Expertise
and insight
for the future

Mikko Peltola

Evaluation of a Robotic RFID Tag Testing System

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor's Thesis

4 November 2020

Author	Mikko Peltola
Title	Evaluation of a Robotic RFID Tag Testing System
Number of Pages	52 pages
Date	4 November 2020
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Smart Systems
Instructors	Antti Paukkunen, Engineering Director, Voyantic Ltd Keijo Länsikunnas, Senior Lecturer

This thesis explored the challenges and requirements needed to design and implement a small-scale robotic RFID measurement system. The primary goal was to provide the case company with the foundation of knowledge and experience required to successfully pursue future undertakings with robotics. The secondary goal was to implement a flexible prototype system for various RFID testing applications and to integrate it with the case company's RFID measurement equipment.

To achieve these goals, research on robotics was conducted, which led to an iterative system design phase. This was followed by a selection process for the system's components, after which the design was implemented. The implementation phase consisted of configuring hardware and software development.

The thesis achieved its goal of building a knowledge base that the case company can leverage in future projects relating to small-scale robotics. The project also resulted in a working prototype of a programmable robotic system with an accessible GUI front end, that connects via network to the back end that is housed in a robot controller.

The thesis demonstrated that the development of small-scale robotics in-house without prior experience is feasible. However, it was noted that a baseline of experience in the field is beneficial by streamlining a project's development cycle especially in the early stages. Another insight was the difficulty of evaluating the real-world performance of components based purely on written specifications, which may lead to unforeseen consequences. Equipped with the experience gained from the thesis, the case company is better equipped to pursue future robotics projects, which will manifest in reduced development costs and more cost-effective choices in hardware.

Keywords	robot, manipulator, RFID, ROS

Tekijä Otsikko Sivumäärä Aika	Mikko Peltola Evaluation of a Robotic RFID Tag Testing System 52 sivua 4.11.2020
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	tieto- ja viestintätekniikka
Ammatillinen pääaine	älykkäät järjestelmät
Ohjaajat	tuotekehityspäällikkö Antti Paukkunen lehtori Keijo Länsikunnas

Insinöörityössä kartoitettiin pienen mittakaavan robottijärjestelmän suunnittelun ja kehityksen vaatimuksia ja haasteita sovellettuna RFID-performanssimittaukseen. Työn päätavoitteena oli kohdeyrityksen tietotaidon kehittäminen robotiikassa mahdollisia tulevia projekteja varten. Toisena tavoitteena oli kehittää joustava prototyyppijärjestelmä vaihteleviin RFID-mittausapplikaatioihin integroiden kohdeyrityksen RFID-mittalaitteistoa robottijärjestelmään.

Insinöörityö toteutettiin kirjallisuustutkimuksella robotiikan alalta, jonka pohjalta toteutettiin iteratiivinen järjestelmäsuunniteluvaihe. Suunnitelman pohjalta käynnistettiin valinta soveltuvien laitekomponenttien ja kehitystyökalujen löytämiseksi. Kehitys tapahtui fyysisen laitekonfiguroinnin ja ohjelmistotuotannon menetelmin.

Projektin päätavoitteessa onnistuttiin. Suurin hyöty kohdeyritykselle oli insinöörityössä kerätty tieto alan käytännöistä ja mahdollisten piilevien ongelmien karttamisesta. Lisäksi työn lopputuloksena oli prototyyppi ohjelmoitavasta robottijärjestelmästä. Järjestelmä koostuu graafisesta käyttöliittymästä, komentoja suorittavasta robottiohjaimesta sekä itse robotista.

Insinöörityö osoitti pienen mittakaavan robottijärjestelmän toteuttamisen olevan mahdollista yritykselle ilman aiempaa kokemusta alalta. Tästä huolimatta huomioitiin pohjustavan kokemuksen alalta tehostavan ja virtaviivaistavan kehitystyötä erityisesti projektin alkuvaiheissa. Lisäksi laitteiden ja kehitystyökalujen reaalimaailman performanssien arviointi ainoastaan teknisten tietojen pohjalta osoittautui odotettua mutkikkaammaksi. Varustettuna insinöörityöstä kerätyllä tietotaidolla kohdeyritys kykenee tulevaisuudessa toteuttamaan robotiikkaprojekteja kustannustehokkaammin leikkaamalla tuotekehityskustannuksia ja tekemällä tuottavampia valintoja materiaalihankinnoissa.

Avainsanat	Robotti, manipulaattori, RFID, ROS

Contents

List of Abbreviations

API	Application programming interface
CLI	Command-line Interface
CPU	Central processing unit.
EPC	Electronic Product Code
IDE	Integrated development environment
GPIO	General-purpose input/output
GUI	Graphical user interface
LTS	Long-term support
NFC	Near-field communication
OpenGL	Open Graphics Library
OS	A computer’s operating system.
RFID	Radio-frequency Identification
ROS	Robot Operating System
TCP/IP	Transmission Control Protocol (TCP) and the Internet Protocol (IP)
TID	Transponder ID
TTL	Transistor-transistor logic
udev	Device manager on Linux (Userspace /dev)
UEFI	The Unified Extensible Firmware Interface

UHF	Ultra-high Frequency
URDF	Unified Robot Description Format
USB	Universal serial bus
VM	Virtual Machine
WLAN	Wireless local-area network
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

1 Introduction

This thesis was commissioned by Voyantic Ltd, which will henceforth be referred to as the case company, that is a leading provider of RFID testing and measurement solutions. The case company was interested in expanding their knowledge on whether small-scale robotics could be utilized in laboratory testing relating to RFID. The focus was on planning how such a system would operate, identifying various limitations that would need to be considered, and possibly designing and implementing a concept level demo system that could be used for marketing.

The primary goal of the thesis was exploratory in nature. The intention was to build a foundation of theoretical and practical knowledge about the design of small-scale robotic systems and their suitability in RFID tag testing, as it could have the potential to reduce manual work in certain testing applications. In addition, this could reveal what is feasible for the case company to develop in-house, and at what cost, regarding material costs and development effort. The knowledge was deemed valuable as robotics was an uncharted field of technology for the case company. A secondary goal of the thesis was to attempt to implement an eye-catching, cost-effective, interactive system for marketing purposes.

This thesis will explore the basics of robotics along with key physics concepts regarding robotics and RFID technologies in section 2. Section 3 focuses on the design process and its challenges, and on crystallizing specifications for the implementation. The selected tools and components for the prototype build are described in section 4, followed by the implementation phase in section 5. The thesis' conclusions and proposals for future development are found in section 6. Finally, section 7 contains a summary of the thesis as a whole.

2 Research

The following section will introduce several key concepts central to the project concerning robotics and RFID technology. Starting from the basics, the first sections will provide an introduction to the fundamentals of robots by describing the essential building

blocks of a robotic system. This is followed by two key physics concepts relating to the field of robotics. The next section introduces common robot archetypes and their typical use cases, followed by a loose guide on how to approach the more practical considerations when designing or selecting a robot. The last section introduces the basics of UHF RFID technology.

2.1 Physical Makeup of a Robotic Manipulator

A robotic manipulator is comprised of a number of smaller parts which can be grouped into six categories: an arm, end-of-arm tooling, a drive system, a controller, a teaching method, and a robot cell. These concepts will be individually expanded upon below.

2.1.1 Links and Joints

What we think of as the robot's arm is constructed of elements called links. A link in a robotic manipulator is any unpowered rigid body. They are what give the robot its form. Any number of elements in a system can be thought of as a link if the bodies have no translational or rotational motion between them [1, p. 3.] The dimensions and the material of the links are central when considering the physical properties of the robot such as reach, weight, strength, and resistance to environmental conditions.

While links give a robotic arm its dimensions, joints enable motion and dictate how it can move. The two basic types of joint constructions are prismatic and revolute, which perform translational and rotational motion respectively. In addition to their movement types, joints may be either active or passive. Active joints are driven by an actuator to change or hold the manipulator's position, while passive joints enable movement but do not have a dedicated power source of their own. [1, pp. 3-4.] Of the two, active joints are ubiquitous in most robotic arm constructions, with the exception of delta type parallel robots.

Prismatic joints allow for the output link to move along an axis. As the movement is translational, the offset is measured by the change of distance. Two common types of prismatic joints are linear and orthogonal joints. A linear joint moves the output link in parallel to the input link, allowing for pushing or pulling translational motion. An

orthogonal joint moves the output link at a ninety-degree angle to the input link. [2, p. 261.]

Revolute joints spin the output link around an axis, and the change in position is measured by the output link's change in angle. A twisting joint turns the output link along the same axis as the input link. Rotational and revolving joints are in principle analogous to the linear and orthogonal joints respectively. A rotational joint's output link turns along the same plane as its input link, while the revolving joint's output link turns along a plane that is perpendicular to the input link's plane [2, p. 261.]

The combined factor of the links and joints ultimately determine a robot's work envelope, which is the space where it is capable of reaching and able to conduct its operations effectively. The effective work envelope may be altered by manipulating the placement of the robot and by changing its orientation to better suit the application. [3.]

2.1.2 End-of-arm-tooling

The primary function of a robotic arm is to transport a tool to a specific location and orientation for it to perform the assigned task [2, p. 264]. This tooling is referred to as end-of-arm-tooling (EOAT), also called an end-effector. The type of EOAT depends heavily on the end application, but common ones include both servo-operated and pneumatic grippers, welders, suction cups, and spray nozzles for material coating. A platform called a quick changer may be used to swap between several EOATs while the robot is in operation to have a single robot perform multiple different tasks. [2, pp. 266-268.]

The EOAT of a robot is a crucial and highly specialized component in the system. As such, it may not necessarily be from the same manufacturer as the robot's arm. Due to being so dependent on the specific use case of the customer, custom made EOAT designed for specifically that task are not out of the question.

2.1.3 Drive System

The drive system of a robot, also referred to as the power source, is the method by which the robot's actuators are powered, and how it is able to move and perform tasks. Most robots are powered by either pneumatic, hydraulic, or electric drive systems [2 p. 269].

In a pneumatic drive system, compressed air is channeled in a controlled manner to performed a specific task. As air is abundant in most terrestrial environments, it is easily pressurized and stored in pressure vessels. An advantage over its competitors is the relative simplicity of its piping, as used air merely disperses into its environment without the need for return lines and complex closed-loop systems. [2, pp. 269-270.]

The compressibility of air is, however, also what leads to its biggest flaw, which is accuracy. The challenge in precisely controlling objects launched by bursts of air leads to most pneumatic systems employing simple physical hard stoppers at the end of their intended trajectories. Pneumatic drives are typically used only in more simple and cheaper solutions that do not prioritize high accuracy, like in the EOAT for pick-and-place operations of smooth objects, such as sheets of metal or panels of glass.

Hydraulic drive systems operate by pushing either water, oil, or other incompressible fluids. It is common for hydraulic manipulators to have an excellent power-to-size ratio, being able to generate more force than their pneumatic and electric counterparts with similar sizes. Also, due to the relative incompressibility of the fluid being used, hydraulics are commonly capable of more precise movement than pneumatic systems. The relative accuracy and good power ratios lend hydraulic manipulators to perform well in heavy duty lifting and precise moving of large objects. [2, pp. 269-270.]

However, there are several drawbacks which explain why hydraulics are not as ubiquitous as electrically driven robots. First, the necessary support machinery required for the system to operate add to the complexity and size of the machine. Secondly, a high-pressure system inherits hazards relating to leaks, cavitation and catastrophic component failures. Both of these factors increase the maintenance costs of hydraulic systems.

The most common type of drive system in modern robotic manipulators is electricity [2, p. 270], which has numerous advantages over their competitors. Examples of commonly electrically driven actuators used in robots are DC and AC servomotors, and stepper motors.

Electrically driven actuators offer several advantages compared to their counterparts. The first benefit of electric motor based actuators is the amount of control that they exhibit, allowing for fluid motion, fast speeds, and notable accuracy and repeatability.

Being electrically driven, they lend themselves well to computer control, which allows for continuous status tracking and extremely fast reaction capabilities. [2, pp. 270-271.] As electrical cabling is flexible and relatively compact when compared to pressurized lines, electrically driven systems often occupy less space than alternative drive systems. They also have virtually no risks of environmental contamination, and noise pollution is quite limited when compared to pneumatic and hydraulic systems. The maintenance work with electrically driven systems is also relatively simple when compared to hydraulic and pneumatics.

The major drawback of an electrically driven manipulator is its relatively low payload capacity. The issue is compounded when increasing the number of actuators in an arm, as each additional actuator adds an extra weight further from the arm's base. This produces increasing amounts of torque that the system needs to overcome, requiring more powerful servos to compensate. However, more powerful servos often come at the cost of weight, which in turn requires even more powerful motors. This is why high power-to-weight ratio servos are ideal, but this is also reflected by their high price. While electrical drives can be applied flexibly, both pneumatic and hydraulic systems of a comparable size are able to beat electric systems in terms of raw power.

Moreover, electric drives may come with gearing and geared motors also have a list of their own inherent problems. One of these issues is called backlash, which is the decoupling or free-play between the teeth of two gears. This play between gears causes displacement and looseness in motion, especially during changes in direction. [7, p. 245.] Another common issue is the wearing out of gears, which is can be generally referred to as gear fatigue. This issue is characterized by the malformation of the gear's shape by bends of the gear or wear and pitting of the gear teeth. [4.]

2.1.4 Controller

The controller is both the brain and signal hub of a robotic system, responsible for gathering data, sending data, and decision making. They range from small single microcontroller all-in-one systems up to industrial cabinet sized controllers that house dedicated subsystems for various tasks.

Modern electrically driven robots operate as a closed loop feedback system [2, p. 270]. This means that status information from the arm's actuators, such as speed, position,

and temperature, are continuously transmitted to the controller. The data from the servos are compared to the robot's current goal, and any necessary adjustments are calculated. The calculated results are then forwarded back to the actuators, thus forming a repeating cycle of listening, decision making and responding.

