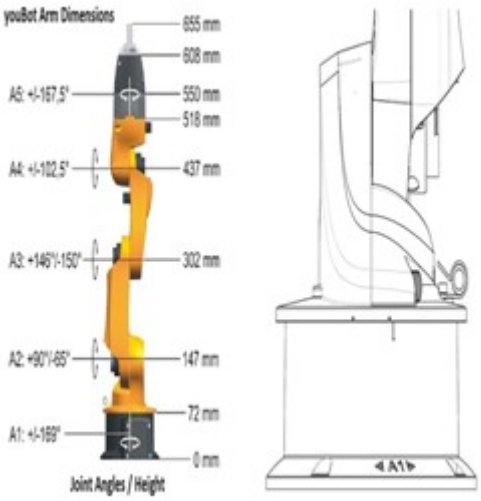


Illustration 2: Arm joints

No. of Axis : 5

Height : 655 mm

Work envelope : 0.513 m2

Weight : 5.3 kg

Payload : 0.5 kg

Position repeatability : 1 mm

Power supply: 24 V

Drive train power limit: 80 W

Axis Speed : 90 deg/s

Encoders threshold: 1.000 Hz

Axis data:

Name Range Velocity

Axis 1 +/– 169° 90 °/s

Axis 2 + 90°/– 65° 90 °/s

Axis 3 + 146°/ - 151° 90 °/s

Axis 4 +/– 102,5° 90 °/s

Axis 5 +/– 167,5° 90 °/s

Gripper

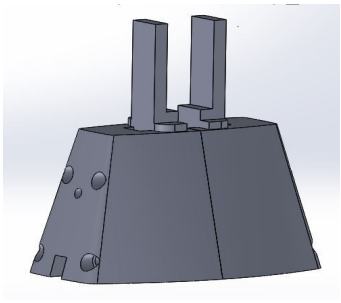


Illustration 3: Gripper Youbot

Gripper stroke : 20 mm

Gripper range : 70 mm

The TMCM-KR-842 is the gripper board for the KUKA youBot arm placed inside the gripper at the upper end of the arm. The main purpose of this board is controlling the two linear stepper motors (connected separately to the board) inside the gripper. These motors move the two gripper fingers.

As a natural design, Kuka Youbot is amaurotic. As a part of this research project, we added a camera for detection and recognition purposes. Here are the few specifications of the Asus Xition PRO.

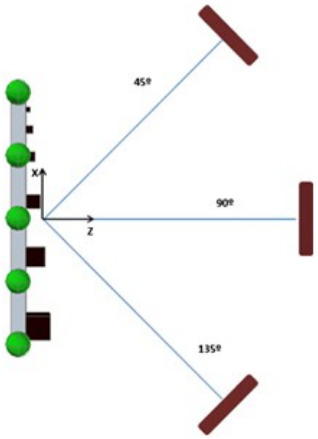


Illustration 4: Asus Xition Pro (VGA (640x480) 30fps, QVGA (320x240): 60fps)

Illustration 5: Youbot with camera

The natural physical appearance of the Youbot is without the perception camera and the laser. We have added a developer's camera Asus Xtion Pro and Hokuyo laser. In order to accomplish our target of bringing human interactions into act, we have added two other perceptions to Youbot. Hence, the new hardware components are as follows:

Asus Xition Pro	Youbot Arms	Hokuyo URG-04LX-UG01
Sensor RGB& Depth& Microphone **Depth Image Size** VGA (640x480) : 30 fps QVGA (320x240): 60 fps **Resolution** SXGA (1280*1024) **Field of View** 58° H, 45° V, 70° D (Horizontal, Vertical, Diagonal) **Distance of Use** Between 0.8m and 3.5m **Power Consumption** Below 2.5W **Interface** USB2.0/ 3.0 **Software** Software development kits(OpenNI SDK bundled)	**No. of Axis** 5 **Height** 655 mm **Work envelope** 0.513 m2 **Weight** 5.3 kg **Payload** 0.5 kg **Position repeatability** 1 mm **Power supply** 24 V **Drive train power limit** 80 W **Axis Speed** 90 deg/s **Encoders threshold** 1.000 Hz	**Measuring area** 20 to 5600mm, 240° **Accuracy** 60 to 1,000mm : ±30mm, 1,000 to 4,095mm : ±3% of measurement of angular resolution **Step angle** Approx. 0.36°(360°/1,024 steps) **Scan time** 100ms/scan

2. Method and implementation

2.1. Hardware specificities and configuration

The standard Kuka youBot is actually an assembly of three modules: a arm, a base, and an internal PC. These 3 main parts are "autonomous" and exchange data through the built-in EtherCAT[1] bus. Thanks to this configuration the modules can be used independently, thus allowing more flexibility and hardware customization (i.e. by adding a second arm, or by removing the base...). It's originally wired as following:

[1] EtherCAT is a real-time communication protocol

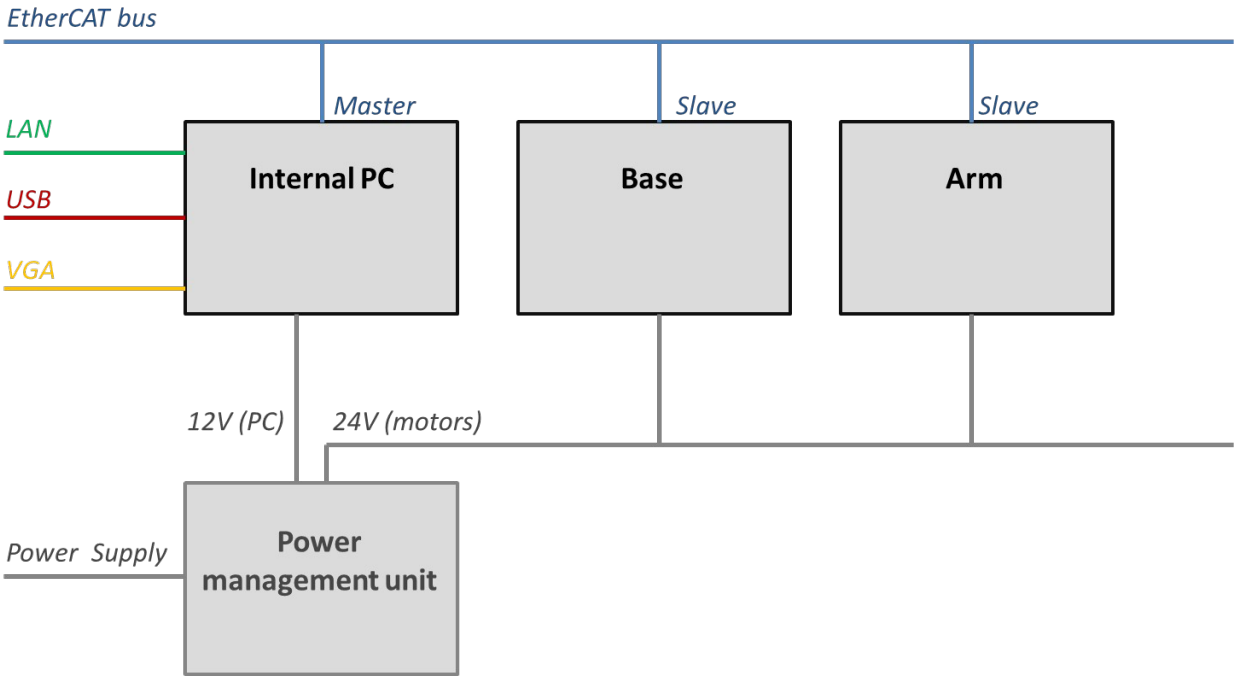


Illustration 6: YouBot schematic block

The official youBot driver consists in a C++ API linked to an EtherCAT driver (Open Source **SOEM**) that is responsible for sending data through the bus ; this API can be installed not only on the internal PC but on almost every computer running on Ubuntu (or other linux system) as well ; this enables even more levels of control[2] :

- Assuming that the drivers are installed and properly configured, the algorithms can be runned directly on the **internal PC** which is already connected to the EtherCAT bus, and can take care of all the software part.
- Algorithms can also be runned on an external computer executing youBot driver and connected to the bus ; in this case this computer will send data directly to the actuators over the **EtherCAT** protocol (internal PC being bypassed).

