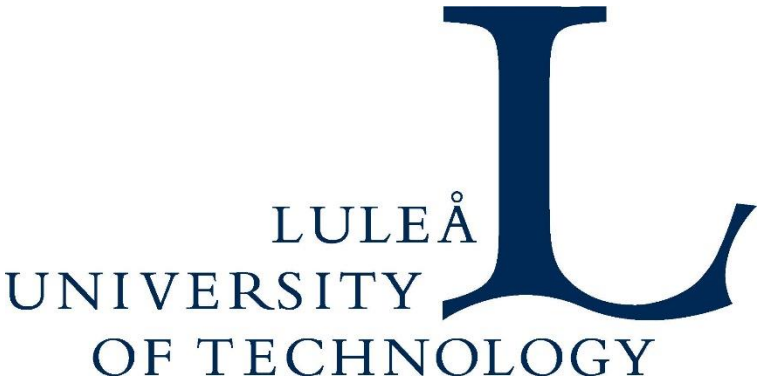
Integrating Baxter Robot in the Factory of the Future

Project in Computer Science – D7034E



Author

Alejandro Solanas Bonilla

Supervisor

Arash Mousavi Derazmahaleh

15 of January 2017

Abstract

This document introduces the current state of industrial robotics, focusing in the research aspect of a collaborative robot, the Baxter Robot. Baxter robot is the world's first dual arm collaborative robot, simple to train for repetitive production task such as discrete part handling, line loading, packing and unpacking, and many more applications.

The objectives are resumed not only in situating Baxter as a new generation of industrial robots, but to, analyse its capabilities, design a possible solution to integrating it into the Factory of the Future.

The project first objective is to create a working example on a production line, as how Baxter would react on a production environment, having complete knowledge about the functionality and capability of Baxter Robot.

Secondly, develop techniques for learning robot motion control techniques, that reduce the requirements placed on robotics experts, to increase the profusion of robot behaviours and promote robot autonomy. These techniques be accessible to non-experts as well. Both goals target the test of the capabilities of the Baxter Robot inside an industrial automation scenario, with the posterior improvement, if possible, of the control platform.

At first instance, an overview of how Baxter is situated at Robotics current world is given. Followed by a summary of its main components and capabilities, including the workstation setup and all the needs to have it ready. Relevant tools and technologies involved in the development and research are explained and justified.

Once presented the problem and the potential solution, a technical analysis of each aspect of Baxter Robot, designing a plausible scenario and how to implement it. Continuing with further details with the implementation of the solution, the execution of the code and a simple overview of the source files structure.

The report concludes with a conclusion in function of the accomplished objectives and a potential future work action.

Contents

Table of Figures

Introduction

The dictionary meaning of robot is a mechanical man or a more than humanly efficient automation. It is an automatic apparatus or device that performs functions ascribed to human beings or operates with what appears to be almost human intelligence. But this defamation does not give a human shape to the robot. The robot does the work of a human being.

This document introduces the current state of industrial robotics, focusing in the research aspect of a collaborative robot, the Baxter Robot. Baxter robot is the world's first dual arm collaborative robot, [1] simple to train for repetitive production task such as discrete part handling, line loading, packing and unpacking, and many more applications. Not to mention the research and evaluation of the capabilities of Baxter Robot, as a component of a production line in the factory of the future.

The aim of this project is to, not only situating Baxter as a new generation of industrial robots, but to, analyse its capabilities, design a possible solution to integrating Baxter robot into the Factory of the Future, and explaining how to implement this solution. This solution will simplify the use and setup of the Baxter, and the execution of simple demo programs.

Through this process, the Baxter's setup, technologies involved, detailed analysis of the system and the chosen implementation, are to be detailed explained. Afterwards, the execution and code of the solution, characteristics and how to properly execute it. More details about the code and the source files will be found at the github repository [2]

Finally, this document will give a conclusion of the Baxter and the objectives accomplished, followed with a potential future work actuation.

Figure 1 - Baxter Robot in AIC3 Laboratory

Literature Review

For years, automatic machines have been performing labour-saving work and efficiently repeating tasks such as churning out millions of one types of part. Unfortunately, automatic machines are not suited to small batch work since physical changes must be made to the machine in order to perform a different task. Robots, on the other hand, are flexible and can be easily switched to perform different jobs by simply changing the robot control program.

Automation is a central competitive factor for traditional manufacturing groups, but is also becoming increasingly important for small and medium-sized enterprises around the world [3]. The versatility of smart and collaborative robots makes them a perfect for a wide range of industries and applications such as, CNC machining, metal fabrication, packaging, test and inspection, PCB handling and ICT, molding operations, and loading/unloading parts.

As far as technological trends are concerned, companies will, in the future, be concentrating on the collaboration of human and machine, simplified applications, and light-weight robots. Added to this are the two-armed robots, mobile solutions and the integration of robots into existing environments. There will be an increased focus on modular robots and robotic systems, which can be marketed at extremely attractive prices. As an example of the two-armed robots is the Baxter Robot [1]

Introduction to Baxter

Baxter robot is the world's first dual arm collaborative robot, [1] simple to train for repetitive production task such as discrete part handling, line loading, packing and unpacking, and many more applications. Featuring two 7 degree of freedom arms, each with a maximum reach of 1210mm, Baxter can be deployed to work on two independent tasks to increase versatility, or simultaneously on the same task to maximize throughput. Baxter is safe next to people on a production line, without the need for caging. See imagen below for extended features.

Whereas traditional industrial robots perform one specific task with superhuman speed and precision, Baxter is neither particularly fast nor particularly precise, but it excels at just about any job that involves picking stuff and placing it somewhere else while simultaneously adapting to changes in its environment, like a misplaced part or a conveyor belt that suddenly changes speed. This is the Baxter main strong characteristic

There are two other major barriers to the adoption of industrial robots that Rethink wants to overcome: ease of use and cost. As for the first, Baxter doesn't rely on custom programming to perform new tasks. Once it's wheeled into place and plugged into an ordinary power outlet, a person with no robotics experience can program a new task simply by moving Baxter's arms around and following prompts on its user-friendly interface (which doubles as the robot's face).

While a traditional two-armed robot, including sensors and programming, will typically set you back hundreds of thousands of dollars, Baxter costs just $22 000. To achieve that, Rethink designed the robot from scratch. [4]. This case studio considers the Baxter Research Robot Product with the basic accessories, the electric parallel grippers, and the mobile pedestal, not the industrial version.