Data received by the controller may arrive from within the robot cell or as external inputs. Among internal inputs, the core of the arriving messages are from the closely-monitored actuators of the arm itself, often including information about joint positions, speeds, and internal temperatures. The second major source of interest is the EOAT and any related status information. Auxiliary sources of incoming data include sensor suites, other devices, and other robots. Examples of these could be computer vision, proximity warnings, trigger signals from measurement devices, and synchronizing messages between robots. Inputs that are sourced outside of the robot's sphere of influence include an interface for manual operation and teaching, other operator instructions, and emergency shutdowns to name a few.

A controller's responsibility is to guide the robot according to predetermined instructions. These instructions range from rigid point-to-point movement instructions to relatively free-form conditional directions, depending on how elegant and intelligent the system is. [2, pp. 274-275.] An example of the latter would be a smart, vision-enabled robot that is tasked to arrange an assortment of objects by their colour. In both cases it is typical for the controller to perform on-the-fly calculations to ensure the instructions are followed as closely as possible [2, p. 275]. Even if the ideal trajectories were pre-calculated in a simulation, the system has to perform corrective tweaks due to real-world factors, such as external forces, inertia, and timing synchronization between connected systems.

Outgoing data consists of information that the controller itself has processed and signals that may be routed through the controller to other targets. These include bridging communication between other connected devices, processing and forwarding human readable system status information, and the calculated corrective measures mentioned previously, to name a few examples.

2.1.5 Programming Methods

The programming method of a robot is the approach to teaching the robot whatever task it is required to perform. The three most popular ones are teaching by demonstration, a

teaching pendant, and offline-programming. A robot can be taught with one of these methods, or it may support several alternatives.

Teaching by demonstration means that the robot is physically moved through a series of poses, saving each position along the way. One of the main advantages of teaching by demonstration is the extreme simplicity of it. Due to its intuitive nature, the method is easy to grasp by the human operator, who can program a long and complex string of instructions in a relatively short duration. However, the disadvantages of this method are quite apparent if the robot is heavy to handle, or if it is required to operate around the clock. [2, pp. 294-300; 5, pp. 261-262, 264; 6.] Also, applications which require extreme precision at the millimeter scale are ill-suited for demonstrative teaching, as humans may struggle to achieve the required accuracy.

The teaching pendant is a device that allows for remote controlled teaching of a robot via buttons or a joystick. This method is similar to teaching by demonstration in many regards, except that the training process utilizes the robot's own drive system instead of the operator's muscle power. One clear advantage of a teaching pendant over demonstrative teaching is higher accuracy with fine-tuned motion and, quite often, the option to directly feed coordinates to the device. They do however share the need to be out of commission while teaching is in progress. At the present time teaching pendants are one of the most popular methods of programming a robot, but that may change in favour of the third method in the future. [2, pp. 294-300; 6.]

Offline-programming, sometimes called software-based training or simulation programming, is a robot programming method performed via a computer model. Detailed descriptions of the robot, its surroundings and the object of interest are modelled, which allows for the operator to accurately simulate the robot's tasks. This may be done without access to actual hardware, and may be performed and tested independently without the need to halt production lines. It also provides an opportunity to test a robot's performance without risking real-world accidents. A drawback of this method the initial time investment required for the robot's operator to learn the necessary skills required for the method to be effective. [2, pp. 294-300; 6.] The better the computer model, the more accurately it will perform in the real world.

2.1.6 Robot Cell

A robot cell, also called a work cell, is an area or housing that cordons off a robot to protect the user, bystanders, and the surrounding environment from harm. It often performs an integral part in routing I/O signals to and from the controller and integrating material feeders for the robot. It should be noted that the term, depending on the source, may also describe a fully equipped robot solution including sensors, the controller, and any other peripherals. In projects using smaller and less dangerous robots, a fully realized material casing may be overlooked in favor of a simple predetermined safety zone.

2.2 Degrees of Freedom

The number of ways in which an object may move and orient itself are described by its degrees of freedom. In three-dimensional space, six degrees of freedom are necessary for full translational and rotational control of an object. Translational degrees of freedom are for moving along the x, y and z axes, while rotational degrees of freedom allow for the object to spin around the aforementioned axes. These are also often called roll, pitch and yaw respectively. Revolute and prismatic joints reduce the freedom of any latched objects by five, meaning they each have a freedom of one. Other joint types may have more than one freedom. [7, pp. 5, 63.]

The degrees of freedom of a system is described by the mobility formula. N is the number of links, including the base link connected to the ground. j is the number of joints, and f_i is the amount of freedom of each joint in the system. [16, pp. 8-10,] This is depicted in Formula 1.

$$M = 6(\mathrm{N} - 1 - \mathrm{j}) + \sum_{i=1}^{j} f_i \quad (1)$$

A simple open chain, which describes a series type robotic manipulator, gets simplified down to the sum of freedoms of the system's joints [16, pp. 8-10], as seen in Formula 2.

$$M = \sum_{i=1}^{j} \mathrm{f_i} \quad (2)$$

When considering serial type manipulators, each additional joint adds to the total freedom of the system. The number of controllable freedoms is counted separately from the total freedom of the system. When robotic manipulators are concerned, the number and the orientation of connected actuators determine if a system is holonomic, non-holonomic, or redundant. A non-holonomic system has fewer controllable degrees of freedom that it is actually capable of operating in. This is rarely the case with robotic manipulators, and more often encountered with mobile robots. A classic example of a non-holonomic system is a car. It is capable of moving on the XY plane and rotating along the z-axis, meaning it has three total degrees of freedom. However, it only has the capability to throttle and steer, which is the control of two DOF. Holonomic systems on the other hand have a one-to-one ratio of controllable degrees of freedom and total degrees of freedom. A redundant system is one where the amount of controllable degrees of freedom exceed the total degrees of freedom the system is capable of performing. Redundancy increases the dexterity of a system. [5, p. 42.] If a robotic arm has several consecutive actuators of a similar type mounted in the same orientation, it will result in a redundant system.

2.3 Kinematics

Kinematics is a subfield of mechanics that studies the motion of objects. Its focus lies particularly in the geometry of motion and its time-based properties, excluding factors such as force and mass. In this sense it can be considered to be closer to mathematics than physics. Kinematics relies heavily on matrix manipulation and vector mathematics, and it is the basis upon which motion planning in robotics is built upon. [8, pp. 33-34.] It is also integral for the calculation of critical information such as a robot's work envelope. The two subtypes of kinematics central to robotics are forward kinematics and inverse kinematics.

Forward kinematics is the method which is used to calculate the position and orientation of the EOAT relative to the base, given a set of joint and link parameters. In effect the process translates the EOAT's position from joint space to Cartesian coordinates. Forward kinematics is relatively simple, as any given configuration results in one determined end position. [5, p. 62; 8, p. 34.] Conversely to forward kinematics, in inverse

kinematics the problem lies with finding the angle or angles of the robot's joints given a position and orientation of the EOAT. This is a difficult problem that gets even more difficult with an increasing number of DOF, as the possible solution space grows. In inverse kinematics relating to robotic manipulators one must consider the possibilities of there being a single solution, multiple solutions, or perhaps no possible solutions at all that satisfy a given EOAT position [7, p. 102]. In case of obstacles existing between the requested EOAT location and the base of the robot, any available solutions can be analyzed to find ones that would avoid collision [7, pp. 103-104]. Almost all modern industrial robots use inverse kinematics algorithms for finding joint angles of a robot.

2.4 Main robotic manipulator archetypes

Robotic manipulators can be divided into six main archetypes: cylindrical, spherical, SCARA, articulated, cartesian, and parallel type robots. The number of joints and their specific configuration are variables that depend on the specific model of the robot. The archetypes described below are commonly found configurations.

Serial manipulators, also called open loop manipulators, is an umbrella term for robots that have their joints and links connecting to one another forming a single chain. [9, p. 19.] The sections of serial robots are often referred to in an anthropomorphic sense mimicking human anatomy. It is common to find references for example to the "waist", "elbow", and "wrist" of a robot. The specific joint referred to by these terms is entirely dependent on the robot's configuration. All arm-like robot configurations, such as the cylindrical, spherical, SCARA and articulated manipulators are serial robots. Cartesian robots can be classified as serial robots if they adhere to the classification rules laid out earlier in this paragraph.

Parallel manipulators are closed loop designs, as opposed to the open loop design of a single robotic manipulator. They consist of several chains of one or more joints that converge into a single platform or EOAT at their end. In many cases this construction method grants them increased stability and precision. Errors in parallel manipulators are not cumulative along a single chain, but rather evened out by several, often shorter, chains. [9, p. 19.] Their construction however makes them more complex, which leads them to be more challenging to program. Simplified serial and parallel manipulator configurations are illustrated in Figure 1. Notable types or parallel robots are the Stewart

platform, which often serves as the moving base of flight simulators, and delta robots that are often used for swift and accurate assembly work. Further exploration of parallel robots is not a part of this thesis.

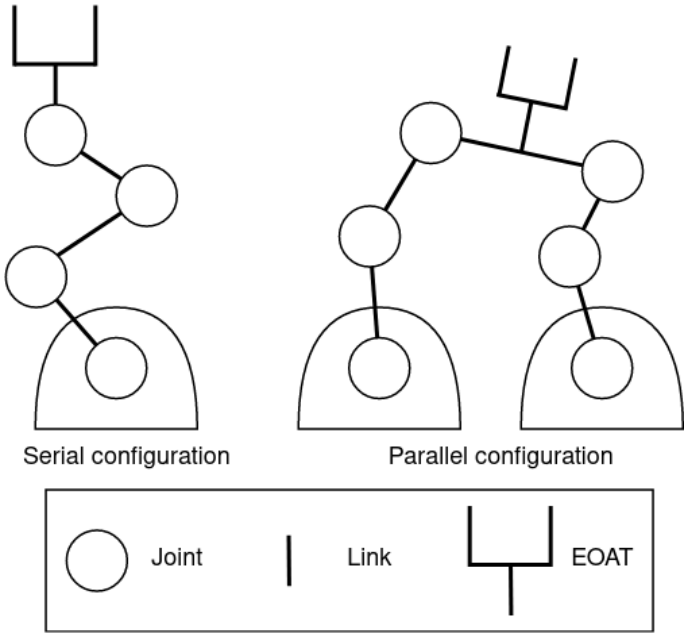

Figure 1. Serial and parallel robot configurations.

To ease the visualization and the describing of various robot types, the configuration of how either revolute and prismatic joints are connected in series will be simplified with what will be called a pattern. The notation starts from the base of the robot and ends at the tip. The letter R stands for revolute, and the letter P is used for prismatic joints. For example, the notation R-P-P would correlate with a robot with a revolute joint at its base, followed by a prismatic joint, and finally ending with a second prismatic joint.

Cylindrical manipulators are fairly simple, consisting of three joints in the R-P-P configuration, as can be seen in Figure 2. It is a three DOF holonomic robot design. The most prominent feature of this design is EOAT resting at the end of an extending horizontal linear joint. The main advantages of the system are its rigidity and a good extending horizontal reach. The system is, however, rather inflexible due to having only three degrees of freedom. It also requires a relatively large clearance compared to its footprint. As the name implies, the design's work envelope is cylindrical in shape. Though mostly phased out in favor of more agile systems, they are still used in some industries

due to their mechanically simple and rigid nature, especially if horizontal point work is required.

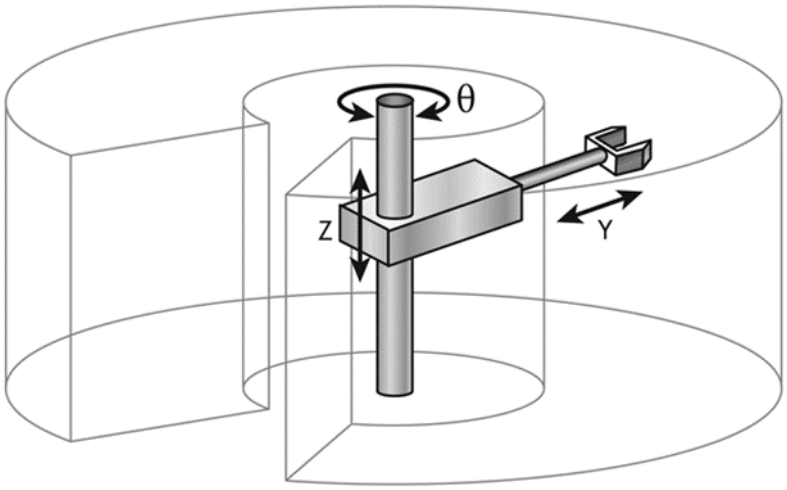

Figure 2. R-P-P configuration cylindrical robot design [17].

The spherical robot is a slight variation of the cylindrical manipulator. Spherical manipulators typically follow either the R-R-P or R-R-R pattern. Figure 3 below demonstrates an R-R-P type spherical robot. They are also called polar robots, referring to the polar coordinate system they are often programmed in. The work envelope of this type of robot is doughnut shaped with a gap directly above the unit's base. Like the cylindrical robot, they have fallen out of favor in many fields, but they can still be found in heavy-weight factory operations and are often powered by hydraulics.

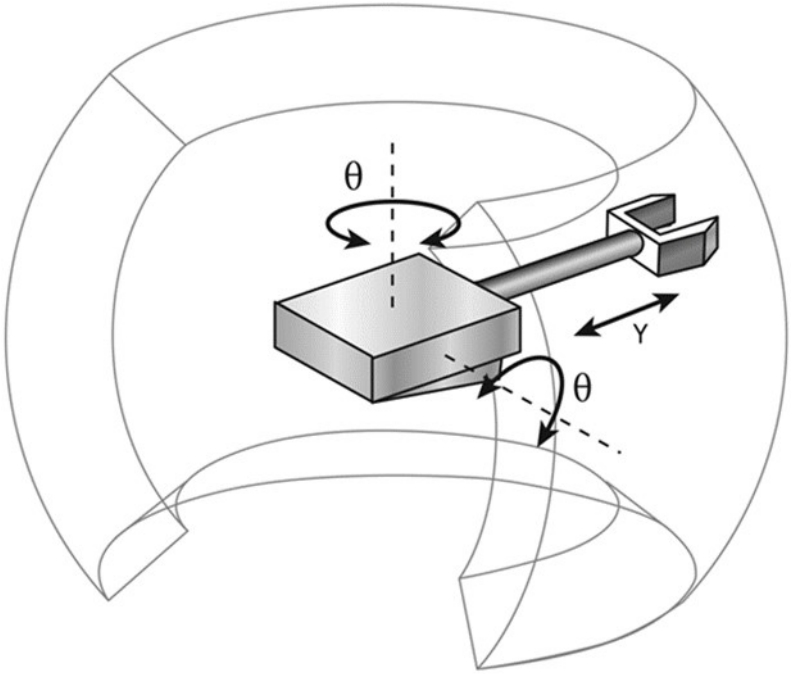

Figure 3. R-R-P configured spherical robot design [17].