Both Camera and laser sensors are working with **USB 2.0**, so they need to be worked on a PC (either internal or external), as following:

[2] See http://www.youbot-store.com/wiki/index.php/API_architecture

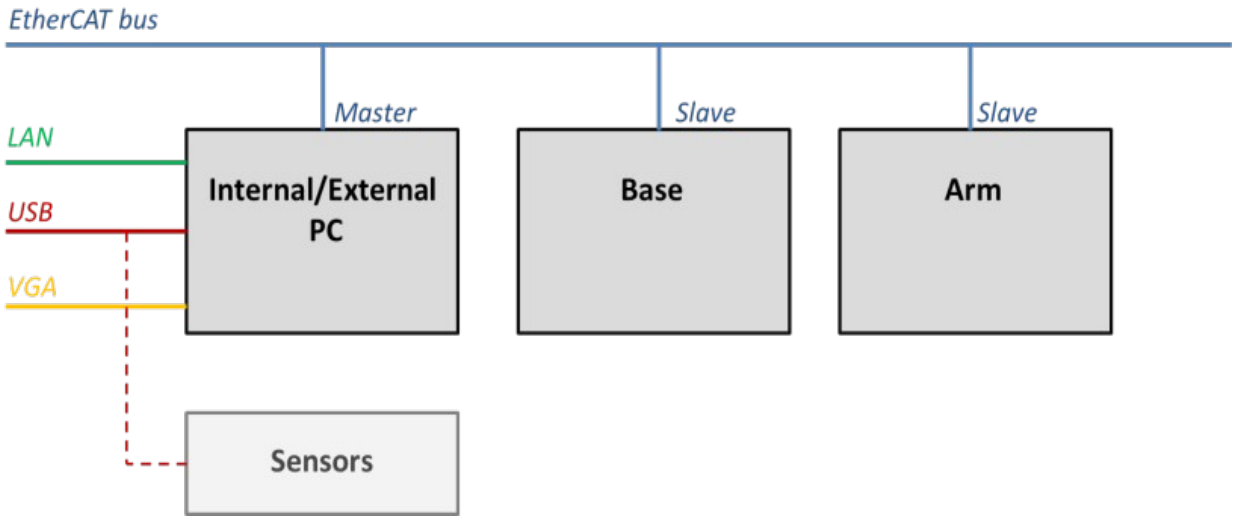


Illustration 7: USB sensors wiring on YouBot

The "core" PC is also supposed to act as a gateway between the youBot and the outside world, being connected to whatever wired or wireless connection (Bluetooth, Wifi, Ethernet, ...) depending of our needs.

2.2. Program functional breakdown

On the "coding" side, we divided the project in three main research fields, each of us being responsible for one:

- **Detection:** consists on all the investigations regarding to the robot ability to understand the world surrounding him. This includes sensing (i.e through Asus Xtion camera) as well as the algorithms for the treatment of raw data (people detection, face recognition, object recognition and 3D mapping...)
- **Navigation:** this part aims to make the robot able to move in its environment in a "smart" way, including mapping, self-localization, trajectory planning from its pose to the target, dynamic obstacle detection and avoidance.
- **Arm kinematic:** includes all the calculation required to make the arm catch an object by applying inverse kinematic in order to convert the actual object pose in a set of angular instructions for the youBot articulations. This also has to determine the best way of catching the object according to its shape.

Representing it in a schematic block way, it would look like this:

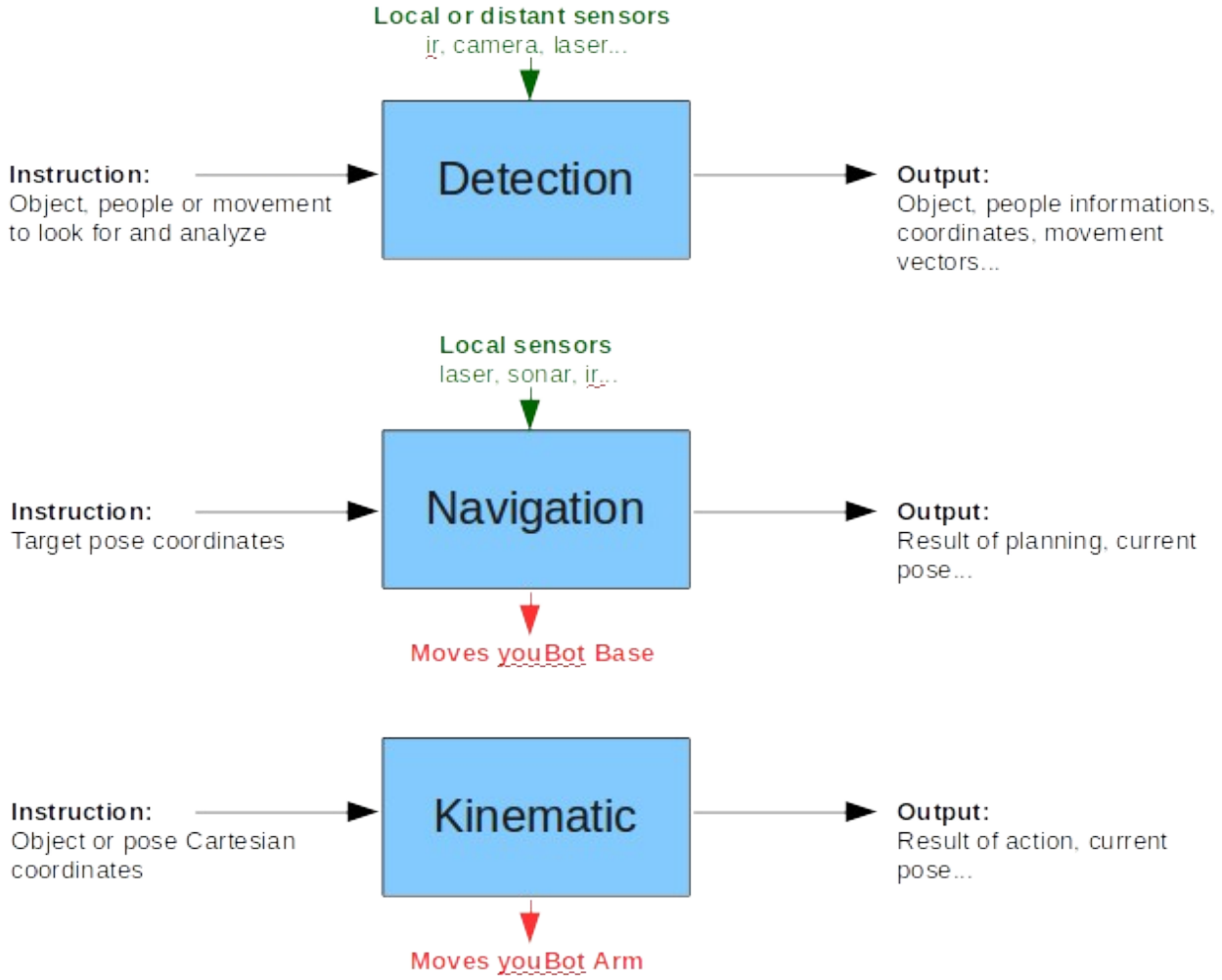


Illustration 8: Representation of the three main research fields

The project is also composed of two others "modules", the first one being responsible for communication with the final user, and the second one managing communication between all the nodes through "high abstraction level" algorithms, thus being the actual intelligence of the robot:

- **User interface** is the set of modules that permit the final user to send instructions to the robot. Three parallel methods are being developed: keyboard control, voice recognition, and web interface.
- **Action server** is the "core" part of the project, containing different routines and demonstration algorithm that shows how to combine the "blocks" in order to perform various complex tasks.