To summarize, Baxter is a humanoid, anthropomorphic robot sporting two seven degree-of-freedom arms and state-of-the-art sensing technologies, including force, position, and torque sensing and control at every joint, cameras in support of computer vision applications, integrated user input and output elements such as a head-mounted display, buttons, knobs and more. There are 2 versions of Baxter, Baxter for Manufacturing and the Baxter Research Robot. The **Baxter Manufacturing Robot** is delivered pre-installed with the Intera software, allowing Baxter to work seamlessly in manufacturing settings. The data sheet for the Baxter Manufacturing Robot can be found at the link below[1]

Figure 2 - Baxter Overview [5]

[1] http://sdk.rethinkrobotics.com/mediawiki-1.22.2/images/6/65/BaxterDataSheet.pdf

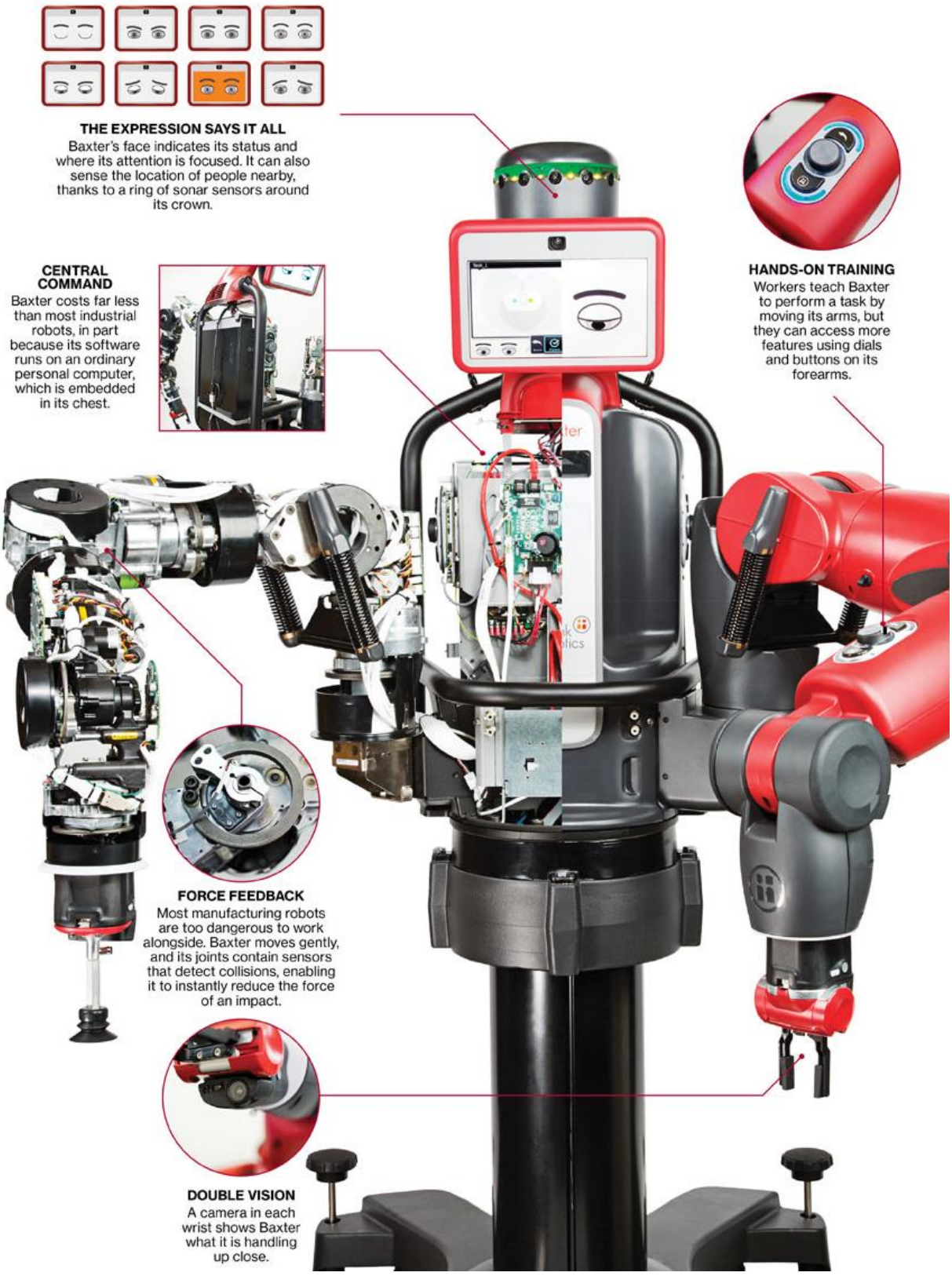


Figure 3 - Baxter review sheet [1]

Setting up the Robot

The Baxter setup must be explained in two phases [6], the Hardware or physical setup and the development workstation setup. Also on-robot workspace setup and running code on-board is available[2].

- **Hardware Setup**: [7]

  The hardware Setup is direct. Just find a correct place where the Baxter can operate freely, and mount the Baxter on the pedestal. After adding the desired accessories and plug it. All the detailed info can be consulted at the referenced website[3].

- **Workstation Setup**: [8]

  The workstation setup can be detailed into several steps, but before that it is recommended to check the minimum system requirements. It is suggested to follow the installation steps with the website[4], additionally for this project, an installation script is available at the repo at Github [2]. The following steps can be resumed as:

  1. Install Ubuntu: 14.04 (recommended) or 12.04 required versions. Also, you can choose a different linux distro or version at your own risk, having in mind that ROS must be available. Also, python and c-compilations tools should be installed.

  2. Install ROS: Indigo(Recommended), Hydro or Groovy.
     a. Configure Ubuntu repositories
     b. Setup sources.list
     c. Setup keys
     d. Verify Latest Debians
     e. Install Ros Indigo Desktop Full
     f. Initialize rosdep
     g. Install rosintall

  3. Create Baxter Development Workspace: Folder with the source files.
     a. Create ROS workspace
     b. Source ROS and Build

  4. Install Baxter SDK Dependencies

[2] http://sdk.rethinkrobotics.com/wiki/SSH
[3] http://sdk.rethinkrobotics.com/wiki/System_requirements
[4] http://sdk.rethinkrobotics.com/wiki/Workstation_Setup

5. Install Baxter Research Robot SDK
    a. Install Baxter SDK
    b. Source ROS Setup
    c. Build and Install

6. Configure Baxter Communication/ROS Workspace: An Ethernet network must be established first between Baxter and the Workstation with full bi-directional connectivity[5]. You can also connect the Baxter to a Wi-Fi network and connect via IP address.
    a. Download the baxter.sh script
    b. Customize the baxter.sh script
    c. Edit the 'baxter_hostname' field
    d. Edit the 'your_ip' field
    e. Verify 'ros_version' field
    f. Save and Close baxter.sh script
    g. Initialize your SDK environment

7. Verify Environment

Once, both hardware and workstation are ready, is recommended to check the "Hello Baxter" [9] section where is explained how to communicate with Baxter, moving the arms, and allowing him to say "Hello!" to your lab.

A video with the complete setup can be also found at Rethinks Robots website[6]. The workstation used in this project can be seen in the Figure 4 - Baxter Workstation & Baxter Robot. Any other configuration following the setup process should be correct.