SCARA (Selective Compliance Assembly Robot Arm) robots often come in patterns of R-R-P or R-R-P-R with an additional revolute joint at the tip for rotating objects. See Figure 4 for a typical SCARA configuration. Its work envelope takes the shape of a rounded half-circle in front of the pedestal. The revolute joints of a SCARA are oriented in parallel to each other, which makes it rigid along the Z-axis; It has good mobility in the XY plane, but very limited motion along the vertically. The SCARA are used in precise assembly work and fast pick-and-place operations. It is a highly specialized type of robot that excels in its field, but is not very versatile.

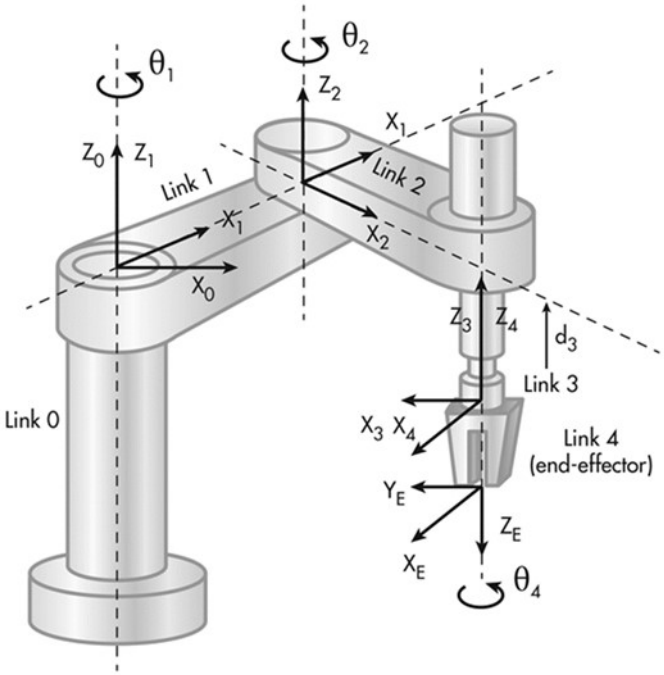

Figure 4. R-R-P-R configured SCARA robot design [17].

Articulated manipulators are solely composed of revolute joints. The typical articulated robot comes in the R-R-R-R-R-R pattern with six revolute joints, which Figure 5 illustrates. Their work envelope is sphere shaped. With the correct joint configuration and orientation an articulated robot has six degrees of freedom, making it capable of moving its EOAT to, with a few exceptions, any position and orientation within its reach. This is what makes it the most popular type of modern robotic manipulator. Their capacity for complex motion makes them a versatile choice, being capable of performing in a variety of different roles. The source of their agility is however also the cause of their weaknesses. Cumulative positional errors and relatively small payloads are problems inherited by the large number of joints required for complex motion.

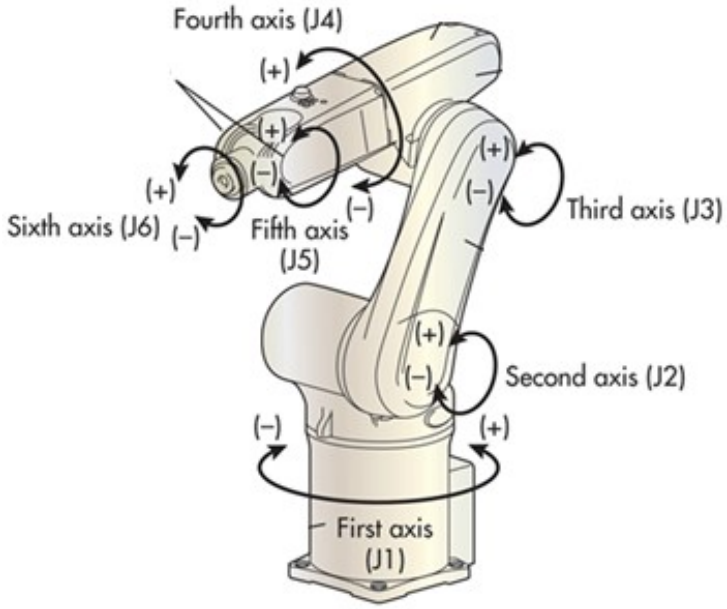

Figure 5. R-R-R-R-R-R configured articulated robot design [17].

Cartesian robots, also called gantry and XYZ robots, rely strictly on the P-P-P pattern. An example of this type of cartesian robot is shown in Figure 6. Each joint is orthogonally connected to the other, making it a 3 DOF holonomic system. As such they are capable of moving the EOAT or objects from point to point, but are unable to affect their orientation without the help of added revolute joints. Cartesian robots are known for being precise, stable, and simple. The obvious downside is their large footprint. Their work envelope exists solely within the bounds of the robot's frame. Cartesian robots are available in radically various sizes and their applications include CNC machines, 3D printers, and pick-and-place operations.

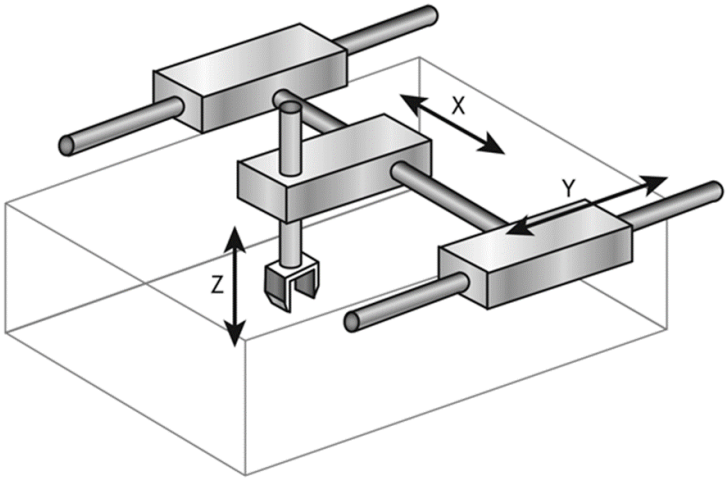

Figure 6. P-P-P configured cartesian robot design [17].

2.5 LOSTPED

LOSTPED is an acronym for load, orientation, speed, travel, precision, environment, duty-cycle. It is meant to help with the selection of an automation system. It is a useful guideline that covers the primary factors that need to be considered when buying or designing a robotic system. Pricing itself is not considered in LOSTPED, as the final price of the system is largely determined by the parameters selected with the process.

Load refers to the forces involved in the robot's normal operations, such as the weight and dimensions of the object being worked on. It is important to add the weight of the EOAT to any load calculations, as the system must be able to handle the increase in torque at the end of the kinetic chain. [10.] The maximum load of a robot is often one of the larger factors in the overall cost of the system, as heavier loads require more powerful actuators and link materials that are both light and strong.

Orientation covers both how the robot is mounted and the angles at which the manipulated object is to be maneuvered [10]. The working environment of the robot may dictate the need for a ceiling or wall mounted robot, often to save space or to move the robot out of the way of other processes.

Speed encapsulates both the velocity and the acceleration of the system [10]. The largest and most obvious factor speed affects is the total possible output of the robot; faster systems process more goods in a given amount of time.

Translation, or travel, is simply how far can the robot extend and what is the size of its work envelope [10]. For serial manipulators reach is often linked to the robot's power, as longer reaches translate to more greater forces applied to the arm. Cartesian robots are mostly not affected.

Precision can be separated into accuracy and repeatability. Accuracy refers to the ability to hit a specified location with as little error as possible, while repeatability measures how closely repeated actions are grouped together. Although a combination of both good accuracy and repeatability is ideal, the necessary precision is heavily dependent on the use case of the robot. [10.] Unnecessary amounts of precision will most likely increase the price of the system with no returns on the higher investment cost.

The environment refers to the space the robot occupies. The environment of a robot may contain hazards, such as particulates, moisture and extreme temperatures to name a few, which need to be accounted for with suitable protective casing of vulnerable points. Another consideration, which is rarely an issue for industrial robots, are other fragile or sensitive obstacles in the environment. These include humans or other dynamic entities which may interfere with a robot's work. [10.]

Duty-cycle raises the questions about when does the robot need to operate and for how long. As the durability of a system is related to how much it is used, some use cases might have stricter requirements on the type or quality of building materials of the system. [10.] If a robot is expected to perform 24 hours a day all year round, the durability of the materials and the expected maintenance times need to be accounted for.

2.6 RFID

This section will focus on the basic operating principles of passive Ultra-High Frequency Radio-Frequency Identification systems, or simply UHF RFID. Other RFID technologies, such as Low Frequency, High Frequency, and active tags will not be discussed, as the RFID measurement device used in the project operates with passive UHF tags.

RFID originated as an alternative for barcodes, but the technology has developed and refined significantly as new use cases are discovered. Its many fields of use range from Medical applications, industrial automation, payment options, the tracking of athletes in sports, access control, to retail inventory management. The two basic components of any RFID system are the interrogator and the transponder, which are more colloquially known as readers and tags, respectively. [11, p. 6.] Passive tags do not have a power source of their own and are purely reliant on the reader's power to operate [11, p. 9].

The reader is a device which can perform either read or read and write operations on RFID tags. It has an RF transmitter for sending power and data to the tag, and a receiver to read the tag's response. A reader also houses a microcontroller as the control unit, and a coupling element. Readers are also often capable of being connected with other systems, such as a PC, to receive orders and to forward tag read results. The reader operates by turning on a carrier wave that powers up the receiving tag, after which the carrier wave is modulated to encode data to the stream. In a successful transaction the tag's response is demodulated to extract the received data. [11, p. 318.]

A tag is constructed of a coupling element and a microchip [11, p. 9]. The coupling element is an antenna through which the tag receives both power and data that is supplied by the reader. This process is depicted in Figure 7. The microchip is responsible for data processing, possible cryptographic operations, and data storage. Tags adhering to the GS1 EPCglobal Gen2 air interface protocol may implement four distinct memory banks: TID, EPC, User, and Reserved memory. The tag's unique identifier, manufacturer, and model are stored in TID memory. EPC, short for electronic product code, is intended for storing the information of the object to which the tag is attached to. User memory allows for arbitrary information to be stored on the tag by the user or manufacturer. This is also the only memory bank that is not mandatory to comply with the protocol. Finally, reserved memory stores access and kill passwords if they are in use. [12.] To serve a practical purpose the tags are attached to an object that required identification with accompanying information written to the tag. To accommodate a varied list of end applications, tags come in many form factors, including glass capsules, plastic casings, smart cards, and as labels, which are tags sandwiched between two surfaces with an adhesive layer on one side.

The UHF operating frequencies are 860 MHz 960 MHz, and are often used in systems in which the distance between the reader and tag is greater than 1 meter [11, p. 45].

These kinds of tags respond to the reader by what is referred to as modulated backscatter. As the tag have no power source of their own, it uses a fraction of the incoming power for a response back to the reader. The tag's IC rectifies the power from the sinusoidal carrier wave and wakes up once it has gathered enough power to operate. In order to reply with meaningful data corresponding to the reader's request, the tag influences its reflection characteristics, which is referred to as modulated backscatter. A load resistor connected in parallel with the tag's antenna is toggled to modulate the response to match the data. The returning signal is then picked up by the reader. [11, pp. 47-48.] This is demonstrated in Figure 7.

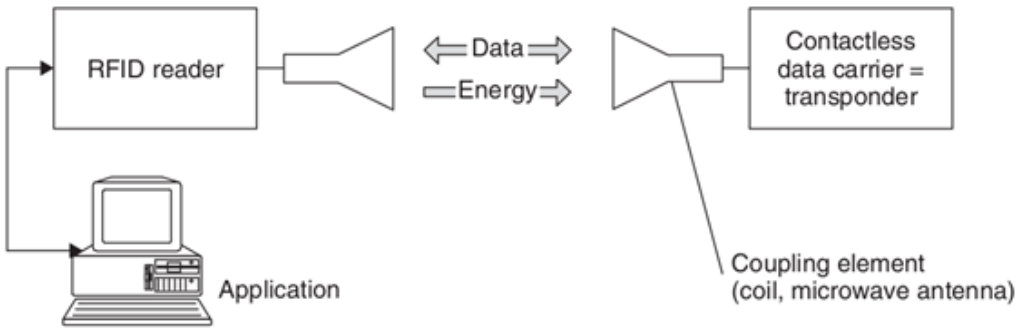

Figure 7. Interrogator and transponder interaction [11, p. 8].

The use of UHF RFID tags has many appealing qualities. From an economic point of view UHF tags are quite cheap (0.05 – 0.15 euros per tag) when ordered in large quantities. They offer a good read range of up to several meters if the signal path is not obstructed by dense materials and has the advantage of reading a large quantity of tags quickly. Unlike in other areas of RFID, most UHF tag manufacturers conform to the GS1 EPCglobal Gen2 air interface protocol, colloquially known as "Gen2" [12]. This wide adherence to a single standard within the sphere of UHF RFID makes it more easily approachable, and ensures that readers and tags may often be, to a certain degree, interchangeable.

3 System Design

The design of the project started with a vague idea of a robotic system capable of sorting between responsive and unresponsive RFID tags by using the case company's RFID measurement products. This general goal gave direction to the research that was conducted previously, and provided useful constrictions to work with. However, the

details of the automated testing setup remained open. It was conceptualized that the final design could be divided up into three top level compartments: A PC for measurement results and to control the robot, an RFID measurement system, and the robot to handle object manipulation. This general direction is represented in Figure 8. The following section will cover the design and refinement of the system setup, along with the iterative process and guidelines that were used for the selection process of the robot.