At the end, we get this architecture:

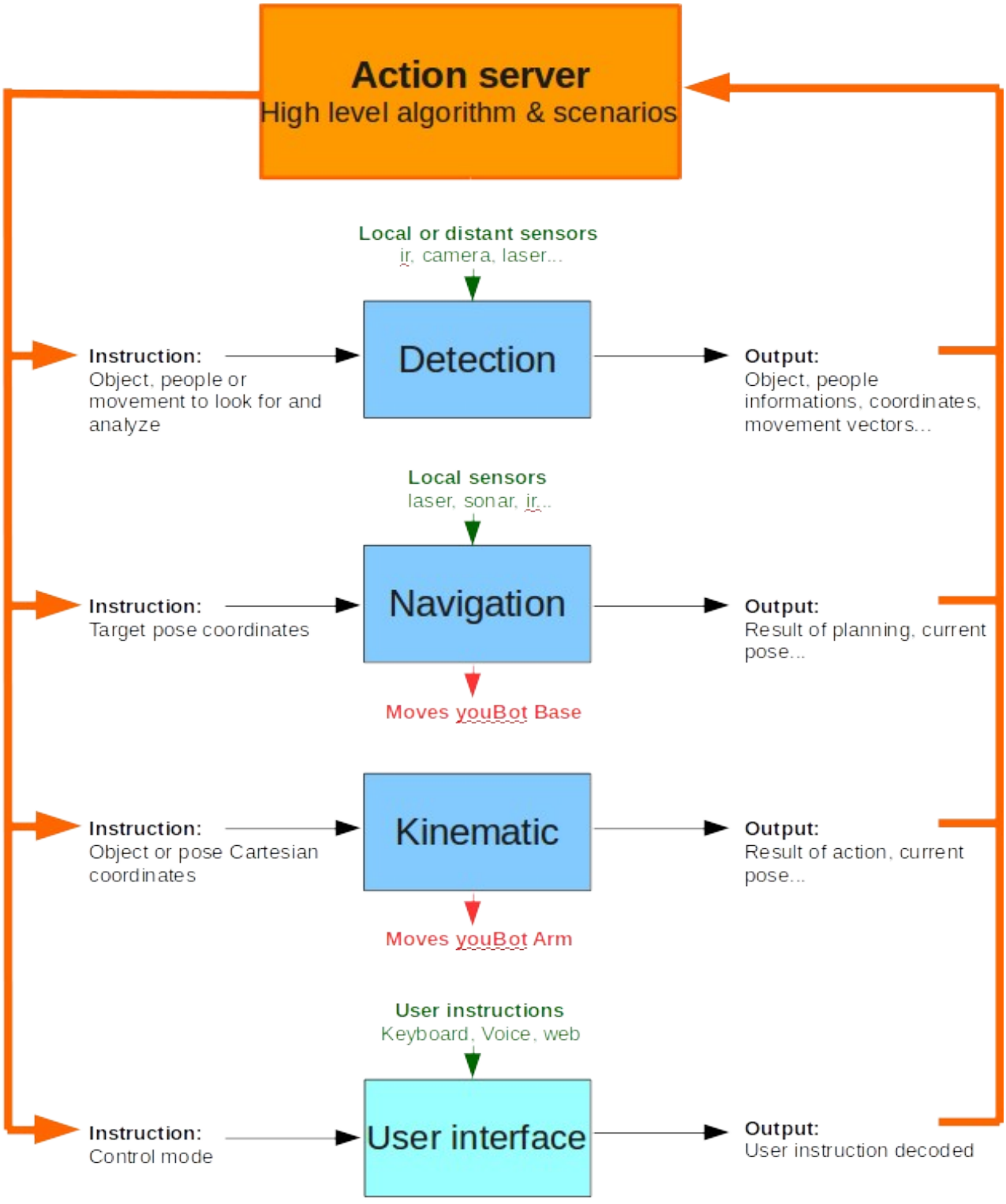


Illustration 9: Program global architecture

This functional breakdown permit us to develop independently each technical field, while making the final routines and scenario be totally conceptual, light, and easy to modify.

2.3. Software philosophy

In order to make our development even more efficient and flexible, we chose to build our programs within the **Robot Operating System** (ROS) environment instead of coding everything from scratch.

According to the organization responsible for it development, ROS is "a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source."[3]

The main concept of the Robot Operating System is to **go further in coding abstraction** by providing a framework that take care of all the hardware-related problematics, while enabling Multi-lingual programming (C++ and python are fully supported, MATLAB and Java are being implemented).

Within ROS, individual programs are working as nodes, in a Peer-to-Peer configuration ; they communicate over defined API (ROS topics, messages, services, etc):

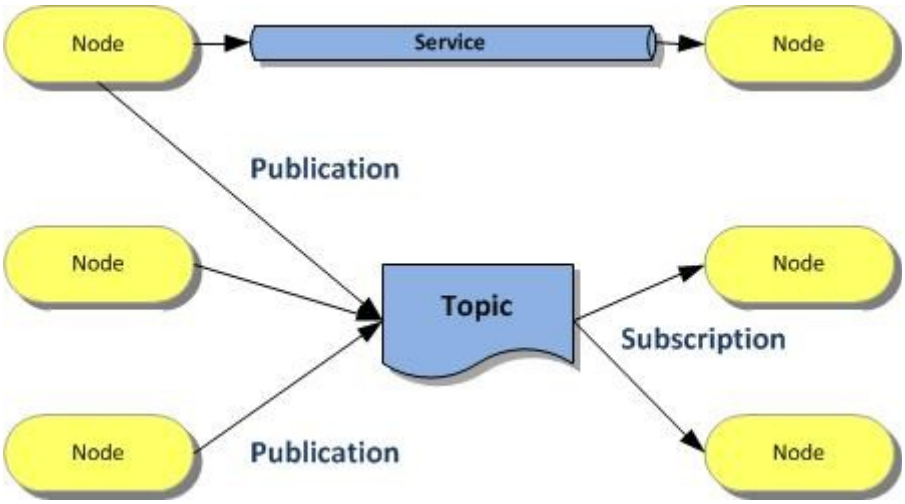
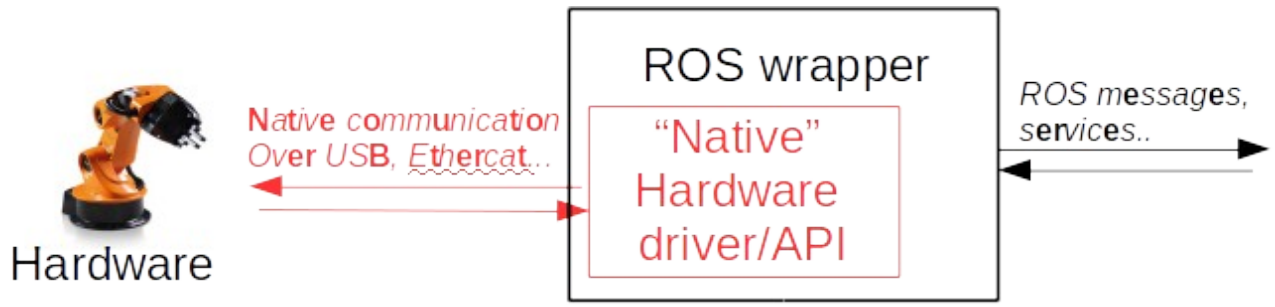


Illustration 10: ROS philosophy

The nodes can be either functionality-related (in our case Detection, Navigation, Kinematic...) or hardware-related (Hokuyo Laser, youBot, Asux Xtion...). The hardware-related nodes are implemented as shown below:



[3] See http://www.ros.org/

This way the functionality-related nodes communicate with the hardware-related nodes using standard ROS API messages ; changes in hardware only requires changes in the related "wrapper" node.