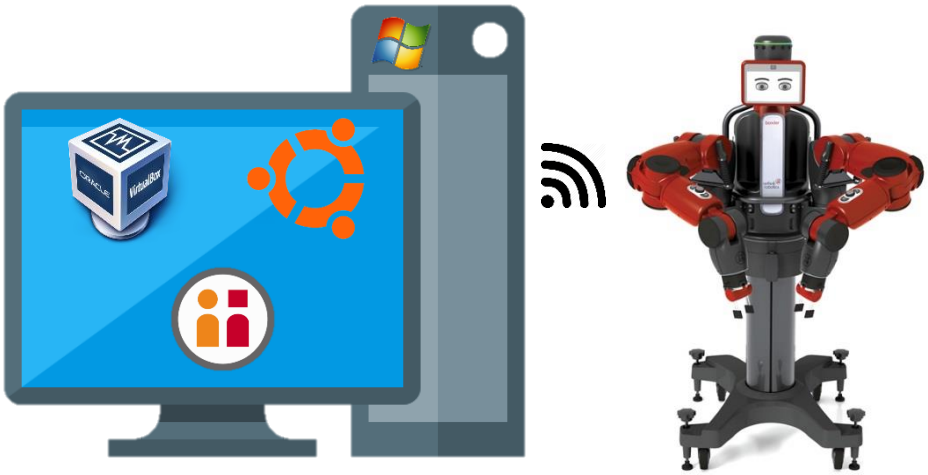


Figure 4 - Baxter Workstation & Baxter Robot

[5] http://sdk.rethinkrobotics.com/wiki/Networking
[6] https://youtu.be/2QfP4qY_7_c

Programming tools

- **Python**:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics [10]. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

So, it must be considered what qualities or capabilities of python makes it a good programming language for robotic engineering. There are few languages better suited to research robotics programming than Python [11]:

- No build process, simple syntax dynamic typing
- It interfaces with C/C++, so many python libraries are just as fast as their C counter parts, because they are their C counterparts.
- It has a huge number of libraries
- It's (mostly) cross-platform, depending on the libraries you're using
- It's a core language of ROS (Robot Operating System), meaning the full power of a distributed robotics system and all its libraries/tools are available to you via Python

In this project, python will be the programming language used to develop with Baxter, alongside with ROS.

- **ROS:**

The Robot Operating System (ROS) is an open-source, meta-operating system for your robot [12]. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS is similar in some respects to 'robot frameworks,' such as Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio.

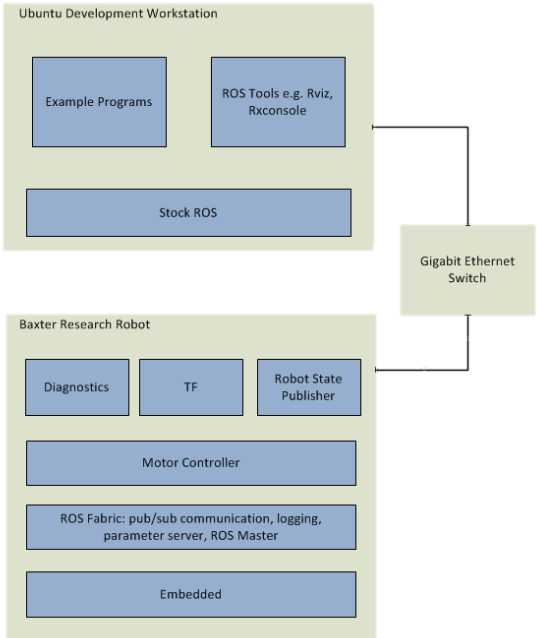


Figure 5 - Research SDK Block Diagram [13]

ROS currently only runs on Unix-based platforms. Software for ROS is primarily tested on Ubuntu and Mac OS X systems, though the ROS community has been contributing support for Fedora, Gentoo, Arch Linux and other Linux platforms.

- **Baxter SDK**:

The Baxter Research Robot SDK provides a software interface allowing researchers of all disciplines to develop custom applications to run on the Baxter platform [13]. The SDK interfaces with the Baxter Research Robot via ROS (Robot Operating System). Baxter provides a stand-alone ROS Master to which any development workstation can connect and control Baxter via the various ROS APIs[7].

The ROS Interface is the foundation upon which all interaction with the Baxter Research Robot passes through and all other interface layers are built upon. The ROS layer is accessible over the network via any of the standard ROS Client Libraries such as rospy (Python) or roscpp (C++).

The Baxter SDK also offers a Python API via the baxter_interface module. This module wraps the ROS Interfaces in component-based Python Classes. The SDK Examples are written using this library in order to demonstrate how to use the robot interfaces.

- **Virtual Box**:

VirtualBox is a cross-platform virtualization application [14]. What does that mean? For one thing, it installs on an existing Intel or AMD-based computers, whether they are running Windows, Mac, Linux or Solaris operating systems. Secondly, it extends the capabilities of an existing computer so that it can run multiple operating systems (inside multiple virtual machines) at the same time. So, for example, it can run Windows and Linux on a Mac, run Windows Server 2008 on a Linux server, run Linux on a Windows PC, and so on, all alongside the existing applications.

In this project, this application is used to prepare the workstation and to have different platforms to test the Baxter controls.

- **Additional Tools**
    - PyCharm: Python IDE for developers[8]
    - Atom: Hackable test editor[9]
    - MoveIt: Software for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics,

[7] http://sdk.rethinkrobotics.com/wiki/API_Reference
[8] https://www.jetbrains.com/pycharm/
[9] https://atom.io/

control and navigation. ROS application ready to work with Baxter Robot. Check the MoveIt tutorial for Baxter[10]

- Gazebo: 3D dynamic simulator with the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. While similar to game engines, Gazebo offers physics simulation at a much higher degree of fidelity, a suite of sensors, and interfaces for both users and programs. Baxter simulator, Gazebo[11]
- RViz: 3D visualizer for displaying sensor data and state information from ROS. Using RViz, you can visualize Baxter's current configuration on a virtual model of the robot. You can also display live representations of sensor values coming over ROS Topics including camera data, infrared distance measurements, sonar data, and more.
- Kinect: Line of motion sensing input devices my Microsoft. Baxter Robot has one in each hand's camera. It's possible to integrate depth sensor data with MoveIt![12] This will allow MoveIt to plan paths for Baxter in dynamic environments.

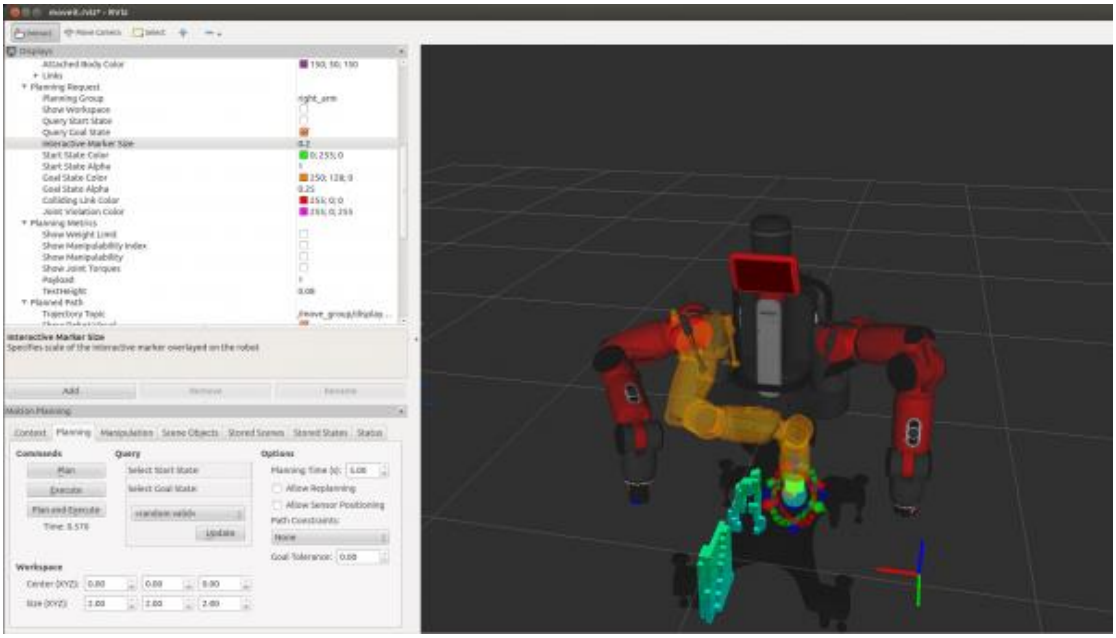


Figure 6 - MoveIt + Kinect

- Kivy: Cross platform Open Source Python library[13] for rapid development of applications that make use of user interfaces.