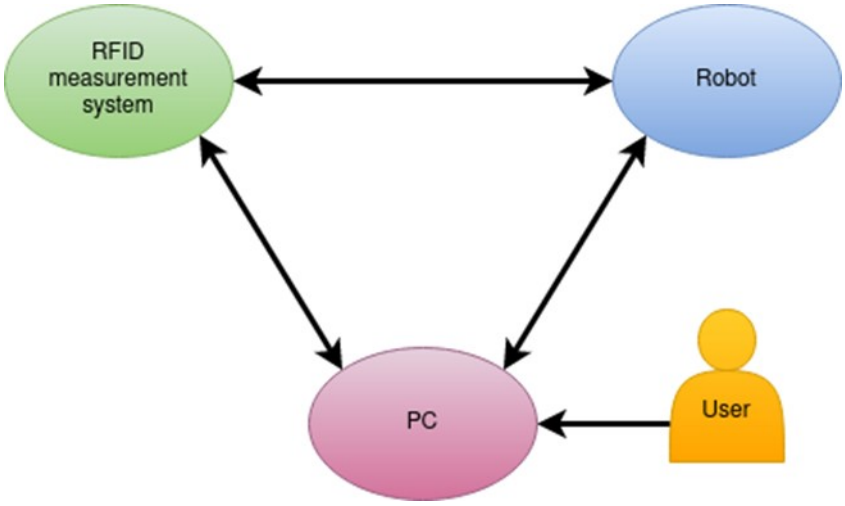

Figure 8. General concept of the system's design.

3.1 Considerations and Mapping of Requirements

The design process was launched by mapping out the general requirements and constraints of the project. The process took several key aspects into consideration, which included practical, technical, financial, and a handful of non-quantitative factors. The mapping process was largely driven forward by coming up with various scenarios while identifying elements that were unfeasible and for what reason. This process was also aided by discussing the design within the case company.

Once rough drafts of possible system designs were complete, a presentation was held for the case company in order to deliver a condensed package of the conducted research, after which a discussion was held in order to gather new ideas and to uncover possible restrictions. The outcome of the session was a combination of both practical and non-quantitative factors that should be regarded during the design process. These included considerations relating to the practical issues of transporting the robot, its

reliability and ease-of-use, and the style of the robot. Many ideas relating to the booth demo application were also introduced.

As the robot was intended to appear at trade shows to accompany the case company's own products, it was necessary to review exhibition guidelines on show pieces displayed at booths. The consensus was that if the show piece has the potential to cause harm it would require adequate safety measures. In practice this confirmed that a robot cell would be required to insulate the robot from the audience.

Considering various designs was largely a process of elimination, as different solutions and capabilities were examined to identify aspects that were, for one reason or another, unfeasible to develop within the scope of the project. That said, as the system was meant to be customizable and re-programmable, shutting down possibilities for minor reasons was avoided. Some of the most influential considerations are described below.

The lab and demo applications would differ from each other quite significantly. Even the variation within the roles would likely be significant as new environments, payloads, measurement setups, and other factors could differ case by case. This would require the system to be readily customizable both physically and in implementation.

An electrical drive system was recognized as the best option for the project. This came down to issues of portability, simplicity and safety requirements that rendered pneumatic and hydraulic drive systems unsuitable or clumsy options. Also, in the case of hydraulics, it is not even certain whether or not such small-scale systems are available.

An alternative payload scheme was thought of: instead of picking up tags and transporting them to an antenna, it was also considered to attach the antenna itself to the robot. Moving around the antenna opened up possibilities with measuring sheets or trays of tags quite effectively. Even in the case of densely packed tags, the antenna could be equipped with a shielding plate to block neighboring tags from responding. This approach would work best with a cartesian type robot. Arm-like robots may have struggled with the concept due to the extra weight at the end of a single chain. Possible risks included servo burnout, swaying, and a pendulum effect with inadequate PID optimization. The tangling of the antenna's stiff coaxial cable should also be considered, as there would be a risk of a introducing a sharp bend in the cable if it were routed along the arm. A solution would be to suspend the coaxial from above, but this would diminish

the advantage of a small footprint. A cartesian robot would have none of the aforementioned problems.

The payload's size, weight, and texture affect the desired EOAT type. Even within the realm of RFID tags there exists a number of form factors for tags that possess these differing qualities. Smooth surfaced objects would lend themselves well for a suction cup EOAT. However, the addition of pneumatics and a compressor increase the complexity of the system. A servo operated gripper would be quite simple, as the drive system would be electric. Customizable fingers would allow for fine tuning of the gripper according to the payload. A gripper does however require more accuracy from the robot in order to be effective. It was also considered that whatever the tag type may be, they could be sandwiched between two pieces of foam to transform all measured items into a standardly sized object. The foam would make the tags identical in size, easier to grip, and more visible to an audience. The foam solution would work far more effectively in a demo setup than when actually measuring dozens or hundreds of tags. As an RF-friendly material, the foam casing was also expected to cause minimal effect to the tuning of the tag, thus no interfering the RFID performance test.

Choosing the packaging or storage from which untested RFID tags were to be drawn was surprisingly influential. The difference between two types of stock could possibly mean adding whole new levels of complexity to the project. One such example would be a box full of tags. If not neatly sorted in a predetermined and specific way, the robot would need to be capable of distinguishing individual tags within a pile, calculating the best way of retrieving said tag, and the capability to physically perform this action. This kind of approach would lean more into the realm of machine vision and machine learning, which is outside of the project's focus area and likely be quite time intensive. The idea of rolls of tags was discarded, since they lend themselves more readily to a reel-type system, which the case company already provides. Stacks would require less real estate in the work envelope of a robot, but would also need to sense the proximity of the ever-diminishing stack via sensor. However, the tray implementation lends itself better to a demo scenario, as a tray would be more easily replenished by the robot during the demo than a stack. As the system is intended to be customizable and programmable, this point may be moot. The decision was to ensure that the work envelope is large enough to accommodate a tray solution if necessary.

3.2 List of Requirements

The compiled list of requirements could be split into three rough categories: LOSTPED requirements, practical requirements, and soft requirements. These will be listed below.

Requirements identified via LOSTPED are mostly physical properties directly relating to the robot and its environment. Reciting through the acronym aids in capturing essential details. Even if the approximated values turn out to be slightly off, they would provide an adequate estimate to work with. The loads the robot is expected to be capable of carrying ranges from a few grams to 300 grams. The payloads would probably vary in size, shape, and material. The robot would most likely be oriented horizontally mounted on a portable pedestal. The speed of the robot is not an important consideration, as the system will not have any tight deadlines to adhere to and the output speed is not critical. The travel, or reach, of the robot is a crucial factor, as the work envelope needs to be large enough to accommodate a stock of RFID tags and the antenna of the measurement system. A 50 cm reach was estimated to suffice. The robot would be expected to perform within +/- 0.5 cm precision-wise. The robot's environment would vary, but most likely they would be free of any meaningful moisture and particle contamination. However, it would more than likely exist with spaces that contained people and inanimate obstacles. The duty-cycle is expected to be sporadic with no continuous periods of operation.

The practical elements dictated that the system must be easily transportable both within the offices of the case company and to trade shows internationally. This meant that the whole system had to be small, light, and easily packed within one or two hard cases. The system also needs to be customizable, as the use case may vary between scenarios. As the project was an introduction to the field of robotics, it would have been wasteful to budget for anything more than an economical small-scale robot. Some of the requirements had a tendency to relate more directly to the system's software implementation rather than its physical properties. These included reliability, ease-of-use, a friendly learning curve, and safety.

The last requirements were soft or non-quantifiable. As the system was to appear at trade shows, it had to have eye-catching qualities to lure in an audience to the case company's booth. This point is purely subjective, but through discussions at the case company it became clear that an arm-like robotic manipulator was considered to have more dazzle than a Cartesian style robot. The other major requirement was the

opportunity to fulfil the goal of learning about the benefits and challenges of developing automation solutions with small-scale robotics. Fully equipped out-of-the box solutions would have been contrary to this goal.

With these constraints in mind it was time to move forward with the general system design and probable applications.

3.3 High-level System Design

This high-level system design describes the physical components and how they connect to each other on a macro level. Software architecture and the specifics of the robot will be elaborated on in the coming sections. It is worth mentioning that the system would allow for a number of different applications, including a variety of demo and automation setups, including ones without RFID testing requirements.

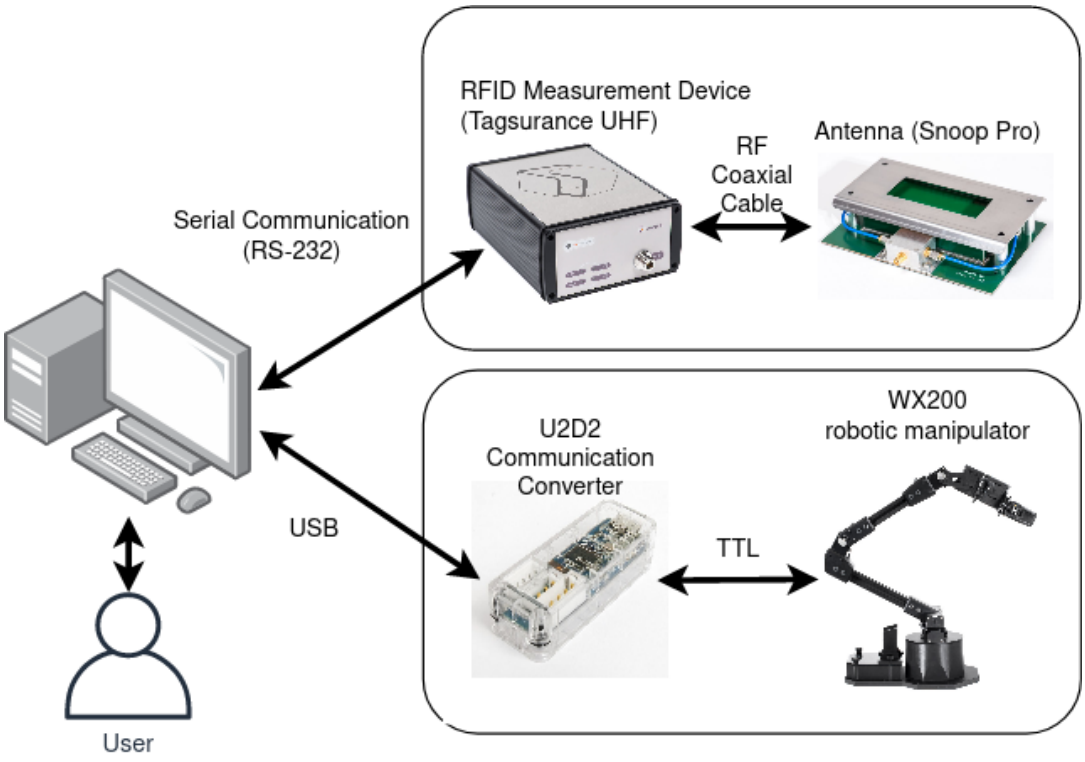

Figure 9. System level design.

The end user's main interface to the system is a PC, which resides outside of the work envelope. Through it the user can configure test cases for the measurement device, see previous measurement results, and to interface with the robot. Whether or not the PC

acts in the role of the robot controller or merely connects to a dedicated one is dependent on the implementation. In practice a PC could perform a double role of providing a user interface and acting as the robot controller. The measurement department has two components: the measurement device and an antenna, both of which are manufactured by the case company. The specific measurement device may be swapped in other configurations, but in this setup a Tagsurance UHF was chosen. It connects to the PC via RS-232 and it is not needed inside the work envelope of the robot. A Snoop Pro coupling element is for UHF testing. Aside from the robot hardware and test tags, it is the only component placed within the work envelope. The robot hardware includes the robot itself and a communication converter. One side of the communication converter interfaces with the robot controller via USB, and the other end is connected to the base of the robot via TTL cable. The final element of the system is the stock of UHF RFID tags for testing. This setup is featured in Figure 9.

3.4 Possible Demo Setup

The vision of a possible booth setup was an interactive demo in which the audience could, in a spectacular fashion, interact with the case company's RFID measurement suite. The mode of interaction would be by feeding a foam-covered RFID tag into a slot in the robot cell. The foam would act as a standard size wrapper which the robot would be programmed to grab in the demo, allowing for a wide range of varying sized and shaped RFID tags to be picked up without prior knowledge of their physical measurements. In practice, the wrapper could be a simple piece of foam with a slit on one side, allowing for any tag to be quickly inserted without the need for a permanent enclosure. An RF friendly material would not affect the tag's performance in during a test measurement.

The robot would be covered in a circular robot cell made out of a transparent plastic, which would have a slot on one side for receiving and dispensing tags. The robot would perform a normal programmed demo routine of tag tray testing until it would get a signal from a sensor that a user tag has entered the cell via the slot. Once the signal activates, the robot would interrupt its current routine after finishing with the current tag, fetch the user's tag and perform a measurement on it. The results of the measurement would be displayed on a connected computer running the case company's measurement software. This could allow for the user and the booth staff to discuss the measurement results,

along with the measurement software and hardware on display. The setup could also facilitate a more thorough examination of the user's tag by having the robot rotate the tag or hold it at various ranges from the antenna. When the specified tag measurements are complete, the robot would pick up the tag and return it to the user by dropping it into the slot. Figure 10 illustrates the possible demo setup.