As a result, using ROS in our project makes our program get 1 more level of abstraction, as shown below:

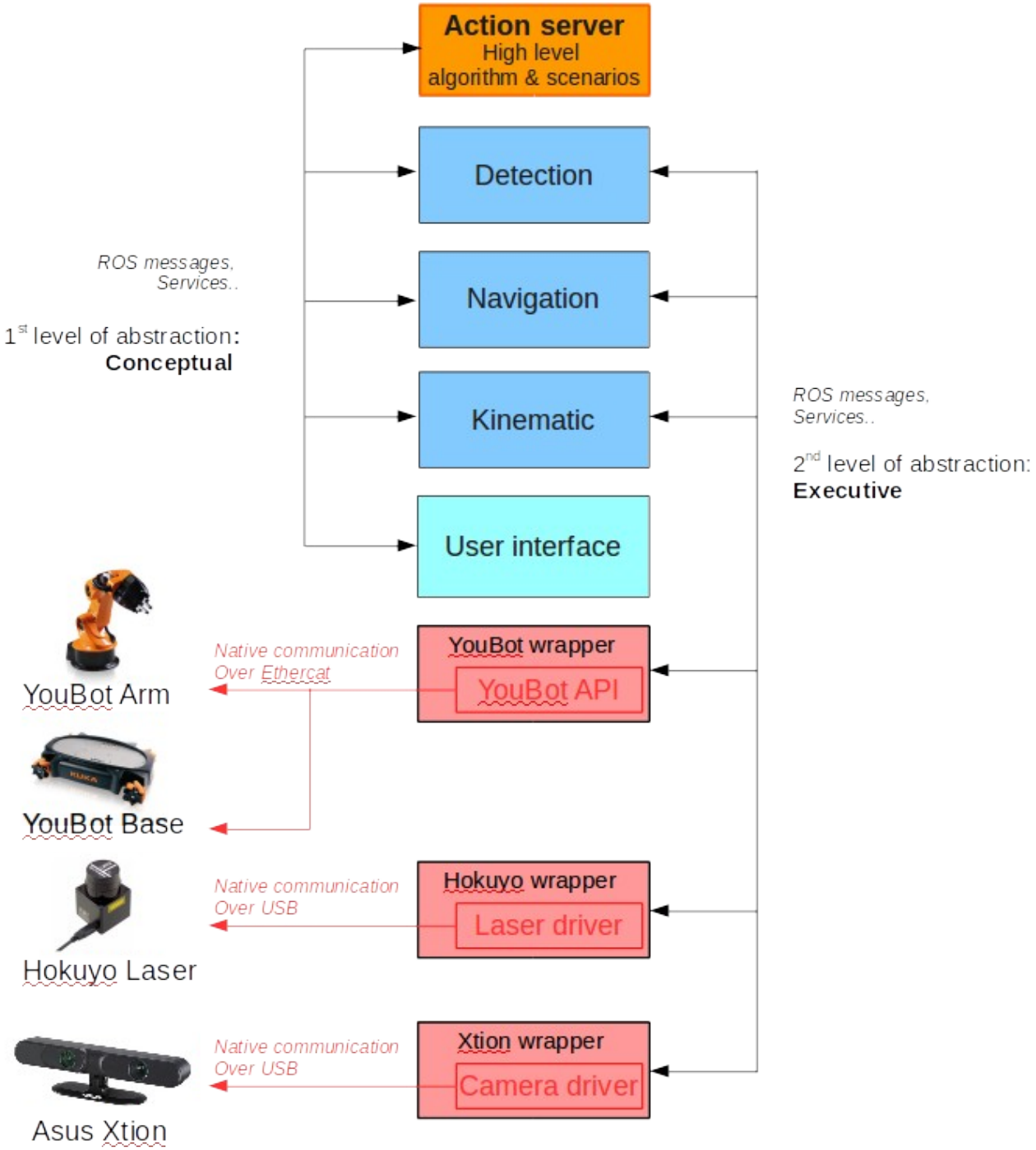


Illustration 11: Representation of the levels of abstraction

This configuration makes us able to add, remove or change hardware very easily, with almost no modifications in the code. Even different robots can be driven by our programs, which looked very relevant to us in a research context.

When talking about the "cooperative" aspect of our project and the importance of sharing the knowledge form student to student and between universities, ROS is one more time a good asset since its entirely built for code sharing.

A lot of robots on the market have to face the same problematics (catching objects, localization, ...) and developers often needs to start with the same "base" ; that's why ROS comes with a lot of pre-built features developed by its community. In our case, starting with these programs made us avoid a lot of fastidious and time-consuming work and directly focus on the core of our project.

Last but not least, we used the ROS implementation of the **ssh** protocol to makes our Workstations and the robot shares the same ROS core system, thus enabling native ROS communication between them over our Local Area Network:

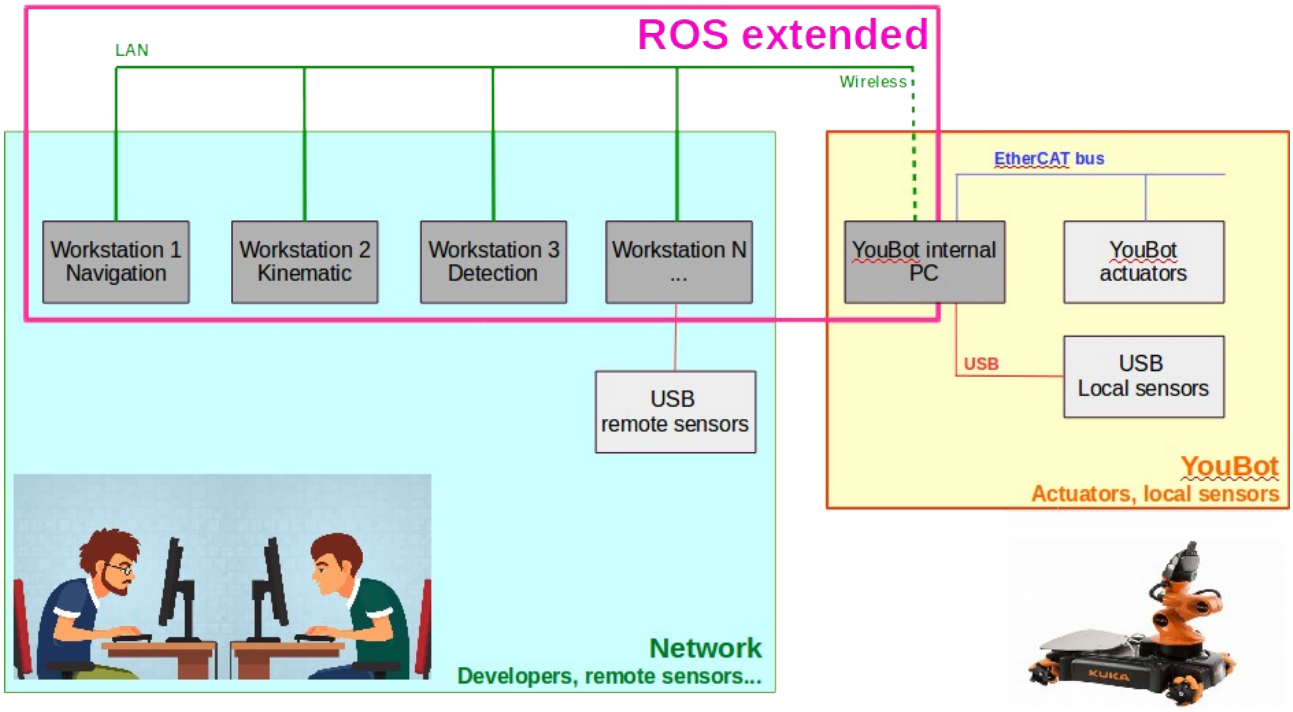


Illustration 12: Network architecture

ROS being extended in the network, nodes can interact perfectly even if they are ran in different computers. We chooses to run on youBot PC the hardware-related nodes only, while algorithm and visualization tools would run on our workstations. The benefits are the following:

- **For development**, it makes us able to develop on our own computers and try the programs directly (no uploads to the robot needed). It also permit us to have several people working in parallel on the same hardware, everyone developing in its own computer, but still being linked to the real robot.
- **For power management**, it permit to split the load between different machines, YouBot internal PC being not very powerful and 32-bits only. By running calculations and visualization tools remotely, we get better performances and save the robot battery.
- **For flexibility**, it makes possible the use of remote sensors or actuators, so the robot can use, for example, data from a remote camera, or even from another robot to perform its tasks.