[10] http://sdk.rethinkrobotics.com/wiki/MoveIt_Tutorial
[11] http://sdk.rethinkrobotics.com/wiki/Baxter_Simulator
[12] http://sdk.rethinkrobotics.com/wiki/Kinect_basics
[13] https://kivy.org/#home

Problem statements and objectives

Motion control is fundamental to many robotics applications, and is known to be a difficult problem. There are many contributing factors to behaviour execution that are potential targets for improvement, for example improved hardware to provide more precise sensor readings or more accurate action executions.

Baxter Robot is presented as one of a new kind, more secure and specialised in simple and not so precise actions. A primary goal of this research is to develop techniques for learning robot motion control techniques, that reduce the requirements placed on robotics experts, to increase the profusion of robot behaviours and promote robot autonomy.

A secondary goal is that these techniques be accessible to non-experts as well. Both goals target the test of the capabilities of the Baxter Robot inside an industrial automation scenario, with the posterior improvement, if possible, of the control platform.

Proposed solution

Two main objectives conform the solution to this research. First one, developing a sample program or programs, with the purpose of testing the capabilities of the Baxter Robot. The scenario will emulate the end of a complete fabrication process, where Baxter will take the final product, dissemble it and put each part in separate places for storage or transportation purposes.

The second one, simplify the control and execution process, creating a platform if necessary, overcoming the limitations of the system and the workstation. The desired result would be a cross platform, probably with a simple GUI making it accessible to non-experts in the field.

Additionally, evaluate the existing simulating tools, fitting their capabilities to the mentioned scenario.

Objectives

- Become familiar with Baxter Research Robot, Standard Operational Tools, Supported External Tools, and resources.
- Setup a complete workstation evaluating the needs and the complexity of the system
- Study the Baxter API capabilities and complementary useful tools.
- Create sample scripts for testing different and useful features of the Baxter
- Evaluate the diverse ways to run a program
- Develop a cross-platform accessible to all users

Analysis

Before starting the design of the problem stated, all the basics and advanced concepts related to the Baxter must be known with the purpose of state a solution having in mind both limitations and capabilities. As mentioned before, Baxter can be operated via ROS API or the Python API (ROS wrapper). The Python API suits better our objectives because not only the same features that exits on ROS are available, but all the capabilities of the programming language.

Hardware

The Baxter Research Robot platform consists of robot hardware, software (delivered via the software development kit (SDK)), and several communication tools to link like-minded researchers.

The robot, as its core can be described in 3 main aspects[14]:

- Robot Components:
  - **Robot**: In order to control Baxter's arms or head, the robot must be entered into the 'Enabled' state. Enabling the robot provides power to the joint motors, which are initially in the 'Disabled' state on start-up or after a serious error, such as an E-Stop (Emergency-Stop). Robot enabling/disabling can be commanded programmatically using the baxter_interface's RobotEnable class. A convenient command line tool is provided for quickly enabling/disabling Baxter. This tool is provided in the baxter_tools package: Enable Robot Tool
  - **Arms**: Baxter's has two seven degree-of-freedom (DOF) arms. Seven DOF arms are desirable as they provide a kinematic redundancy greatly improving manipulability and safety. There are several control systems for controlling Baxter's arms and adding robustness or safety to Baxter. Joint Position, Joint Velocity, and Joint Torque control modes exist for commanding Baxter's arm for your specific application.

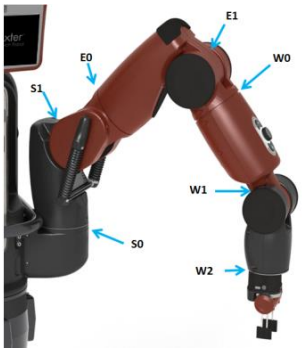


Figure 7 - Joints [15]

  - **Grippers**: Electric and suction grippers.
  - **Cameras**: The Baxter Research Robot has three colour cameras; two are in each arm, and the third camera is located on the head, above the display screen. Any two cameras can be operated at a given time; USB system limitations do not allow all three cameras to be operated at once. Due to 64-bit limitations with the Indigo Baxter's internal USB hub, only two cameras can be powered on at once.

[14] http://sdk.rethinkrobotics.com/wiki/Advanced_Understanding
[15] http://sdk.rethinkrobotics.com/wiki/File:Baxter_arm.png

- **Head**: Baxter's head has a pan joint and a single "Nod" action for movement. There is one camera, one red LED ring, one green LED ring, and 12 individual yellow LEDs along with 12 sonar transducers. You can control the lights on the head and the sonar transducers themselves.

- **Input & Output**: Overviews the distinct types of IO, starting from the top-level: including, sensors buttons and switches, lights, then navigators, then digital and analog io.

- **Screen**: The Baxter Robot head display is a 1024 x 600 SVGA LCD screen, and is published to through the /robot/xdisplay topic.

- Component Guides & tasks:
  - **Robot State and Estop**: The "Robot State" refers to the state variable that shows whether the robot (motors) are enabled or disabled, if the e-stop is engaged, and if there are any hardware errors.

  - **Sonar Control**: Around the head is a ring of 12 sonar sensors for detecting movements within Baxter's proximity. Next to each sonar sensor are yellow LEDs, which by default indicate when the corresponding sensors detect an object within a certain threshold. You can control the sonar sensors and lights to override the default behaviour and either disable or enable each of the 12 channels.

  - **URDF** (Robot description): The Unified Robot Description Format (URDF) is the standard ROS XML representation of the robot model (kinematics, dynamics, sensors) describing Baxter. Baxter generates his URDF dynamically on robot startup. This model is updated when any gripper is attached or detached, an object is 'grasped' or released and its mass is compensated for, and when new urdf segments are provided/commanded to the gripper plugins.

  - **Gripper Customization**: Using URDF the robot description can be modifiable at runtime, allowing to add joints and linkages to the grippers. Also, the user is allowed to set an expected mass for grippers as well as for grasped objects.

- Arm Control Systems:
  - **Arm Control Overview**: Baxter's arm controls flow through on four layers of operation:
    1) User Code running via workstation or SSH (Python Joint Control or Joint Trajectory Action Server or your custom interface)
    2) Joint Control Listeners via ROS topic (on Baxter's internal Gentoo Linux PC)
    3) RealTime Motor Control Loop (the highest priority process on Baxter's internal Gentoo Linux PC)
    4) Joint Control Boards (microcontrollers attached to each arm joint)

The bottom three layers (2-4) are not accessible for user modification because they run on the robot itself and its microcontrollers. Baxter does have three joint control modes for the user to choose from at the top level: position, velocity, and torque control.

Each controller has a subset of Motor Controller Plugins associated with them. Start to finish, a joint is commanded from the user's python code. This message flows over the ROS network and is picked up by the Joint Control Listeners which store the command for it to be retrieved asynchronously by the RealTime Motor Control Loop

- **Arm Control Modes**: Fundamental to the operation of Baxter via the SDK is a familiarity with the available joint control modes. The joint control modes define the 'modes' in which we can control Baxter's limbs in joint space. When a joint position command is published from the development PC, the 'realtime_loop' process which represents a motor control plugin subscribes to this message. This message is then parsed, and represented in memory based on control mode.