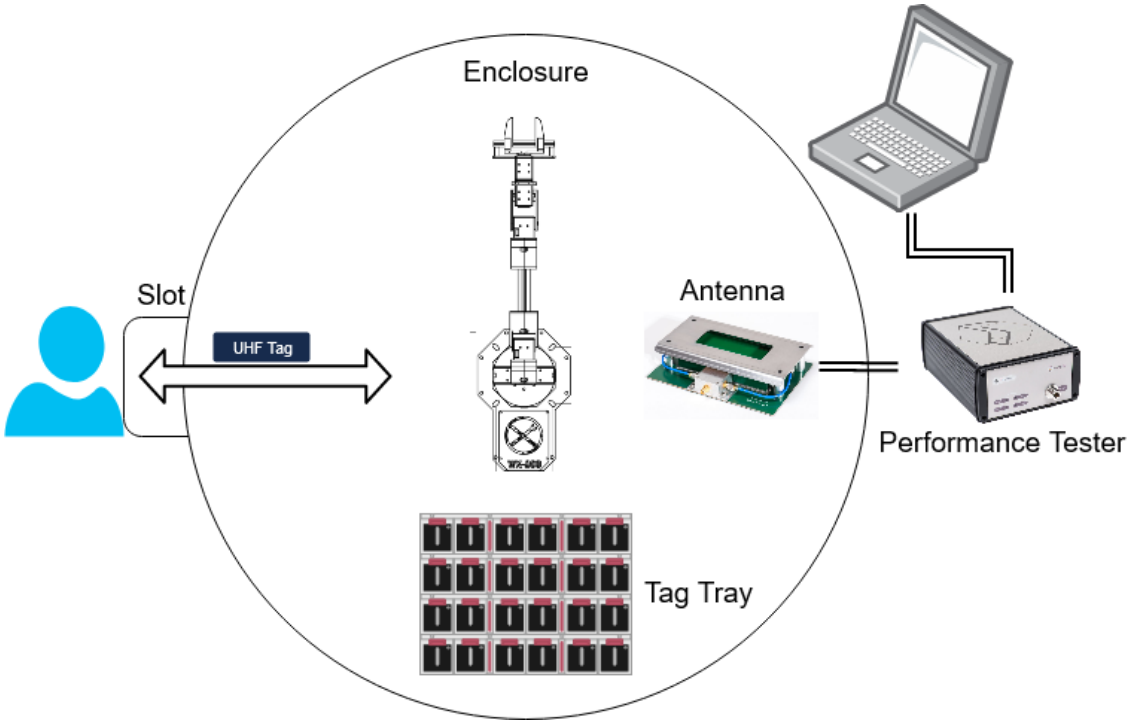

Figure 10. Possible demo setup.

4 Hardware and Software tools

This chapter will focus on introducing the hardware and software tools used in the project, and will explain the reasoning behind each one. Among the listed tools are some items that were discarded during production for one reason or another. These will be explained on a per item basis along the way.

The tools used in the project can be roughly divided and abstracted into several functional sections. The base contains any robot related hardware that connect to the computing platform of choice via the U2D2 communication converter. The OS provides an easier development environment, and acts as a platform upon which application specific tools are installed on. The middleware layer, namely ROS, offers a large

collection of build tools and programming infrastructure for developing core logic for the robot. The final layer consists of libraries and other programs that are built on top of ROS, which add specialized functionalities for interacting with WX200 or providing other movement related tools. The support layer consisted of the R+ Manager 2.0, which allowed for firmware and control table access to perform actuator tests and diagnostics. Figure 11 showcases how the various components were layered.

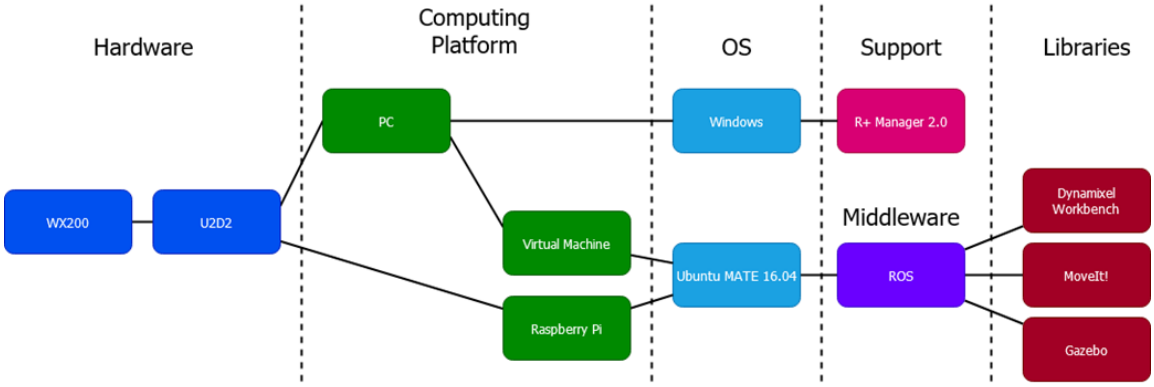

Figure 11. Hierarchical layout of the used hardware and software components and tools.

4.1 Robot Hardware

The robotic arm and any external communications peripherals linking directly to the robot are categorized as robot hardware. The following sections will introduce the Trossen Robotics WX200 robotic manipulator and the Robotis U2D2 communication converter.

4.1.1 WX200 Robotic Manipulator

The Trossen Robotics WX200 robotic manipulator, hereafter referred to as WX200, was a fitting candidate for the project due to several reasons. On paper it fulfilled many of the established criteria of the system design. Price-wise the WX200 falls into the high end of a range that could be described as hobbyist level manipulators. The next price category for relatively light and agile manipulators approaches what could be called research grade robots, that are priced roughly in the range of ten-thousand euros and up. For its price, the WX200 offers a reasonable package in size, DOF, accuracy, and modularity. The WX200 is depicted in Figure 12.

Figure 12. The WX200 robotic manipulator [13].

The WX200 is a middle-of-the-road solution between a full-on do-it-yourself setup and an off-the-shelf ready-made package. A pre-made robot system with a full application suite would defeat the purpose of the project; leaving out a large portion of the actual development process, the valued research and exploration aspect of the project would have been lost. On the opposite end of the spectrum, a building the robot arm from scratch was deemed unnecessary, as the project is intended to focus more on the software element than the physical assembly of the robot. In these aspects the WX200 allows for enough freedom to develop a unique solution.

The WX200 is an electrically driven articulated robotic manipulator. The arm itself has six Dynamixel servomotors, two of which are placed in parallel at the shoulder to provide more torque near the base. The two servos of the shoulder can be imagined as a single double powered servo. This actuator configuration sets the WX200 up as a five DOF redundant system. The first five servos are of the model XM430-W350-T, and the wrist joint has a slightly weaker XL430-W250-T servo. The EOAT is a gripper type design that uses the same servo model as the wrist. It has two plastic gripper fingers sliding on a small aluminum rail. The fingers are mounted on sleds that slide along the rail, which are replaceable with fingers of various form factors. The frame is made of lightweight aluminum beams, while the base is a 3D-printed plastic housing. Other details include the reach, accuracy and payload. The work envelope is the shape of a half-sphere with a radius of 55 cm. The working payload is specified as 200 grams and “is measured by

the arm's ability to repeatedly lift an object at roughly half extension without failure" according to the manufacturer [13]. The accuracy is rated at 1 mm.

The key differences between the two servo types are their power, heat dissipation capabilities, and a small variance in their controllable parameters. The XM430-W350-T actuators had two main features which gave it a distinct advantage in dissipating heat while driving the servos. The first feature is the brushless design of the motor, which generates less heat than a brushed motor. The second feature is the aluminum casing, which dissipates heat more effectively than the plastic housing of the XL430-W250-T. Control table of the weaker servo lacks features relating to setting a specific amount of torque when pushing against an object.

The Dynamixel servos are daisy chained together; instead of connecting separate wires to each servo from the base, each Dynamixel feeds a wire to the next servo in the system forming a chain. This reduces overall wiring and achieves a more clutter-free environment for the robot to operate in. Command packets are sent with an envelope with an identifier. If the receiving servo is not the intended recipient, it will forward the message along the chain to the next Dynamixel servo in the chain. Once the intended target of the message receives the packet, it will send an acknowledgement back along the chain to the message's source. If no response from the target is received within a timeout period, the message is considered lost.

The modular aspect of the WX200 kit is due it being constructed of seven Dynamixel servos. If the WX200 were to be disassembled, the actuators could be relinked in a different pattern and orientation, forming a robot of a different configuration. One possible example would be a SCARA type for fast and versatile pick-and-place operations.

4.1.2 Dynamixel U2D2

The Dynamixel U2D2 is a small communication converter that translates USB communication from a PC to one of three available communication types. It also acts as a barrier that protects the PC's USB ports from being damaged by power surges. As the device itself only carries data between a PC and Dynamixel servos, the U2D2 connects to an external 12 Volt power supply to provide power for itself and any connected servos.

The PC's commands are sent to the U2D2, which converts it to TTL communication, which both of the used Dynamixel models understand. The U2D2 is only responsible for transmitting the converted commands to the first Dynamixel of the chain, after which it listens for responses. These responses are then converted and forwarded to the PC via USB.

As a convenient bonus, the small form factor of the U2D2 allows for it to be stowed in the WX200's control box, which keeps the converter from getting into harm's way. Loose wires are at risk of getting caught when the robot is in motion, therefore it is necessary to keep clutter to a minimum.

4.2 Computing Platforms

Even though computing platforms are listed separately from both hardware and software, the section contains one of each. The Raspberry Pi and the VMware Workstation are categorized separately, as they fulfil a distinct role of being the base on which most other software in the design runs on.

4.2.1 Raspberry Pi 3 Model B

There are several reasons why the Raspberry Pi 3 Model B was chosen to be the robot controller of the project. Being a single-board computer, it houses many useful features directly on a single card in a small form-factor. Powered by a powerful ARM processor, it is capable of running a full-blown operating system. Another major upside of the system is the abundance of communication peripherals. Supporting USB, Ethernet, WLAN, GPIO, and Bluetooth, the Raspberry Pi 3 Model B has all the communication requirements of the project covered. The low price-point also makes it a compelling choice.

The intended role of the Raspberry Pi is to act as a versatile and highly portable robot controller. The small form-factor would enable the Raspberry Pi to be embedded directly with the robot's frame, thus reducing the overall weight and bulk of the system. Due to its versatile communication peripherals, it would be convenient to access the device remotely via SSH to perform configurations while operating the robot.

4.2.2 VMware Workstation Player 15

The VMware Workstation Player 15 is a virtualization platform that enables running a secondary virtualized operating system on top of the natively running operating system of a computer. The machine responsible for running the virtual machine is called a host, while the virtualized system is called a guest.

A virtual machine has several features that makes it suitable for prototyping and development. The major advantage of a virtual machine for this project is that the development and deployment platforms are nearly identical. This means that replicating a functioning system from the virtualized OS to the target system can be relatively straight-forward. Second, the portability of a virtual machine makes it a flexible solution when working with more than one computer. Given sufficient storage, a virtual machine can easily be transferred from one host to another. The final advantage of a virtualized OS during development is the ability to revert the whole system to a snapshot when unexpected and time-consuming problems arise. By creating system snapshots between successful installation and configuration cycles, the risk of wasted development time is mitigated.

Another candidate for the virtual machine software was Oracle's Virtual Box. The VMware Workstation Player 15 and Oracle's Virtual Box offer a fairly similar list of features, but the two key points that tipped towards VMware were the author's familiarity with VMware products and its superior compatibility with OpenGL out of the box. As the project was going to integrate a large number of different technologies, factors that reduced possible incompatibility issues were prioritized.

4.3 Software Platforms and Libraries

This section will introduce both the software platforms on which other programs will run on and libraries that were integral for any new software being created.

4.3.1 Ubuntu MATE 16.04

Ubuntu MATE 16.04 is an Ubuntu distribution at heart, but sports the MATE desktop environment. The desktop environment was inconsequential for the project, but this specific OS was chosen for two main reasons. The foremost reason is that it was

available for the ARMv7 32-bit processor architecture, which is the architecture of the Raspberry Pi 3's CPU. As it was originally an integral part of the project, all design choices had to accommodate the Raspberry Pi. Additionally, in order to ensure better stability and compatibility, the slightly older version 16.04 was selected. As it is an LTS release, it is guaranteed to be supported and maintained up until 21.4.2021. Finally, being able to build identical systems for both the development and the deployment platforms reduces redundant development time, as the back-end of the project could be replicated on the deployment platform with minor changes.

4.3.2 Robot Operating System

The Robot Operating System, colloquially known as ROS, is best described as an ecosystem of supporting frameworks, tools, and libraries to manage software development in robotics. These range from core infrastructure to support tools that help visualize the software architecture of a robot project. The name is somewhat of a misnomer, as it is not an operating system, but in essence ROS is middleware that facilitates the development of the core aspects relating to programming and controlling a robot.

The ROS environment has a number of features on offer that offload base level work from the robot application developer, enabling a faster production cycle. It has been described as follows: "ROS is to robotics as Ruby on Rails or Node is to web development" [14]. By laying the groundwork, it provides an accessible and ready-made infrastructure for robotics development, making it easy for the developer to create Nodes for communication between networked devices such as sensors, actuators and other processing units. Bundled in are CLI tools for project deployment, monitoring and debugging. ROS also provides robot simulation tools in the form of Gazebo.

ROS also has strong non-quantifiable assets. One of them is the ROS community, which is comprised of both companies and individuals. The community plays a large role in the learning process, making it more accessible for people without prior experience in the field of robotics. Another bridging aspect of ROS is its language independence. At the time of writing C++, Python, Lisp, Java, and Lua were all viable development languages that can be mixed and matched effortlessly within a single ROS project.

The basic working principle of ROS is quite simple. ROS can be described as a network of distributed processes, that enables individual executables to loosely couple at runtime, either via message channels or direct messaging. It is designed around a publisher/subscriber model, that is in many ways similar to MQTT, RabbitMQ, and other message broker systems.

At the very core of ROS is the ROS Master, which is a central hub that brokers the communication of Nodes and stores necessary information for the system to run. The Master has three main functions. First, it tracks available message channels called Topics, and two kinds of direct Node-to-Node messaging lines called Actions and Services. The second responsibility of the Master is the storing and brokering of address information of any registered nodes in the system. Third, it houses a set of global parameters usable by any registered Node.

A ROS Node is simply an individual executable running as a stand-alone process. A Node is the main logic processing element in the ROS ecosystem. Their scale can vary greatly from tiny snippets of code that convert and forward messages to massive decision-making units that reign over the movement of a robot. To be useful as a functioning member of the ROS system, a Node must be registered to the ROS Master. On registration, the Node shares its address information with the Master, along with its publish and subscribe requests. When a Node has been successfully registered, it is threaded to run in parallel with other active Nodes.