3. Results & development

3.1. Detection

OpenNI (Natural Interactions)

According to the official documentation given by OpenNI, it can be defined as, OpenNI or Open Natural Interaction is an industry led non-profit organization and open source software project focused on certifying and improving interoperability of natural user interfaces and organic user interfaces for Natural Interaction devices, applications that use those devices and middleware that facilitates access and use of such devices. PrimeSense's depth acquisition was enabled by "light coding" technology. The process coded the scene with near-IR light, light that returns distorted depending upon where things are. The solution then used a standard off-the-shelf CMOS image sensor to read the coded light back from the scene using various algorithms to triangulate and extract the 3D data.

Face Recognition in ROS

Face detection and recognition has been in existence for many applications. We are using a package called "Procrob face recognition" which provides solutions for the ROS working environment. The concept of face detection is performed using many algorithms. We used Haar cascades to implement our face recognition software.

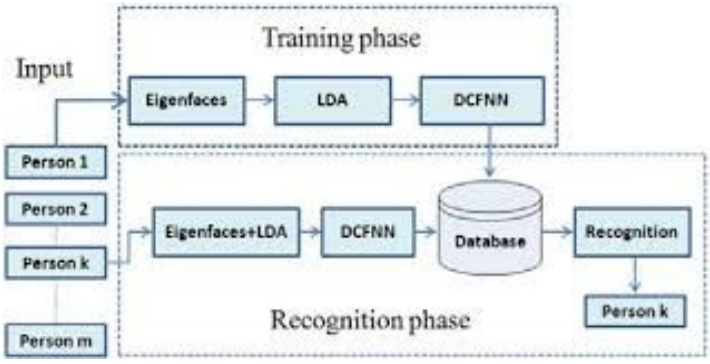


Illustration 13: Recognition phase

According to the algorithm, the program takes multiple samples of the user 1 and saves in its database. Then they are converted into grayscale images and subjected to compilation. This is how face detection works.

Face Recognition also works under same algorithm, but it involves a call function to the database. Once a face is detected, the program compares the pixels of the detected image to the once on the database. If there is a closest match, it shoots out with the, say user 1 name as output.

How could it be used as an industrial application? As we can imagine conglomerate solutions, we came up a few for Youbot. The machine can be trained to recognize user's face and can make it follow them. It is done by creating a transformation file which points the direction and navigating in the pointing direction.

Object detection

Object detection is implemented in Youbot because, as a part of the demonstrator, we need this software to instruct the gripper to pick the object. According to the algorithm, we need to train the system to see and detect only the objects that we are interested in. We neglect other objects in the field area. While the object detection program is running, with the help of our camera, we create an image archive and store it in our database. We launch the camera topic to be read as "/camera/rgb/image_raw". This helps the camera to read the values in the RGB format for the manipulation. This way of object detection is known as SIFT, SURF, FAST, BRIEF. At once the images have been seen on the screen, next step is to add them on the database for detection. *Note:* In this algorithm only the images that already exist in the database are recognized.

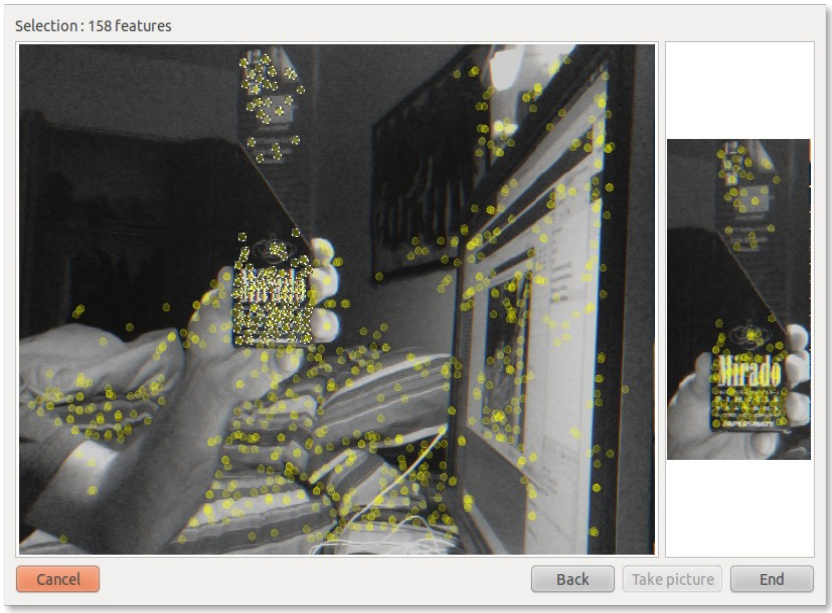


Illustration 14: Object detection: Adding from the scene

So, we select the object that we need to detect before performing any operations. Once we added them to the database, then we reload them into the work area so that the object can be detected. The images are stored in grayscale manner, for the cascade to function with the flow of algorithm.

At once the images are loaded into the work area, the output screen detects the objects from the scene. The program tries to square the objects that are detected from the scene. The main advantage of this software is that, it has the capacity to differentiate the different objects such as object1, object2, etc...

The following picture describes how an object is detected from the scene. The visualization tool also provides us with hands full of options to edit, modify and delete the images taken from the scene. The algorithm draws a rectangle around the detected objects:

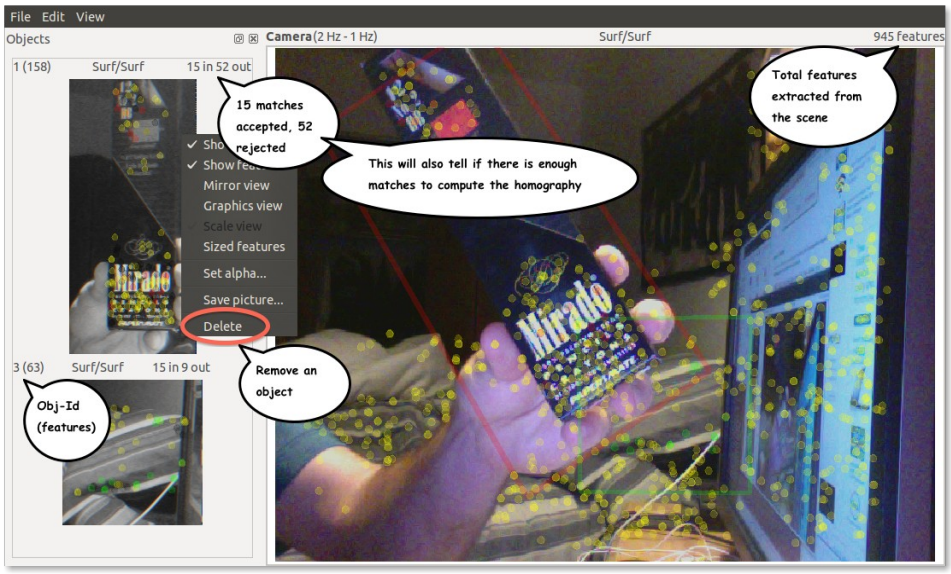


Illustration 15: Object detection: Recognition phase

Once the objects are detected, we relate it to produce a transformation file based on the detection. A transformation file is created with respective to the object and the camera. Once we have a detection in 2D using X transform, we calculate the points in 3D that correspond to 2D pixels, producing a pointcloud (in 3D). By applying techniques such as PCA (principal component analysis) we get the point cloud of the detected object position. The end product is the object pose, however now we need a graspable pose, which depends on the manipulator and gripper. Ideally this graspable pose should get from a grasp planner.