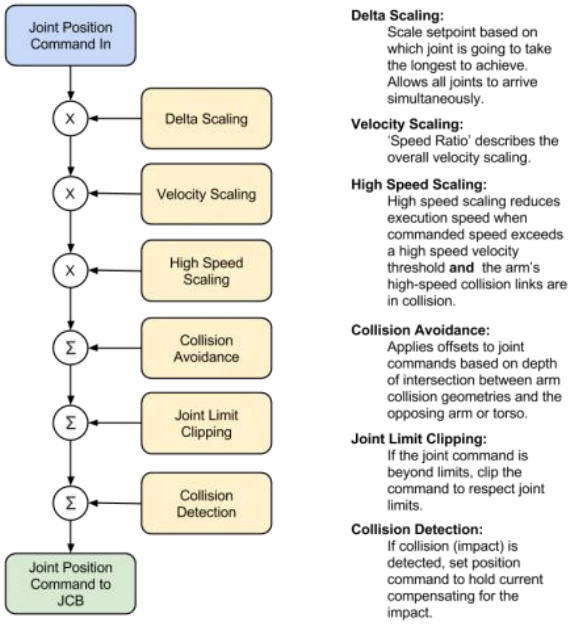


Figure 8 - Joint position control mode

Joint position control mode is the fundamental, basic control mode for Baxter arm motion. In position control mode, we specify joint angles at which we want the joints to achieve. Typically, this will be consisting of seven values, a commanded position for each of the seven joints, resulting in a full description of the arm configuration.

- **Collision Avoidance and Detection**: There are a total of three levels of collision models in baxter. The first, each of Baxter's link has its own collision block which is slightly bigger than the size of that link. When these blocks encounter each other, collision avoidance model is triggered. The collision blocks can be visualized in RVIZ.

  The next collision model involves the detection of two types of collisions - impact and squish. Impact is detected when a sudden change in torque is sensed across any of the joint. Squish is detected when joint tries to press against a stationary object. For instance, when a robot arm tries to push a wall, the torque applied across the joint increases and it immediately stops when the applied torque is greater than a threshold.

The final collision model is high speed scaling. This is to avoid the collisions between two limbs moving at higher speeds. In order to avoid the collision, the corresponding velocities are scaled down and converted back as position commands.

- **Gravity compensation**: In order to oppose the effect of gravitational force acting across Baxter's arm, gravity compensation torques have to be applied across that arm. This is a basic mode that is active from the time the robot is enabled. The built-in gravity compensating model uses KDL[16] for calculating the gravity compensation torques. The springs attached to the S1 joint also require compensation torques, these are found using an internal spring model and applied in conjunction with the torques from KDL

- **Zero-G Mode**: Zero-G mode can often be confused with the mode obtained by disabling the gravity compensation torques. By default, the gravity compensation torques will always be applied when the robot is enabled. In Zero-G mode, the controllers are disabled and so the arm can be freely moved across. In this case, the effect of gravity would be compensated by the gravity compensation model applying gravity compensation torques across the joints, there would be no torques from the controllers since they would not be active, and so the arm can be moved freely around, hence the name. The Zero-G mode can be enabled by grasping the cuff over its groove. Unlike the gravity compensation model, the Zero-G mode cannot be directly disabled. The cuff interaction must be disabled in order to disable the Zero-G.

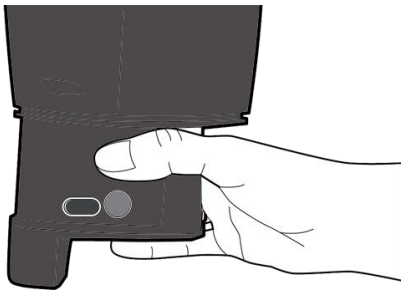


Figure 9 - Zero-G

- **Joint Trajectory Action Server**: Baxter's SDK comes equipped with Joint Trajectory Action Server (JTAS) to facilitate users needing to command either (or both) of Baxter's arms through multiple waypoints. These waypoints are supplied as positions, velocities, and/or accelerations for every joint in Baxter's arms. These waypoints are accompanied by a desired timeframe within which the trajectory must complete. The main benefit of this server is its ability to interpolate between supplied waypoints, command Baxter's joints accordingly, and ensure the trajectory is being followed within a specified tolerance.
  The JTAS has three modes: Velocity, Position, and Raw Position. Velocity mode executes direct control of trajectories by calculating and commanding the velocities required to move the arm through a set of joint positions at a specified time. The result is fast, but less accurate movement.

[16] http://www.orocos.org/kdl

Position mode allows for simple trajectory execution of position-only trajectories. This mode is significantly more accurate than velocity mode, at a cost of a speed reduction. Raw position can allow for the highest speed movement with the most accuracy. By supplying desired Positions, Velocities, and Accelerations for each joint, you give Baxter to anticipate and correct for the dynamics of its movement by "feeding forward" Velocities and Accelerations to the Motor Controller. This enables Baxter to use both feedback from its spring deflection sensors, and feed forward from dynamics of the anticipated trajectory to accurately execute trajectories.

SDK

The SDK provides interfaces that control the robot hardware layer via the Robot Operative System (ROS) to any other computers on the robot's network. By using the ROS network layer API, any client library or program that can "speak ROS" can control the robot directly. For a deeper understanding of the SDK System architecture see the Figure 5 - Research SDK Block Diagram.

Throughout the Baxter setup and the mentioned SDK, simple environment concepts like "RSDK Shell", "ROS Workspace" or "Environment Variables" are useful to have a basic understanding of when using the Baxter SDK and ROS.

- ROS Workspace

  Your "ROS Workspace" (also called a "Catkin Workspace" in the ROS world), is the ~/ros_ws/ setup. Inside the src/ sub-directory, you can create or checkout source code as ROS Packages (such as the baxter sdk packages), which you can then compile and link against via the devel/ or install/ sub-directories.

- RSDK Shell

  The RSDK Shell is a term used to refer to a session in a shell, or Terminal, with the ROS Environment Variables properly configured to point to the ROS Workspace, the Baxter Robot, and the computer's ROS ID. The baxter.sh file is the setup script to take care of the configuration.

  A proper RSDK Shell is needed before use most of the ROS tools or run any of the SDK examples or programs. Therefore, the ./baxter.sh script should be run at the start of each Terminal session. All the script does is start a convenient "sub-shell" that fills in the necessary Environment Variables as you configured.

- Environment Variables:

  The ROS Environment Variables have three essential functions:

  Identifying your Baxter and Workstation on the ROS Network, based on your Network Configuration. Pointing to the path of the ROS Packages, tools,

and programs, when you use rosrun (execute programs). Linking to the libraries and workspaces when compiling source code.

Complementary to de SDK, the repository on github[17] contains several tools and examples that are worth mentioning, because some of these scripts would be necessary to implement the final solution. Also, the scripts to launch a RSDK Shell or to enable/disable, calibrate and control most of the Baxter features are accessible here. This repo will be located inside the ROS workspace after a successful setup.