Topics, Actions, and Services are the three primary ways of inter-Node communication in ROS. Topics are message channels to which any number of Nodes may publish and subscribe. Publishing on a topic makes the data accessible any subscribed Node on the same topic. This method of communication is most often used for a continuous flow of data. For example, a sensor could publish its readings at a certain frequency, while a logic processing Node would subscribe to the topic in order to make decisions based on the readings. Actions and services are similar at their core, but with one key difference: Services are blocking calls that will halt the calling Node’s execution until a response from the serving Node, while Actions are non-blocking. This distinction leads them to be used in different circumstances. As Services are blocking, they are typically only used in situations with fast responses, or if execution should be deliberately halted for a period. Actions are often used to track longer lasting operations in the real world, such as the movement of a robot.

The tenth ROS release version, dubbed Kinetic Kame, was selected for the project. Although a newer release called ROS Melodic Morenia was already available, it could not compete with Kinetic Kame in terms of supported features and libraries. One of the critical issues was that not all of the Dynamixel ROS libraries had been ported over to work with the Melodic Morenia release. Also, as Kinetic Kame was tailored to be used with the Ubuntu 16.04 release, it was reasonable to presume better overall system stability with this ROS version.

ROS also influenced the choice to use a virtual machine with a Linux distribution instead of developing exclusively on Windows 10. Although a cooperative effort between Microsoft and the ROS Industrial Consortium is underway to convert ROS to be fully compatible with Windows 10, at the time of this thesis only a portion of the ROS ecosystem had been converted and tested for stability. In an effort to minimize risks and stability issues, Windows was discarded as an OS candidate for the project.

4.3.3 MoveIt!

MoveIt! is the most widely used platform for motion planning, EOAT manipulation, inverse kinematics, and collision checking in robotics that runs on top of ROS. It is widely used in conjunction with the ROS Visualizer and URDF to calculate and visualize trajectories for robots.

The MoveIt! platform was chosen as a primary component of a software and simulation driven solution for robot teaching. Supporting both programmatic and GUI based drag-and-drop motion planning, it was a promising candidate to be used as the main method of teaching the robot.

4.3.4 Gazebo

Gazebo is a robot simulation suite that offers tools for complex physics simulations, a graphics engine, and virtual sensors to use in a simulation. Gazebo supports both programmatic and GUI control. It is used to perform detailed experiments with various robot designs in simulated environments, which is highly useful for prototyping. Being able to simulate a set of trajectories in a crowded environment, potential real-world risks and accidents may be side-stepped entirely. Gazebo was chosen specifically for rapid

and safe virtual prototyping during development, as well as a tool for the end-user to be able to run a simulation of their robots pathing.

4.3.5 Dynamixel Workbench

The Dynamixel Workbench is a library maintained by Robotis that facilitates the easy use of Dynamixel servos with ROS. By providing ROS compatible wrappers for lower-level functions to modify and control the Dynamixel servos, the Workbench is a versatile development tool. In addition to making abstracting minutiae and making easier application development easier, the Dynamixel Workbench seemed like a promising start-off point for the project.

4.4 Other

The following software components are pieces that did not clearly fit in the other categories, and were run in Windows 10 on the host machine.

4.4.1 R+ Manager 2.0

The R+ Manager 2.0 is a free utility program from Robotis for inspection and manipulation of the RAM and ROM chips of a Dynamixel servo. Its features include reflashing of the servo's firmware, manipulation of the control tables, and fine-tuned manual control of a Dynamixel servo's properties in real-time.

The use of the R+ Manager 2.0 was mandatory, as it is the only way to access Dynamixel firmware updates, or to perform a recovery reflash. The other main feature of the R+ Manager is that it enables manual configuring of a Dynamixel's control table, which allowed for defining safety-critical parameters to the servos before attempting programmatic control of the robot. The simple user interface allows for modification of parameters, such as thermal shutdown limits, acceleration profiles, legal joint angles, and PID fine-tuning in an easy manner.

4.4.2 LabVIEW

Laboratory Virtual Instrument Engineering Workbench, or commonly simply LabVIEW, is a proprietary programming environment from National Instruments. Although the underlying graphical programming language itself is called G, it is common to simply refer to it being LabVIEW. LabVIEW comes equipped with an extensive toolchain that enables application development from start to finish. These tools include the programming environment itself, in-built debugging features, build tools, and tools for setting up run-time configurations. A standout feature of LabVIEW is the fully integrated GUI tools that the environment is built around. Due to the ease of GUI creation and the fact that the case company utilizes LabVIEW extensively, it was a natural choice for the development of the frontend component of the project.

5 Implementation

As this thesis was a research and development project in an unfamiliar field, it was exploratory, iterative and agile in nature. Not all decisions proved to be good or essential for the project. This led to one major design shift influenced by technical incompatibilities and a new insight on what would be the best way for the operator to teach a robot of this scale. This section will describe hardware testing, system configuration, and the two main design directions that took place during the project.

5.1 Assembly and Initial Setup

The manipulator arrived in two weeks from order. In the interim, other setup and preparation was done, along with some studying of the Dynamixel Workbench and Dynamixel SDK libraries to map the default capabilities of tools.

The robot arrived pre-assembled, but because of its relatively light weight and small base, it needed a sufficiently rigid external baseplate to keep the manipulator upright and anchored. This was achieved by screwing the base of the WX200 to a circular plastic plate with a radius of 30 cm and approximately 1 cm in thickness. The result was a stable and portable base plate, to which a transparent robot cell could be attached to if necessary.

As a minor problem with the components, the shipped female-to-female cable that was to fit between the power IC and the U2D2 proved to be incompatible with the U2D2's connector. The pin width matched the U2D2's terminal, but the connector was too bulky to fit. This was corrected by shaving the connector down to the correct size with a scalpel.

Before continuing on to anything else, the manipulator had to be tested in various ways to determine its capabilities and limits. The Robotis R+ Manager proved excellent for this use case.

First up was the testing of the operational range of each servo. Although the servos are able to rotate continuously for more than 360 degrees, the daisy chained configuration limits the total maximum rotation of each servo due to the wires twisting to an increasingly tight knot. The range of movement of servos ID 2 through ID 6 were limited by the robot's frame itself, as the robot would collide into itself if not properly managed. Servo ID 7, the gripper, was limited by the width of its rail. While servo ID 1, the waist, was unhindered by the frame or other obstacles, the cabling between servo ID 1 and ID 2 proved to be the limiting factor.

By doing unpowered rotation tests with servo ID 1, an angle was found that granted roughly 380 degrees of rotation in either direction. This meant that it was important to keep track of the rotations, lest the robot would damage itself or the connector cabling. During development, tracking greater than 360 degree rotations was achieved by a simple solution of attaching one end of a cord to the rotating plate and the other end to the robot's base plate. The wound cord thus acted as an easy visual aid to determine whether or not the robot had over rotated. Setting the optimal initial starting rotation remained necessary throughout the project, as the servos are ignorant to possible over-rotations at start up. Once powered, tracking of cycles beyond 360 degrees was achievable.

After the operational range of each servo had been mapped, a so-called safe zone had to be programmed into each servo's control table to limit its movement. To ensure no self-collision was possible with a single servo's rotation, a safety margin of five degrees was added to each angle of collision. After setting safe angles for the servos to rotate, angular speed limitations were set as an additional safety measure to both secure the user from accidental high-speed collisions and to protect the servo's gears from extreme

deceleration. The conducted movement tests also proved the selected base plate to be adequate to keep the system stable.

Precautions were also taken against overheating. The manuals of the XM430-W350-T and the XL430-W250-T described their maximum operating temperatures as 80 °C and 72 °C, respectively. Leaning on the safer side, hard shutdown limits 10 °C lower than the listed maximum operating temperatures were programmed into the control tables of the servos. After implementing safe shutdown temperatures, simple use case tests were conducted to observe heat build-up within the servos using the R+ Manager. In an attempt to emulate real operational conditions, the manipulator was fully extended horizontally with a 20 gram payload held by the gripper. Contrary to expectations, the only servo that reached the safety margin was the gripper itself. All others, including the wrist servo of the same model, exhibited some rise in temperature before settling at an acceptable level. It should be mentioned that the thermal tests were brief, as long isometric holds at maximum extension are not part of the expected normal operation.

After the unexpected results of the heat tests, the gripping mechanism was analyzed more closely. It appears that a large amount of energy was being wasted due to friction between the gripper's fingers and the rail on which they were sliding. The problem was exasperated when holding objects with longer gripper fingers, as the extension applied more torque against the rails. Additionally, the gripper had problems overcoming this friction when opening up. The problem required reconfiguring and filing down some of the gripper's components to have more leeway and for the components to be symmetrical. The problem was also somewhat alleviated by applying an ABS compatible lubricant to the rail.

It was later learned that the gripper's problems were not merely due to manufacturing flaws in the components, as the gripper design as a whole turned out as suboptimal. After contacting the Trossen Robotics about the issue, an updated gripper rail design was shipped to reconcile the problem. However, the effect of the new design was marginal.

5.2 Building and Configuring the Setup

Having laid the groundwork, the next tasks were to configure the virtual machine and install an operating system. To setup and use a 64-bit guest OS on the VMware virtualization software, it was necessary to enable hardware virtualization via UEFI. The virtual machine was configured to use a large portion of the computer's resources, as most of the development time would be spent within the Ubuntu MATE guest OS.

Once the guest operating system had been installed, updated and configured, work started on setting up the ROS environment. After configuring the OS to accept packages from the ROS repositories, the ROS package was installed via the ubiquitous Advanced Package Manager. The full ROS installation contains a large portion of necessary packages upon which the project was to be built. On top of the base ROS architecture, it includes helper libraries for future packages and tools for visualizing node connections, 2D and 3D simulation, and robot navigation and perception. After setting up ROS, the next important task was to gaining familiarity with the system. The ROS website is an excellent resource that provides ample teaching materials from the basics to more advanced topics. This served as an introduction to Nodes, the Publish/Subscribe messaging structure, services, actions, and package building using the ROS catkin_make build tool.

Next up was git-cloning the necessary Dynamixel packages from the Robotis repository, with the central package being the Dynamixel Workbench. Once cloning of the Dynamixel ROS packages was complete, it was time to perform a check if the Dynamixel Workbench provided CLI tools were able to locate any connected Dynamixels. The initial tests proved unsuccessful, as no communication between the servos and the guest OS was achieved.

A process of elimination was started to identify the problem. Faulty wiring was ruled out after control tests were performed with R+ Manager. Also, as R+ Manager uses the same code when searching for any connected Dynamixel servos as the Dynamixel Workbench, it suggested that the problem was within the VM. Further successful tests with a USB flash drive suggested that the USB ports were also properly forwarded from the Host to the Guest OS. A solution to the connectivity problem was found in the Dynamixel SDK's source files, specifically in the file port_handler_linux.cpp, which provided comments

embedded in the code that suggested creating new persistent udev rules in the OS that modified the default 16 millisecond USB latency.

The changes to the USB latency timer settings in both the Dynamixel source files and the OS udev rules solved the communication problem between the servos and the controlling computer. This allowed for the testing of the Dynamixel Workshop CLI tools. A new attempt at scanning for connected Dynamixels was successful, with all seven servos being found. Each servo was then tested individually by issuing commands for angle limitations, changes in baud rate, and changes in servo position. It was now established that remotely controlling single Dynamixels without R+ Manager worked as intended.

As the previously mentioned tests had determined limits on speed, safety angles, and temperature, it was also important to ensure that these parameters were enforced. The Dynamixel Workbench supported a YAML control file format which could be created that would contain Dynamixel parameters to be uploaded into the servos every time a controlling script executed, ensuring correct configurations even in case the control table values have been altered between executions. Some of the parameters included in the file were communication baud rate, individual IDs for the servos, acceleration and velocity limits, minimum and maximum encoder positional values, temperature limits, drive mode, and operating mode. The last two parameters determine what control table fields are used and how they are interpreted internally. The servos were also given descriptive aliases to more easily identify them in future code: 1_waist, 2_shoulder_left, 3_shoulder_right, 4_elbow, 5_wrist, 6_twist, 7_gripper.

A new issue was discovered when conducting tests in which the two shoulder joints were moved in unison. The motion was far from being smooth and seemed to stutter as if the two servos were fighting against each other. Even when ordered to hold position, stuttery micro-movement could be observed. The control table contains a field for the drive mode which dictates whether the servo adhered to “Normal” or “Reverse” mode, essentially mirroring any positional commands it receives. As one of the servos was set to “Normal” and the other to “Reverse”, this was not the root cause. The next attempt to fix the problem was by calibrating both of the shoulder servos. Calibration is a manual process that required disassembling the robot to disconnect the individual servos in an attempt to match their zero point as closely as possible. The problem was alleviated, but not entirely remedied by this process.

Once these elements were up and running, a backup of the entire setup was created to act as a bulletproof recovery point. With most of the essential software installed and configured, recovering from terminal mistakes was relatively effortless and quick.

5.3 Teach by Simulation Method

Once the basic setup had been completed, work started on the main focus of the project. The aim was to implement a simulation-based design that would allow the user to set the EOAT's position and orientation without necessarily worrying about other details. This method is heavily reliant on complex inverse kinematics calculations. For this purpose, MoveIt! is used for motion planning, after which a full simulation of the robot and its environment would be run in Gazebo in the safety of a simulation before bringing them to the real world. A high-level concept can be seen in Figure 13. As the teaching by simulation branch of development was terminated at a relatively early stage, many implementation details were never fully fleshed out. This section will describe the initial design concept on a general level.