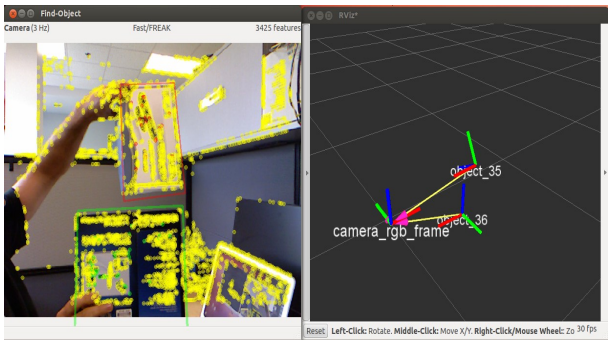


Illustration 16: Object detection: Creating transformation file

3.2. Navigation

The goal of the *Navigation* part of the project is to make the robot able to move in space autonomously from a point to another in a "complex" environment (with obstacle and complex path) like an office or a factory.

The most basic navigation programs consists in a kind of "blind" navigation, where the robot is given a preconfigured direction or trajectory to follow until it detects an obstacle with a local sensor (mostly laser or sonar). When this situation happens, it can either stop and wait, or try to avoid it by using some iterative algorithm. If this logic works for very simple and well-known situations, we realized that it's not powerful enough for situations involving complex path or precise positions since it's subject to a lot of errors, either due to sensing capability or algorithm lacks.

To get rid of these problematics, the solution we've been finally working on consists in a map-based navigation augmented by local sensing and based on the ROS navigation stack. In this context, a static map of the building is used as a reference ; robot or obstacle poses like navigation goals are being expressed as 2D (x;y) coordinates in this referential. The whole navigation stack includes several sub-objectives that can be decomposed as follow:

- **Obstacle detection:** the navigation stack being built around a fixed map, objects poses have to be expressed in its referential ; however sensors (either if located on a robot robot or elsewhere) returns the depth profile according to their own pose. The link between "raw" data from the sensors and the actual pose of the objects detected is calculated by using twist transformation matrix between the map origin and the known translational and rotational position of the sensors. These positions has to be previously known by the algorithm so the virtual environment set up is the exact mirror of the reality. For "moving" sensors, like the one located on the robot, one more step is necessary since the twist transformation has to include the robot self-localization.

- **Self-localization:** the robot has to know its own position at every moment. This has been achieved by using a SLAM (Simultaneous Localization And Mapping) algorithm that combines odometry (move measurement) and local depth sensing to get more accuracy: the static map being known, a comparison is made between space theoretical profile (walls, pieces of furniture...) expected due to odometry-based localization and the actual depth profile measured by the sensors so the robot can correct its position dynamically.

- **Trajectory planning:** starting from a given 2D pose target in the map referential (x;y), and after calculation of its own pose, the robot has to determine the best theoretical path to move to the target according to the building map, thus avoiding walls and dead-end (global planner). In addition to this, it also has to avoid unexpected obstacles on the road when it detects it with its local sensors, deviating from its original path accordingly and coming back to it when the obstacle

is gone (local planner). Different approach are currently being tested : in addition to the original planner algorithm from ROS navigation stack, we implemented an algorithm called *eband local planner* that relies on a chain of bubbles, and we are working on an even more sophisticated approach called *teb local planner* that do the same but being a time-based planner instead of being constraint-based like the others.

The navigation stack has also been implemented a gateway for the "detection" part of our project so the detected objects or people are automatically added to the map, their absolute location being known this way. Since the final program obtained integrate all these parameters in the same algorithmic environment, our supervisors programs only needs to send simple instructions (a target pose, or the object to look for) to get all the job done internally. This approach is also useful for information sharing between different robots moving in the same environment, as shown below:

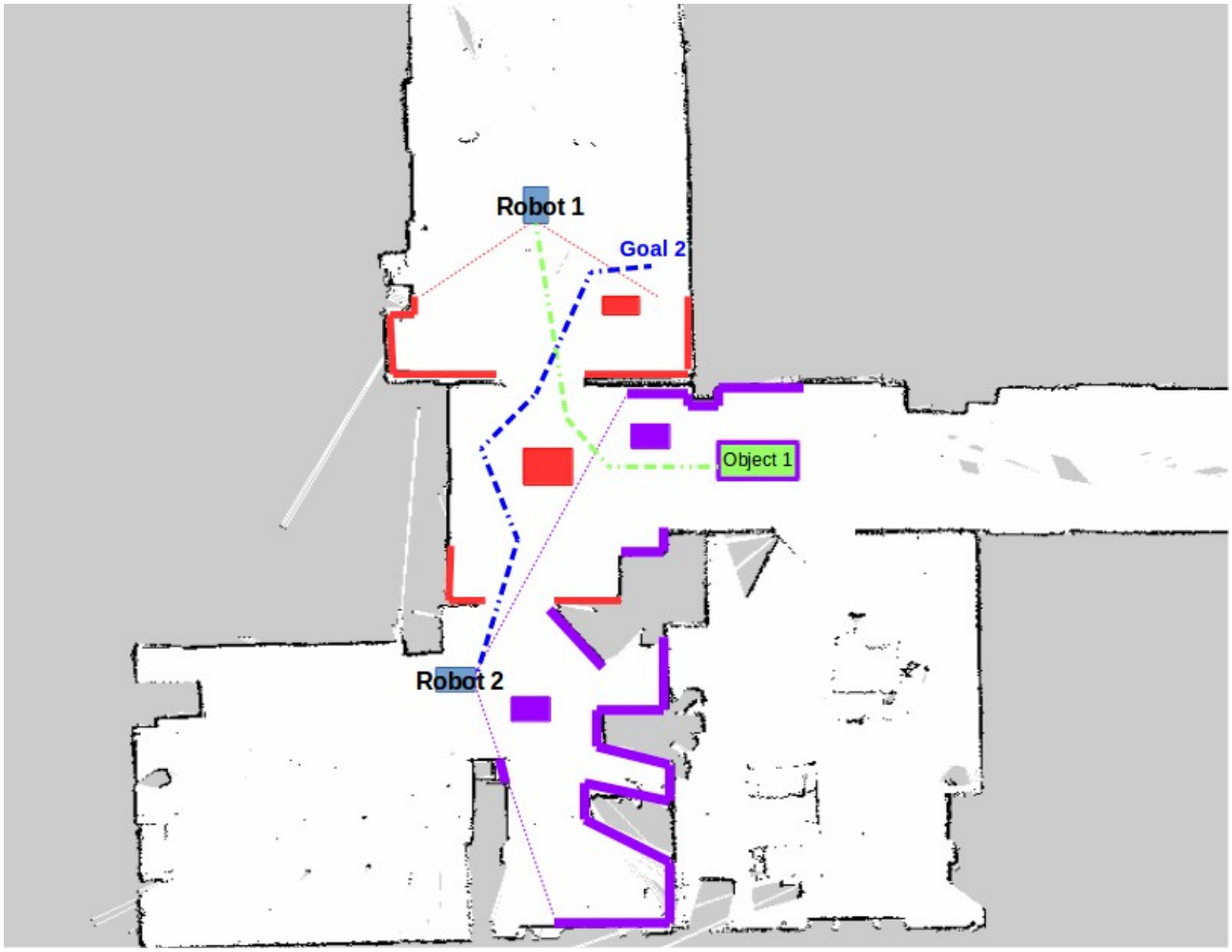


Illustration 17: Example of cooperation between two robots at ICAM

In this example, **Robot 2** uses its own sensor data (purple) as well as the one from **Robot 1** (red) to calculate his trajectory to his goal (**Goal 2**). At the same time, **Robot 1** make the same operation in order to catch **Object 1** which has been detected by **Robot 2**.

3.3. Kinematic

A robot is a machine that can interact with its environment. After navigation and detection of objects steps the robot should be interacted and manipulated, picking up, moving objects around and putting them down again. This manipulation depends on an accurate and precise manipulator. For that we use Inverse Kinematics.

Inverse Kinematics

Inverse kinematics specifies the end-effector location and computes the associates joint angles. For serial manipulators this requires solution of a set of polynomials obtained from the kinematics equations and yields multiple configurations for the chain. The case of a general 6R serial manipulator yields different inverse kinematics solutions, which are solutions of a five degree polynomial.