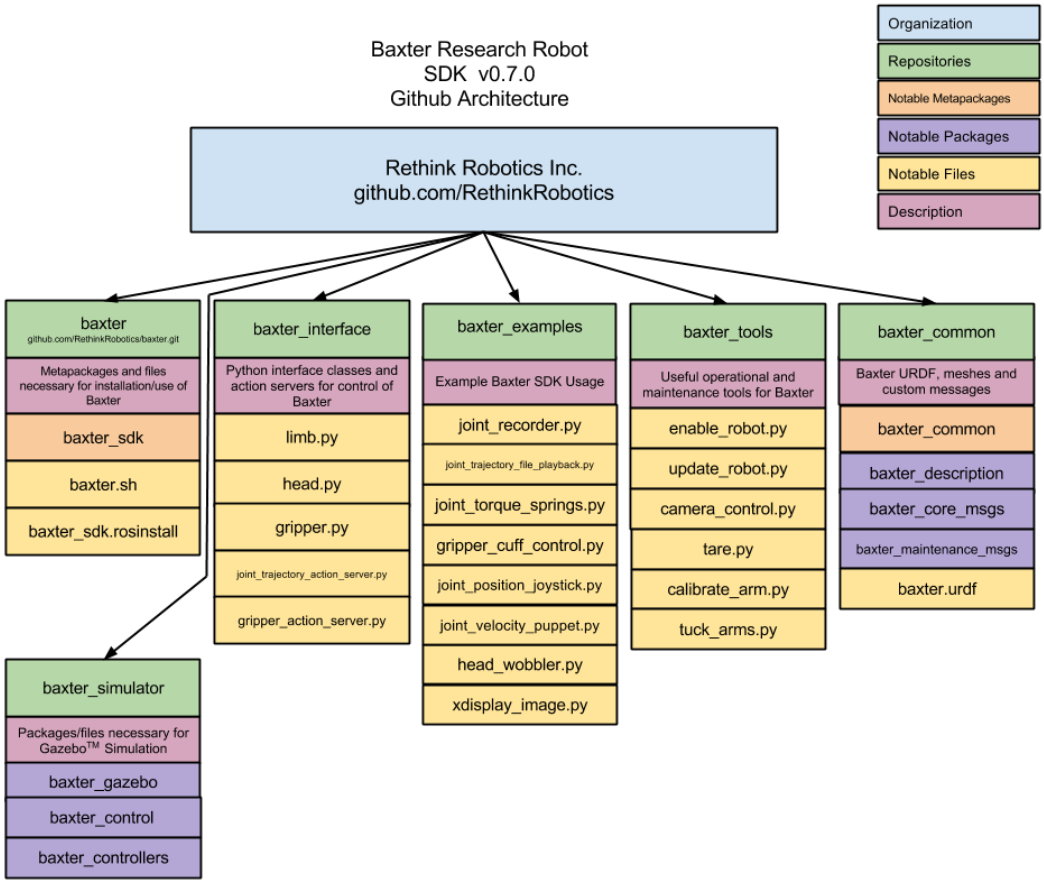


Figure 10 - Repository Architecture

The baxter sub repository contains metapackages and files for installation/use of the Baxter SDK. These include the baxter_sdk metapackage which contains all of the Baxter SDK packages. Also included in this repository are rosinstall files for ease of installation, and baxter.sh, a bash script designed to help users easily setup their ROS environment as quickly as possible for use with Baxter.

The baxter_tools sub repository contains useful operational and maintenance tools for use with Baxter. These scripts prove useful for everyday use (robot enable/disable, camera control) and maintenance tasks (tare, calibrate, update robot software).

[17] https://github.com/RethinkRobotics

The baxter_interface sub repository contains python interface classes and action servers for control of Baxter. These python modules extract the direct ROS communication with the robot allowing users to easily program Baxter without having intimate knowledge with ROS. Also included in this repository is the joint trajectory and gripper action servers. These are standard action servers for use with popular ROS tools such as MoveIt!

Behaviour

To start working with Baxter robot, with a ready workstation, execute the already configured baxter.sh script in a terminal. Baxter and workstation must be able to connect via IP, either being on the same network or configuring an external connection.

Once an instance is running, ROS can be used to give order to the Baxter, using pure ROS code or the Python API. Also, this instance is needed to use and launch graphical tools like MoveIT, RViz, or Gazebo.

One important fact about this script, is that it allows only one single running process. If multiple processes are needed, another terminal with a baxter.sh instance must be opened. This limitation translates into partial functionality for a user interface, because when the server is ready and listening, even if an event is triggered to execute some action, and a new thread is created for it, it can't run asynchronously, so no time limit process will block the entire server.

Design

Evaluating the disassembly program, can be explained in two main states that will be sequentially simplified in more simple steps. From the state that the robot is ready and no action have been effected so far, to the stated that the robot has disassembled the pieces and drop them in their respective position. These states need to split in several sub states which can be resumed in the following:

- Baxter is ready and in neutral position (or desired initial position)
- Arms move to catch position
- Catch the item
- Disassembly position
- Disassembly process
- Storage position
- Drop pieces
- Back to neutral position

Baxter robot, even having 7 degrees of arms freedom, is not capable of performing all kind of movements and operations. It has its own limitations and it's

needed to have this fact in mind for the implementation. Due to Baxter arms movement limitation as mentioned, or how the Baxter automatically calculate the movement from one position to another, the planned movement may not be achievable. To solve this problem, instead of going from State X to State X+1, this process has to be subdivided in two or more substates to make the movement more precise, or adjusting the initial or final position.

Besides considering the execution process, two main approaches can be differenced to implement the disassembly program. Using the Python API or using a simulation plus control tool like MoveIT. Both approaches have been evaluated and tested, and these are the results of each.

Using the Python API, requires the manual record of the desired positions and actions to reach, on for each mentioned state, or if the position is exactly known, this could be translated to Baxter Kinematic model. A control tool like MoveIT, or a simulator tool like Gazebo gives a more visual approach, but it requires a pre-planned movement actuation. So, for this case, where just testing the capabilities of the Baxter, recording the positions it's a better approach.

Concerning the cross platform, it has been chosen to develop a user interface on Python using a client-server model using sockets, taking advantage that the API to control the Baxter is also in Python.

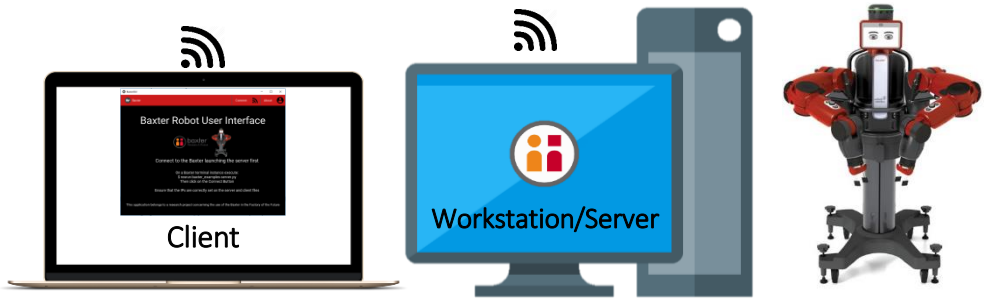


Figure 11 - Application Control Diagram

Implementation & Code

Control Application

The application is developed with Python using a client-server model, and Kivy for the user interface at the client side. The application runs in any kind of system that can run Python. The execution process can be consulted at the next headland. After a correct launch, this is how it looks:

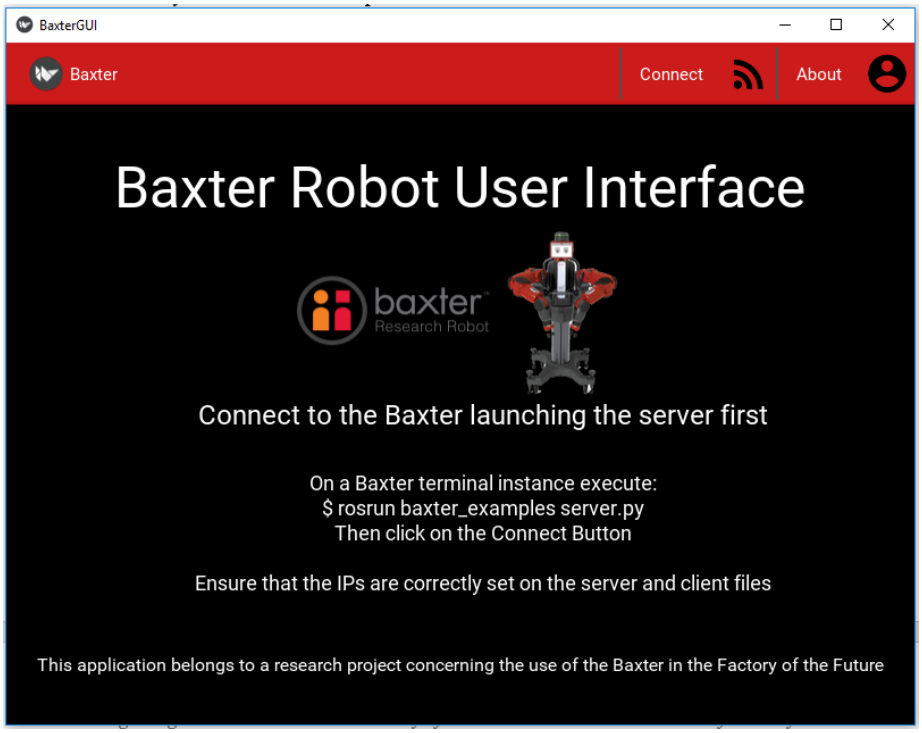


Figure 12- Connection Screen

This is the starting screen of the application, from here a small description about the project shows up, and we can navigate using the top bar with the connect and about buttons. The connect button will need the server up and running, and both client and server have their IPs correctly configured. This button will swap to the control screen where programs can be executed and are classified in two aspects, essential, equal or modified scripts from the scripts already available on the repo, and the custom scripts, developed by the user.

Clicking on each program will give a small description about the program functionality, and the correspondent execution button. The main script for showing the Baxter capabilities is the Open Cylinder option, under the custom scripts tab. The Baxter essential tab, includes demos from the rethink repository.

From this point, these are some pictures resuming how the important aspects of the application are resumed:

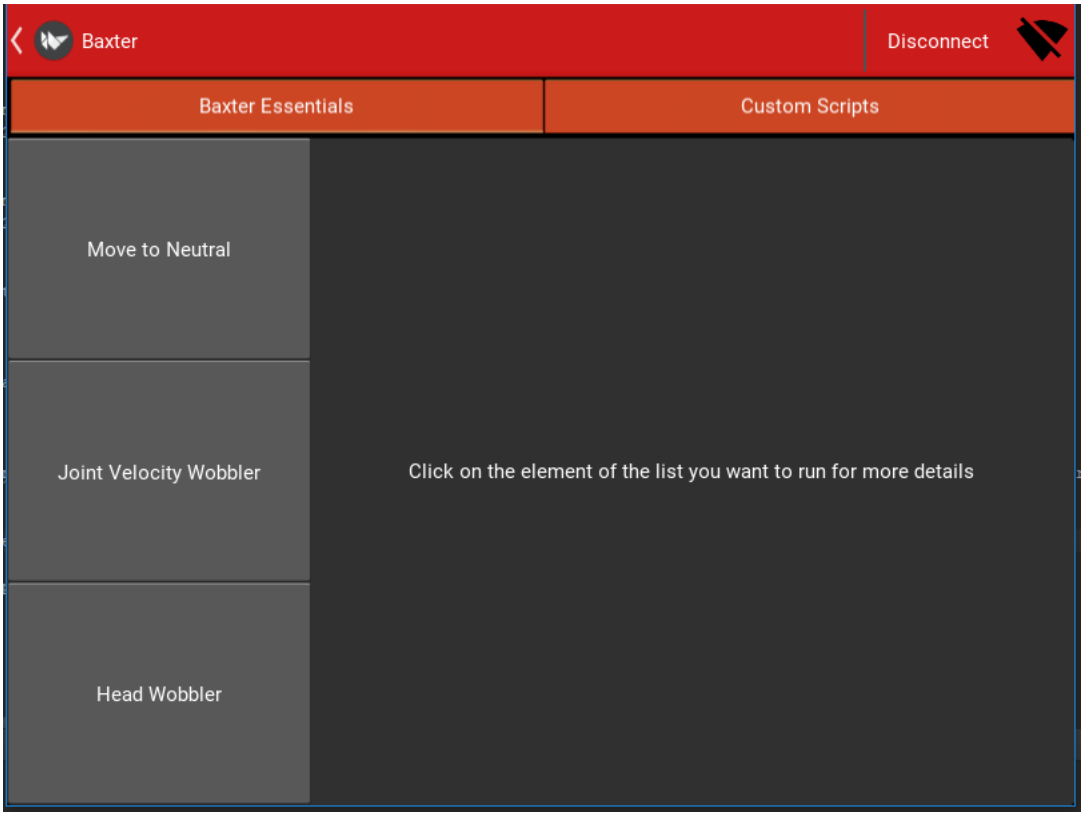


Figure 13 - Baxter Essentials Screen

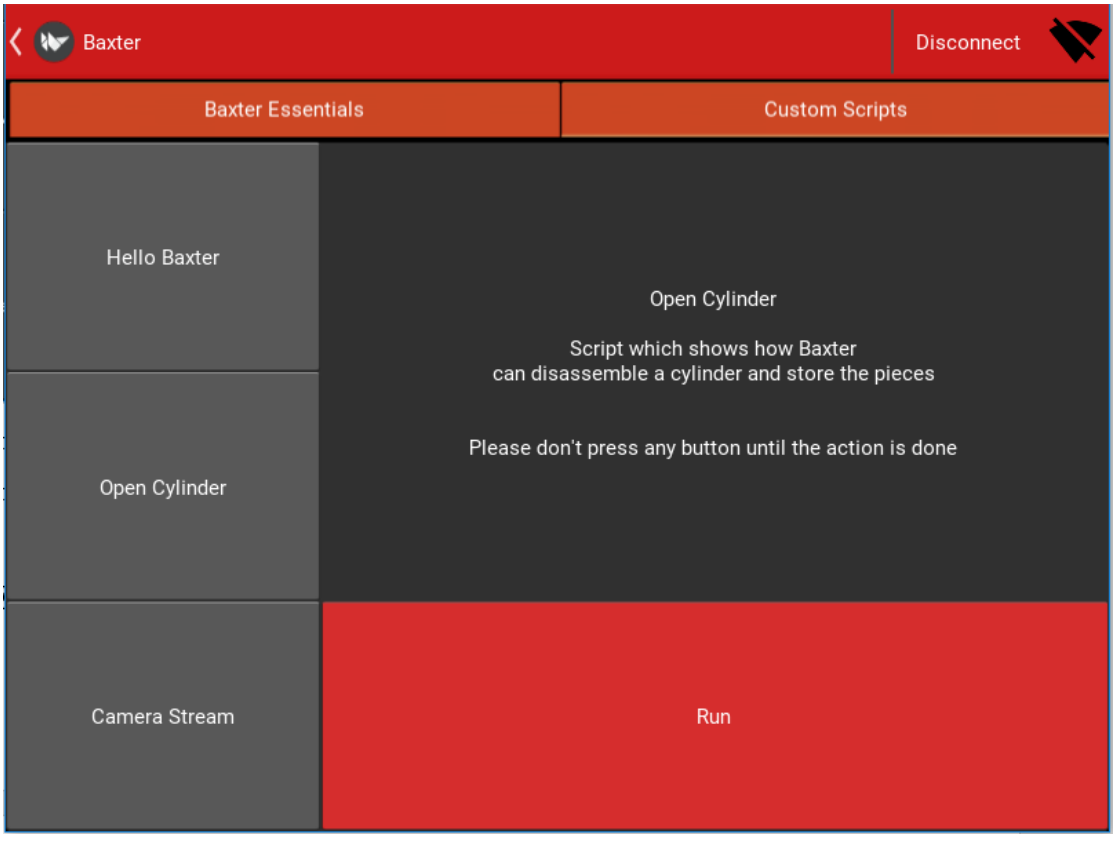


Figure 14- Open Cylinder Screen

Execution

The execution process can be resumed in the following steps:

1. Assuming the Baxter is working and connected to the network, and the baxter.sh is configured, from the workstation, in a terminal, run the baxter.sh script to start the ROS instance.
2. Ensure that the scripts wanted to execute are placed in the desired route inside the ros_ws folder, e.g: baxter_examples/scripts
3. Enable the robot using "$ rosrun baxter_tools enable_robot.py –e" or executing the given script startStop.sh founded inside de scripts folder. This scripts also works for disabling the Baxter if it's already enabled.
4. Run the script at your choice via rosrun. As already mentioned the scripts should be placed inside the source folder ros_ws. It's recommended putting them inside $ ros_ws/src/baxter_examples.
   E.g. the server can be run with "$ rosrun baxter_examples server.py", but also, a script called server_run.sh is available
5. Once the server is up and running, execute de Python script client.py in the system at your choice. For the application to work, the IP of the server must be known to correctly configure the client.
6. From the client, connect to the server and execute the Open Cylinder script

Sources

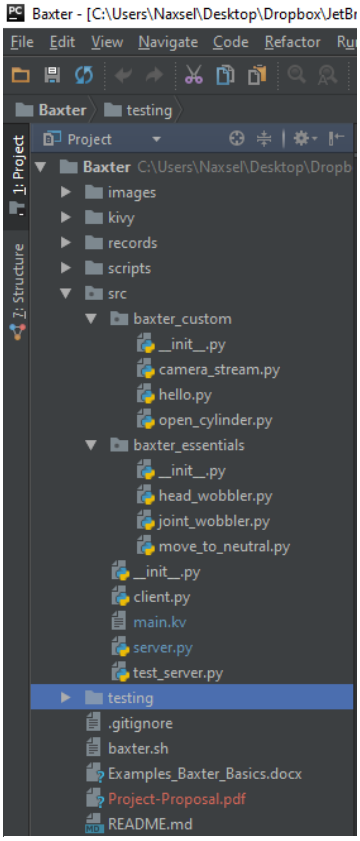


Figure 15 - Code Structure

The source files can be founded at the project github repository [2]. The image on the left resumes the final structure and main files of the project:

- Images: folder where the images for the UI are stored
- Kivy: Test examples of some functions and capabilities of the Kivy framework
- Records: Text files saving several positions using to program the open cylinder script
- Scripts: useful scripts for installation or execution
- Src: Main folder of the application: client.py and main.kv conforms the user interface, server.py and both subfolders contains all the server logic.
- Testing: General testing folder, prototypes and experimental approaches.

Inside de root folder the README.md contains an introduction to the project and several instructions of how execute it.

The baxter.sh file, is the same file mentioned on the Setup Workstation chapter

Conclusion

Finally, all the Baxter main working aspects has been explained and analysed, giving a detailed overview of the capabilities and integration with the Factory of the Future, as a new generation of safe collaborative robots.

A program showing one of the specialty of the Baxter has been developed, using both arms, for a simple, repetitive action which separates a produced piece in an automatic production line, and stores each part in separated places. A user interface which allows all kinds of users to control Baxter Robot without needing system administrator knowledge.

This research, could be applied to a real processing line, where Baxter role would be defined, developing the desired operations, offering them via the user interface with further configuration.

Future Work

The main point of this document was showing the capabilities of Baxter Robot, so the user interface is not prepared to facing the issues of a production environment. Further options for configuration, number of iterations of each program, timers, preplanning of actions, adding new scripts without using the code would be nice additions, which they weren't a relevant aspect for a showcase, they become quite important for a working space.

As Baxter SDK evolves, recheck for new functionalities and improve the existent code. Maybe in future releases of the SDK, the thread issue with the ROS instances can be solved. Research with graphic tools like opencv with the Kinect options to automatically detect objects and adapt movements on the fly.

Bibliography

[1] "Baxter," Rethink Robotics, [Online]. Available: http://www.rethinkrobotics.com/baxter/. [Accessed 1 2017].

[2] A. Solanas, "Baxter," Github, [Online]. Available: https://github.com/Naxsel/Baxter. [Accessed 1 2017].

[3] "IFR," [Online]. Available: https://ifr.org/ifr-press-releases/news/world-robotics-report-2016. [Accessed 1 2017].

[4] "Spectrum," [Online]. Available: http://spectrum.ieee.org/robotics/industrial-robots/rethink-robotics-baxter-robot-factory-worker. [Accessed 1 2017].

[5] "Baxter Overview," Rehink Robotics, [Online]. Available: http://sdk.rethinkrobotics.com/wiki/Baxter_Overview.

[6] R. Robotics, "Main Page," [Online]. Available: http://sdk.rethinkrobotics.com/wiki/Main_Page. [Accessed 1 2017].

[7] "Baxter Setup," Rethink robotics, [Online]. Available: http://sdk.rethinkrobotics.com/wiki/Baxter_Setup. [Accessed 1 2017].

[8] "Workstation Setup," Rethink Robotics, [Online]. Available: http://sdk.rethinkrobotics.com/wiki/Workstation_Setup. [Accessed 1 2017].

[9] "Hello Baxter," Rethink robotics, [Online]. Available: http://sdk.rethinkrobotics.com/wiki/Hello_Baxter. [Accessed 1 2017].

[10] "What is Python?," Python Software Foundation, [Online]. Available: https://www.python.org/doc/essays/blurb/. [Accessed 1 2017].

[11] C. Liddick, "What qualities or capabilities of python makes it a good programming language for robotic engineering," [Online]. Available: https://www.quora.com/What-qualities-or-capabilities-of-python-makes-it-a-good-programming-language-for-robotic-engineering.

[12] "What is ROS," [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Accessed 1 2017].

[13] R. Robotics, "Baxter Software Developers Kit," [Online]. Available: http://sdk.rethinkrobotics.com/wiki/Baxter_Research_Robot_Software_Developers_Kit_(SDK).

[14] "manual," VirutalBox, [Online]. Available: https://www.virtualbox.org/manual/ch01.html. [Accessed 1 2017].