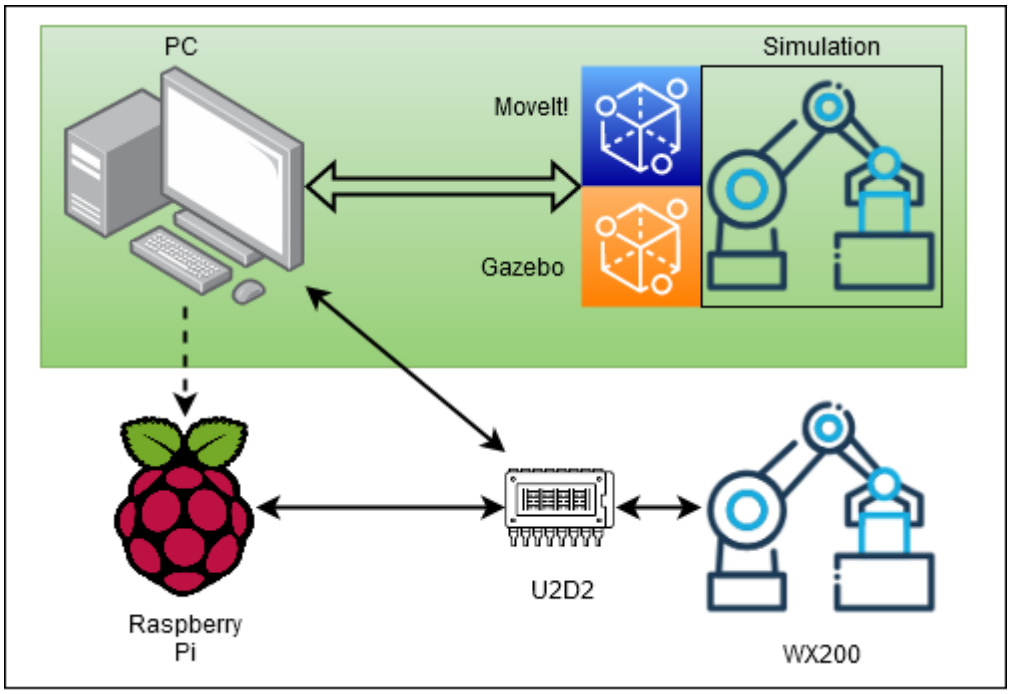

Figure 13. The created trajectories could be forwarded to the WX200 directly from the PC or transferred to a Raspberry Pi, depending which platform would be used as the robot controller.

In this design the user would use MoveIt! to plan and implement the desired sequence of motions. A screenshot of the MoveIt! User interface is illustrated in Figure 14. As MoveIt! is purely designed for motion planning, the created sequence is loaded up in Gazebo, where the robot's motion and any modelled 3D environment is simulated. These steps would be most suitable to be performed on a moderately powerful PC. The playback of motion sequences on the real-world robot would be performed either via Gazebo plug-ins or via a self-made client GUI program, which would communicate with the robot controller via network. These options would allow the user to command the robot remotely.

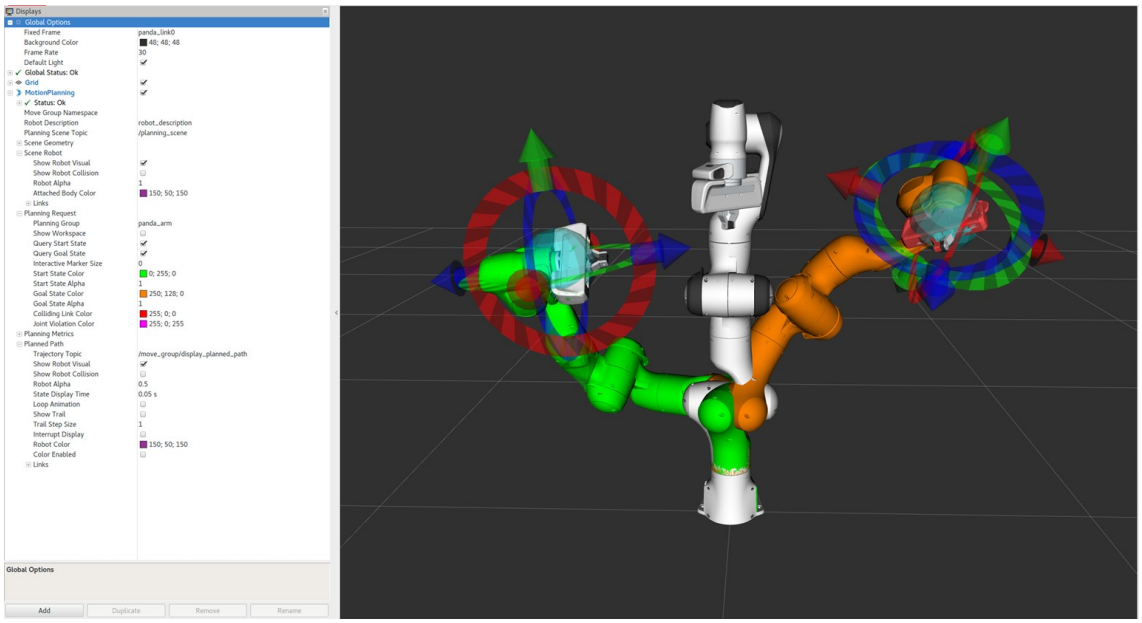

Figure 14. A screenshot demonstrating EOAT control in MoveIt! [15].

The simulation-based teaching system offers many useful features when designing trajectories for the robot. The first is simply the ability to inspect the movement of the robot in full 3D from any conceivable angle. The timescale adjustments made possible by the simulation allows for close-up scrutiny of intricate movements with time slowed down, or sped-up movements to get a better grasp of the entire sequence as a whole. Also, the interactable objects and any possible environmental features (i.e. obstacles) can be modeled into the simulation, which means the level of accuracy of the simulation is entirely up to the user. Another advantage of this teaching method is that the user does not require the hardware to be on hand. The work can be done virtually anywhere and by anyone without access to the hardware.

At some point along the development cycle a better understanding and feel of the user experience side of the design began to crystallize. It became apparent that translating real world environments, with significant accuracy, into a three-dimensional model is both difficult and time-consuming. This problem gets highlighted even more when considering frequent changes in either the robot's environment or the objects it interacts with. This could lead to the need for compromise where either the model would need more work to be accurate or the real-world environment would have to be changed to better suit the model. This would, in some cases, lead inevitably to more fine tuning of the model, as some real-world elements simply cannot be changed. This raised the question of the rate of iteration when creating new sequences or modifying existing ones. Another user-centric issue is that this method of teaching the robot would probably have a significantly longer familiarization time and a steep learning curve for the user to get fluent in using the program.

The other major issue with this approach was purely technical. Gazebo and MoveIt! were extremely prone to crashing. It was often difficult to run the program for more than a few minutes without a fatal error occurring. Many combinations of different versions of Gazebo and Move It! were tried, and none proved to be any stable than the other. Clean slate installations, including the OS and all programs, did nothing to solve the issue. In the end all effort in trying to solve these issues proved futile.

It is as of yet undetermined what the root cause of the Gazebo and Move It! crashes were, but in the end, it seemed to come down to two probable candidates. The first of these was the fact that our setup was running on a VM. It appears that some Intel processor families do not work as well as expected with virtual machines run on the VMware Workstation Player when it comes to OpenGL support. Another possible cause is be some kind of version incompatibility issue between ROS and MoveIt! or their supporting libraries. Both of these causes were too tied to the fundamental architecture of the project that, unless reasonably resolved, would lead to the abandonment of the simulation driven paradigm.

5.4 Teach by Demonstration Method

After experimenting with the simulation-driven solution, teaching by demonstration was tested to provide a more hands-on approach for the user. The main goal was to

implement a more simple, easily approachable system, that allowed for faster iterations in a changing robot environment.

The new teach by demonstration system would lead to the development of a client-side program called the Remote Robot Interface through which the user interacts with the robot. The RRI communicates with the robot controller, which houses the majority of the system's program logic. The robot controller's responsibilities are divided to four independent ROS Nodes. The general software architecture is portrayed in Figure 15.

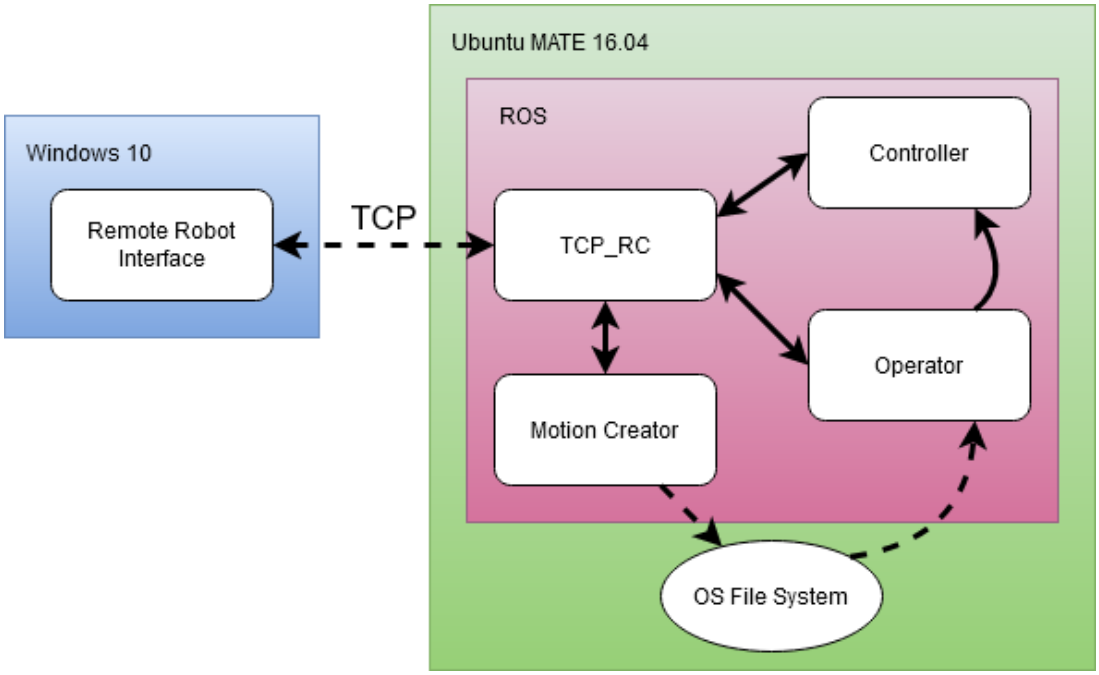

Figure 15. Overview of software architecture.

5.4.1 RRI

The Remote Robot Interface, or RRI, is a GUI client program for motion recording and playback. It is the only software component that the end user directly interacts with, as the rest of the software implementation is in the robot controller. A screenshot of the GUI is shown in Figure 16. RRI is compatible with any robot configuration that uses seven or fewer Dynamixel servos. Its main features are recording of several kind of robot movements, editing existing ones, combining sequences together into compound motions, and the playback of programmed sequences. Any single robot position is called a Pose, an Action is a chain of one or more Poses, and the linking two or more Actions creates a Routine.

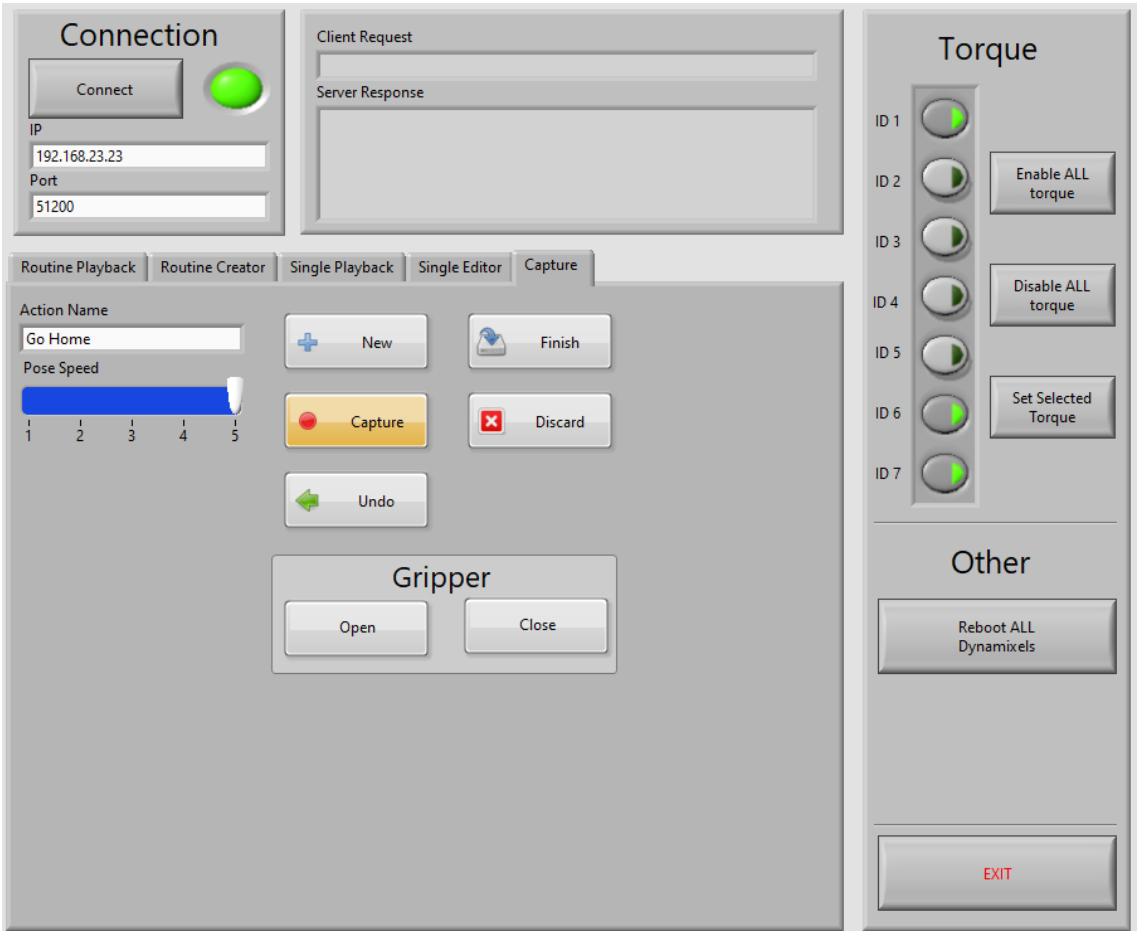

Figure 16. RRI graphical user interface.