Inverse Kinematics Solution

- Jacobian
- Cyclic Coordinate Descent (CCD)
- Required to implement in assignment 2

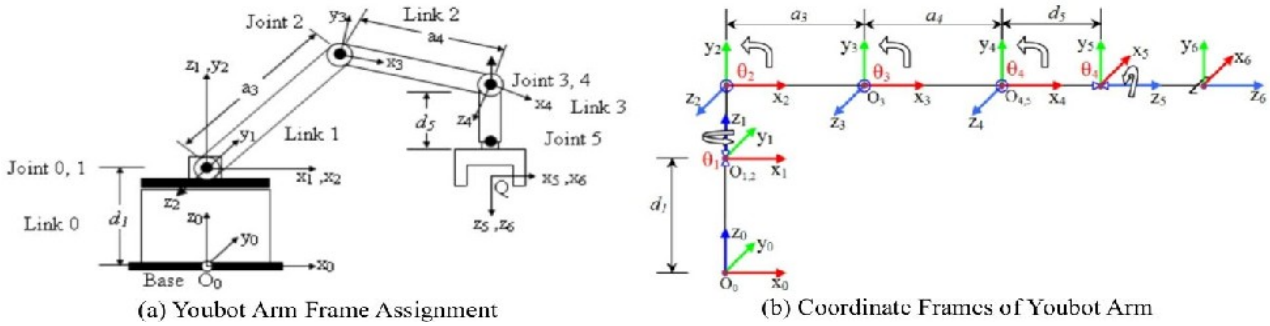


Illustration 18: Schematic of the Youbot manipulator

Inverse Kinematic programmatic algorithm

1. Begin by calculating the vector from current effector position to target (goal)

2. If the vector distance between current effector position to target is less than threshold, then exit

3. Using this information above, calculate the Jacobian by calculating the formula of each entry of the Jacobian

4. Invert the Jacobian (Pseudo-inverse of Jacobian is required)

5. Determine the force F (vector offset between effector position and goal)

6. Compute joint velocities

7. Integrate joint velocities to obtain joint rotation and apply the rotation of the arms

8. Check if goal has been reach and then exit

9. Apply changes

This algorithm of inverse kinematic with Jacobian solution was built during this project and has been tested successfully.

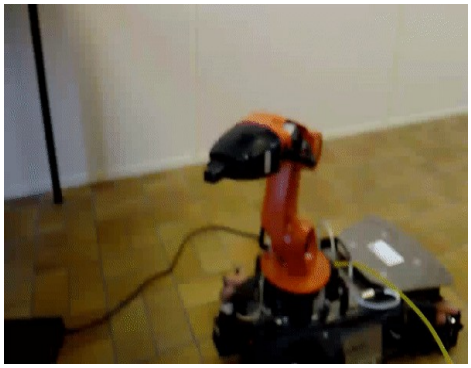


Illustration 19: Youbot arm several coordinates in space

Moveit! For robotics Industrial

Moveit! Is a development environment for robotics which is used to create advanced simulations, incorporating the latest improvements in motion planning, manipulation, 3D perception, kinematics, control and navigation. It provides an easy-to-use platform for developing advanced robotics application, evaluating new robot design and building integrated robotics products for industrial, commercial, R&D and other domains.

Moveit! Provides a great flexibility to use the inverse kinematics algorithms using Moveit! setup assistant.

For this project we use kdl kinematic solver from Orocos Kinematics and Dynamics Library. This library transform Denavit Hartenberg parameters into matrices, which are then multiplied together to calculate the relationship between joint positions and end-effector pose, but users can write their own IK solver as a Moveit! plugin and switch from the default solver plugin whenever required.

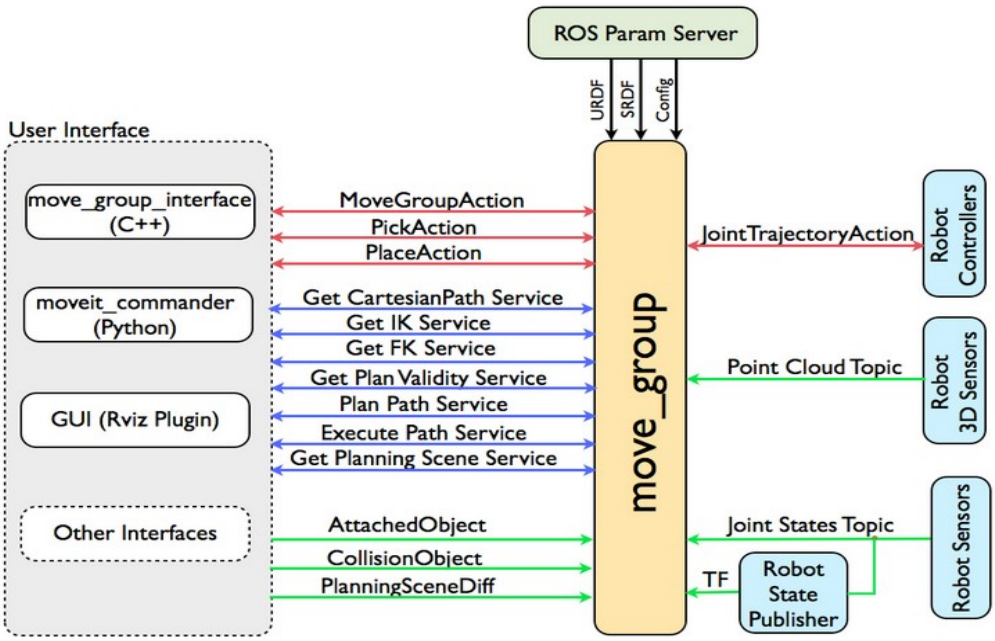


Illustration 20: System Architecture for Moveit!

In simulation mode we used the object information pipeline for Jaco arm and we have changed all script to make it compatible for Youbot robot.

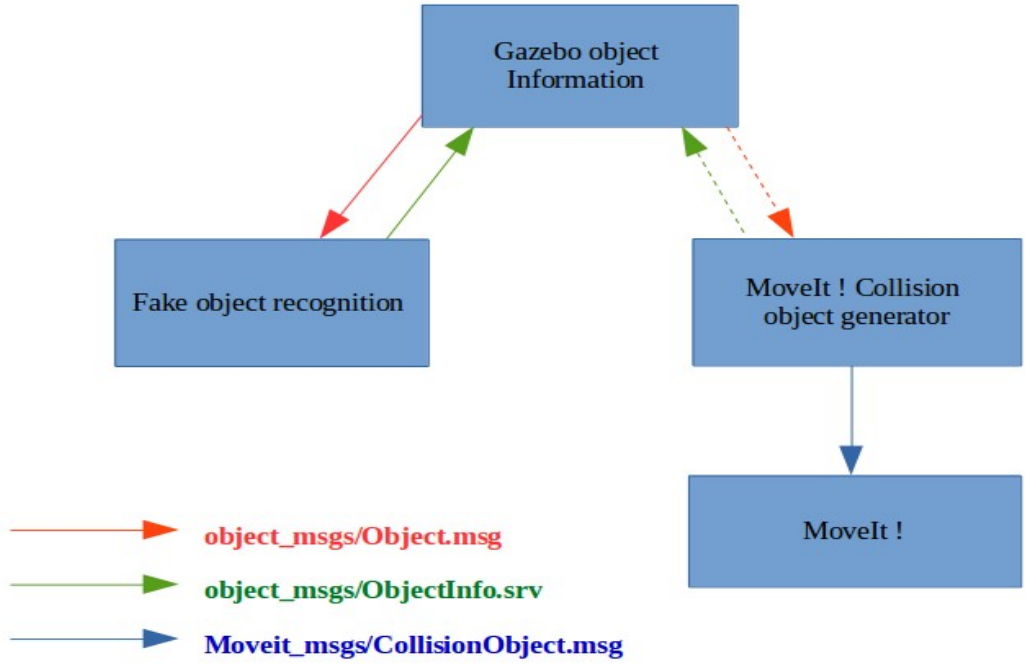


Illustration 21: Explanation of Nodes

Node 1 . Gazebo object information service

The Gazebo object information services provides a service which can be used to request object information out of Gazebo.

Node 2 . MoveIT! Collision object generator

The MoveIT Collision object generator subscribes to messages type: object_msgs/Object, converts them to the right message type accepted by MoveIT! and then forwards that information to MoveIT! Every time such a object_msgs/Object message arrives.

Node 3 . Fake object recognition

The fake object recognition can be used to simulate recognition of some objects in the scene. For each object recognized; it will send out an object_msgs/Object message which can be received by the MoveIT! Collision object generator (Node 2).

Advantage object information pipeline

The object recognition should plug in with any other algorithm, the object recognition should be its own package which can be replaced individually.

The MoveIT! Collision object generator (Node 2) can work with any way the object information is obtained – be it with a fake object recognition as described, or a real object recognition.

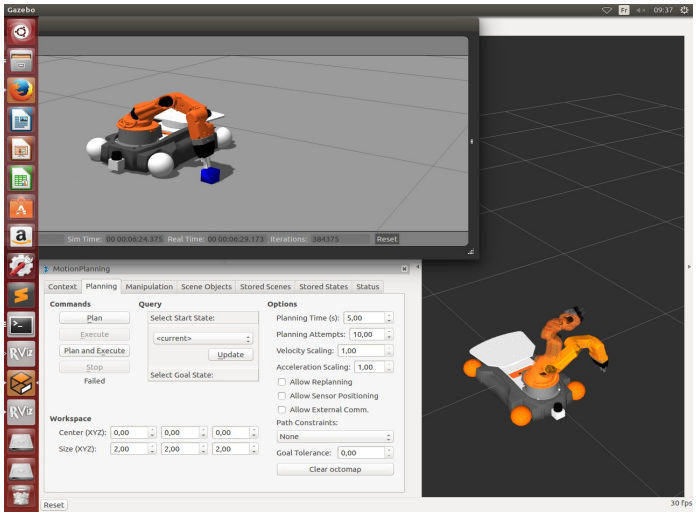


Illustration 22: Simulation in gazebo

Illustration 23: Picking with a KUKA Youbot

The plug in Moveit! for ROS interface was built during this project and at the same time along with navigation stack components.

Architecture Grasp Execution

The diagram below depicts the ROS services and action used by the automated grasp executor to achieve the grasping task.

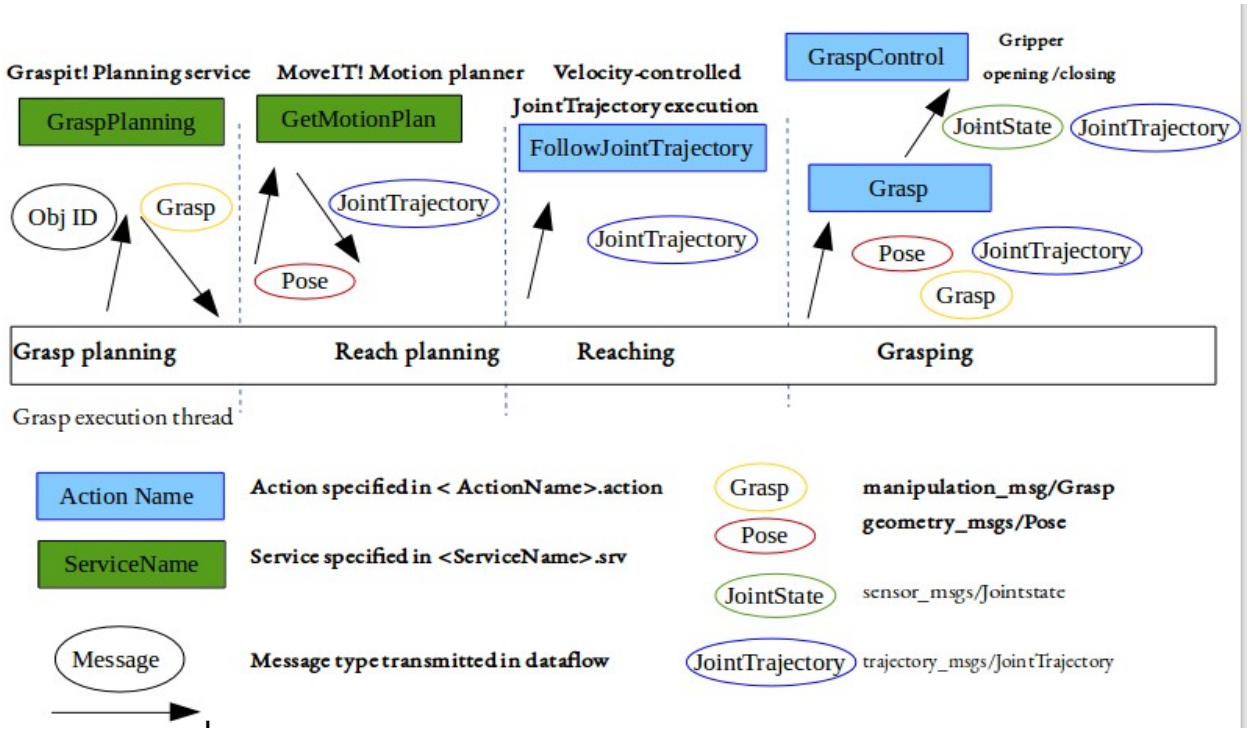


Illustration 24: Architecture Grasp Execution

Grasp planning: this could be any service to return eligible grasps, either from a database (MoveIT!) or by on-line grasp planning.

Reach planning: the pose object in the data flow in the end effector target pose.

Reaching: the joint trajectory execution should ideally be controlled by velocity commands, using the velocities specified in the joint trajectory.

Grasping: if the precondition are fulfilled, meaning the reach action was successful and the arm is at the required target pose for grasping.

Bibliography

http://www.ros.org/
http://wiki.ros.org/indigo/Installation/Ubuntu
https://moveit.ros.org/
http://www.youbot-store.com/wiki/index.php/Youbot_driver
https://github.com/JenniferBuehler
http://docs.ros.org/kinetic/api/moveit_tutorials/html/
https://github.com/JenniferBuehler/gazebo-pkgs/wiki/Object-information-and-recognition
https://github.com/JenniferBuehler/grasp-execution-pkgs/wiki/Object-information-pipeline
http://www.orocos.org/kdl
http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv
http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(c%2B%2B)
http://wiki.ros.org/navigation
http://wiki.ros.org/navigation/Tutorials/RobotSetup
http://wiki.ros.org/navigation/Tutorials/Navigation%20Tuning%20Guide
http://wiki.ros.org/navigation/Tutorials/SendingSimpleGoals
http://wiki.ros.org/gmapping
http://wiki.ros.org/amcl
https://www.orocos.org/wiki/orocos/itasc-wiki/itasc-tutorials/youbot-lissajous-tracing-tutorial
http://edu.gaitech.hk/turtlebot/map-navigation.html
http://edu.gaitech.hk/turtlebot/create-map-kenict.html#create-map
http://edu.gaitech.hk/ros/ros-programming-basics.html
https://github.com/ros-planning/navigation
https://github.com/ros-planning/navigation_tutorials
https://github.com/svenschneider/youbot-manipulation
https://github.com/FactoryOfTheFuture/Youbot
https://github.com/FactoryOfTheFuture/youbot_vision
www.openni.ru - An official documentation webpage.
structure.io/openni – A brief explanation on what is PrimeSense and its uses.
wiki.ros.org/face_recognition – A ROS documentation on the package that has been used
wiki.ros.org/find_object_2d – A ROS documentation on the package with various algorithms and concepts that are used for object detection.
answers.ros.org/question/256248/how-to-get-object-coordinates-to-grip-objects/ - A brief explanation how to create a transformation file to point the direction.