Poses are recorded by physically moving the robot into the desired position and hitting the capture button. Recording single Poses by themselves is not useful, but any number of Poses can be stringed together to form an Action. An Action is roughly analogous to a function in programming, as it is a self-contained package that contains a set of sequential instructions. One such example could be a “Go Home” Action in which the robot would be programmed to neatly fold away into a neutral starting position. Another common example would be “Place Object On Pad”, which would drive it to perform the motion of dropping off something at a predetermined location. The Routine Manager feature allows queueing of Actions and the insertion of Operations before or after an Action. Available Operations are Wait Timers and Listening for a trigger signal from an external device. Figure 17 below demonstrates a Routine where the robot moves to the home position followed by a Wait Operation to allow time for external devices to initialize. Action 2 picks up a tag, followed by Action 3 that places it on an antenna. Operation 2

listens to the result from the measurement device and performs either Action 4a or Action 4b depending on the test result.

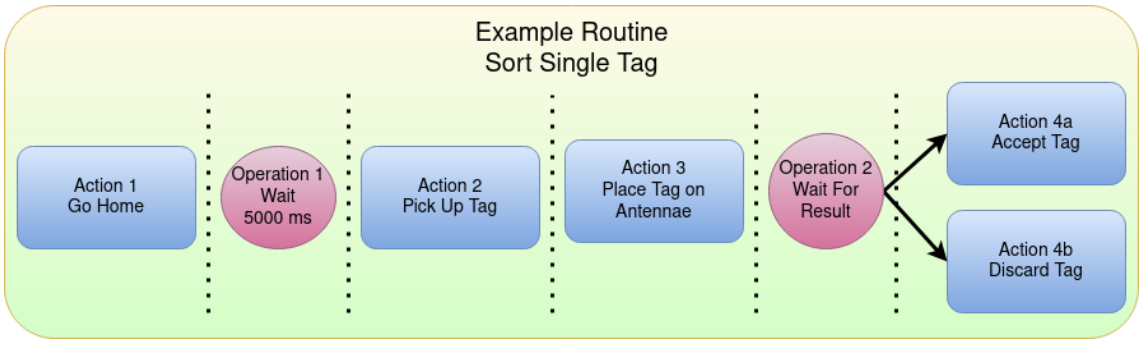

Figure 17. Routine Demonstration.

There are several features for finer control when programming new Actions. These include Pose recapturing for modifying single Poses within an existing Action, pausing and resuming an Action during playback, setting the desired playback speed of every single Pose, and single stepping through an Action one pose at a time or directly jumping to any specific Pose. It is also possible to enable and disable the torque of each Dynamixel individually, which helps in creating more fine-tuned Poses as the user can go “hands free” not having to hold every joint in place simultaneously. For example, all but three of the last joints could be frozen into place, which would allow the user to program the grippers exact position with finesse. Finally, the user may order for all connected servos to reboot themselves, which might be necessary in case of a thermal shutdown or if a servo becomes unresponsive.

RRI was sketched out and programmed in LabVIEW. As some of the main goals of the new teach by demonstration paradigm were simplicity and ease-of-use, a clear and pragmatic GUI was the obvious choice. This is where the value of using a programming language that inherently provides GUI programming capabilities was evident when considering time and effort saved to other unfamiliar options such as the Qt framework. RRI uses a simple producer-consumer software pattern, with the producer thread handling events and passing on user input to the main consumer thread, which is responsible for communicating with the back end via TCP and updating the GUI state. A secondary consumer thread contains the logic for Routines. As the back end housed in the robot controller itself has no understanding of Routines, it is purely reliant on client-side logic to string together actions and operations. The Routine process does however

require external status information at key points from the back end or other connected devices in order to execute at the correct time.

5.4.2 Back End

The rest of the architecture consists of what could be called the back end of the project. It is housed in the robot controller, which runs the Ubuntu MATE 16.04 operating system. Four distinct ROS Nodes split the responsibilities of the system. Two of these Nodes, TCP_RC and Motion Creator, were programmed from scratch. Joint Operator and Controller are Nodes from the Dynamixel Workbench that have been modified and expanded upon to suit the needs of the project. ROS is started and configured with a Bash script, with all of the aforementioned Nodes booted up with special ROS specific launch files. These launch files are used to pass information about the Node in question to ROS Master, for passing arguments to the Node, and for setting parameters for relaunching in case of fatal failures. All four Nodes will be further explored below.

The TCP_RC Node could be described as the communication hub of the system. It is a TCP server that handles communication to and from RRI, and issues requests internally to other Nodes. Facing outward, the most interesting part of TCP_RC is the implementation of an API for controlling the robot to which the client side RRI adheres to. When receiving and parsing a valid request from the client via network, TCP_RC contacts other Nodes via a ROS Service. The Node is robust and it will reinitiate itself in case of an unrecoverable failure, which was easily implemented using the Node's launch file. TCP_RC was programmed in Python, as it has a very capable Socket module. Another reason was the large amount of string related operations that the Node handles. Both of these factors would have been more costly in development time if implemented in C or C++.

Motion Creator is a ROS Node written in C++ that handles everything relating to capturing or editing robot motion. It has three main phases that the Node works through during a typical motion recording process. The first phase happens when the end user is in the process of recording an Action, during which Motion Creator keeps track of each joint's state and timing information. The second phase performs a validity check that the Action is coherent and complies with an internal rule set, after which the Action is properly formatted and written to disk as a YAML file. The third phase parses over newly written Action file and copies out each single Pose into its own separate action file. This phase

is necessary for it to be possible to step through or seek to specific Poses, as Action files are consumed by the Controller Node in their entirety. When editing Poses in pre-existing Actions, Motion Creator loads and edits the Action file in memory until the new Pose is finalized. It then overwrites both the original Action file and the individual Pose's sliced Action file. During all operations both successful and failed tasks, along with supplementary information about the recorded Action, are communicated back to the client software via TCP_RC.

The Joint Operator is a Node written by Robotis that is found in the Dynamixel Workbench library. The Node was modified to conform to the new architecture, and new features were added for the end user to manage available Actions. These structural modifications and additions form a third of the Node's final size. The primary function of the Joint Operator is to parse out an Action file for the raw positional and temporal data, namely information on joints, Poses, and timings from an Action file. The formed bundle is then published on the joint trajectory channel for the Controller Node to read. The newly added functionalities mostly relate to the reading, modifying, and deleting of Actions in the file system and informing the user of readily available Actions and Poses for playback or deletion.

The Controller is the main workhorse Node of the setup when dealing with actual raw calculations and issuing commands to any connected servos. It is responsible for finding and initializing the necessary Dynamixels, reading published joint trajectory messages from Joint Operator, performing trajectory calculations, and communicating the trajectories to the servos. These functionalities were written by the Robotis team. To make the controller more suitable for this thesis' application, a number of pre-existing functions were modified to signal their in-progress status and completion in order to support Routines. Also, modifications to the execution of motion were made to facilitate stopping, pausing, and resuming Actions. The Controller also handles Service requests for rebooting Dynamixels and for issuing torque commands.

The teach by demonstration paradigm offers several advantages over its predecessor that are worth noting. The most obvious one of these, from the user's perspective, is the overall simplicity of the system. Without the need for extensive training, modelling the robot's work environment, and using MoveIt!, the end-user is able to create a working robot Actions within the first minutes after being introduced to the system. This approach is also quite versatile. In the case that a testing or showcasing setup varies ever so

slightly from case to case, or if the system is used for ad-hoc jobs, it is easy and fast to accommodate these kinds of changes. This is further expedited if the only change in the environment is an object's position or size, as any individual robot Pose can be modified on the fly. The robot's physical properties also facilitate the teach by demonstration method, as the light and small frame of the robot is easily manipulated by hand.

6 Conclusions

The outcome of the project was an easily programmable robotic system with a user-friendly graphical interface and a feature rich back end. The setup is suitable to be run on either a single computer, such as a laptop, or housing the back end within a small form-factor single-board computer. Due to unexpected problems during development, the project did not reach the point at which the case company's RFID measurement equipment was integrated to the system.

The primary goal of this thesis was the mapping and exploring of the possibilities and challenges of robot technologies, especially in the context of in-house development. The project's exploratory and experimental nature led to a development cycle that saw several shifts during the project's lifetime. It was quite often that a new discovery or a technological impasse led development towards unexpected directions. In the long run, time spent on experimenting with different paths turned out to be an unexpectedly valuable resource. Working out practical solutions for each problem granted experience and unique views of the situation at hand. As is the case when covering unfamiliar ground, mistakes can provide more information than arriving directly at a solution. If this goal was to be rephrased as "learn what pitfalls to avoid in small-scale robot automation projects", the project succeeded with flying colors. The case company benefitted from this thesis by growing its capacity to produce more accurate and realistic project specifications in any possible future robotics endeavors.

The secondary goal of creating a prototype system aimed for marketing and small-scale automation in-house was not achieved. The current prototype could be expanded upon to meet these goals with more time and effort, but there are indicators that suggest this might not prove to be a prudent solution. Some of these reasons and challenges will be highlighted further in this section.

An early indication of the inadequate quality of hardware in use was when one of the two cheaper XL430-W250-T Dynamixels was damaged beyond repair. Despite wide thermal margins, the servo burned out while lifting a foam cube weighing around five grams. A possible solution would have been to replace both XL430-W250-T servos with more expensive Dynamixel counterparts that communicated using the same protocol. However, even the slightly pricier XM430-W350-T may not have sufficed, as the next challenge will demonstrate.

What ultimately led to the decision to halt development were problems with servo backlash and gear fatigue. The issue arose when attempting to teach trajectories and positions that required any degree of precision. The expectation is that the robot will mimic whatever positions were recorded as accurately as possible, especially if the application requires the manipulation of small objects, as is the case with RFID tags. However, the playback of the Action reveals that the positions are off-target by a significant margin. The cumulative looseness due to backlash and gear fatigue combined with the link lengths of the robot reduce the accuracy through the kinetic chain, resulting in the misalignment of the EOAT. This error makes object handling and movement near obstacles precarious, as what was thought to be a simple grab-and-move would cause the robot to collide with its environment. The reason this was not noticed sooner is because the issue grew worse over time, which can probably be attributed to gear fatigue. What started out as a minor issue developed into a debilitating flaw over time. It is worth noting that this problem is not exclusive to the teach by demonstration paradigm, as it would have affected the simulation driven solution in a similar way.

Two possible ways of either compensating for or averting the backlash and gear fatigue error were planned. The first possible solution would have been to compensate for the error in software. However, even with adjustable error correction, the continued wearing down of the servo gears would remain a problem. Additionally, as the error varies from servo to servo, a general solution would be difficult to implement. The other solution would be to disassemble the robot and change it into a SCARA configuration. With all but one or two of the servos oriented in parallel rotating around the vertical axis, the configuration could possibly eliminate the drooping of the arm. The downside to this is the major loss of vertical dexterity, as the SCARA configuration would barely have any vertical mobility. As of now both solutions remain as possible future development opportunities.

Based on the experience gained during the project, it is clear that developing an in-house solution for small-scale robotic automation is definitively feasible. There are excellent tools, such as ROS and its innumerate libraries and modules, that are open-source, accessible, and ever gaining in popularity that lower the barrier to entry for parties interested in robotics. The field does however require a baseline of experience. One possible solution could be to hire a contractor to bootstrap the process by providing area specific expertise, expediting development and guiding around possible pitfalls through the initial stages of development. Another area worth investing in would be higher quality hardware, mainly concerning the robotic manipulator itself. Purchasing a small, high-quality robotic arm for the prices starting at tens of thousands of Euros is understandably out of the scope of an exploratory thesis work, but quite feasible if considering a more involved robotics related R&D project for a company.

Many of the shortcomings during the project's lifespan seemed to illustrate the difficulty of selecting tools based merely on technical specifications, as the details on paper may not translate one-to-one to the real world. A degree of familiarity with the technical field in question can go a long way to saving in material costs and development resources. The experience gained from the thesis may also be utilized in future robotics projects to better gauge the real-world performance of robotics related hardware and when purchasing contract work for related projects.

There is some truth to the humorous adage used in embedded systems: "Rule one: Read the data sheet. Rule two: Read the data sheet again. Rule three: Don't trust the data sheet."

7 Summary

The thesis was commissioned by the case company primarily to build a foundation of both theoretical and practical knowledge about the design of small-scale robotics applications relating to RFID tag testing. A secondary goal was to attempt to implement a prototype for marketing purposes and small-scale automation within the company.

Central concepts relating to robotics and RFID were researched to provide better understanding for the design and implementation of a teachable robotic manipulator system. The system design was done by leveraging research, following guidelines

established in the industry, and by utilizing the feedback from within the case company. The iterative process of drafting requirements and a design approach focusing on elimination led to the specifications of the prototype build.

Two methods of implementing a teachable robot system were explored during the project. The first method was a simulation driven model, while the second method followed a teach by demonstration model. The initial implementation was abandoned in favor of a new development direction due to new realizations concerning the user experience and technical limitations. The final outcome of the second development direction was working prototype of a programmable robotic system with an accessible front end, that connects via network to the back end housed in a robot controller.

The thesis achieved its goals of exploring and refining the design aspect and pitfalls of developing in-house small-scale automation with robotics for various applications. Development was eventually halted due to the robot hardware failing and not performing to specifications. The developed prototype was not used in its intended role due to servomotor related hardware issues. However, the thesis supplied the case company with a solid knowledge base that will prove useful in any future robotics undertakings. Finally, possible solutions for alleviating the hardware issues are discussed, along with insights on how to improve any future development endeavors.

References

1 Jazar, Reza N. 2010. Theory of Applied Robotics: Kinematics, Dynamics, and Control, Second Edition. Springer.

2 Kandray, Daniel. 2010. Programmable automation technologies : an introduction to CNC, robotics and PLCs. Industrial Press.

3 What Is A Work Envelope? RobotWorx. Web reference. <https://www.robots.com/faq/what-is-a-work-envelope>. Read 14.2.2019.

4 Failure Modes is Gears. 2008. Web reference. Bright Hub Engineering. <https://www.brighthubengineering.com/cad-autocad-reviews-tips/8443-failure-modes-in-gear-part-one/>. Read 17.2.2019.

5 Matarić, Maja J. 2007. The Robotics Primer. MIT Press

6 Methods of Programming Robots. 2018. Web reference. DZone. <https://dzone.com/articles/methods-of-programming-robots>. Read 14.2.2019.

7 John J. Craig. 2004. Introduction to Robotics: Mechanics and Control (3rd Edition). Prentice-Hall.

8 Bajt, Tadej; Mihelij, Matja. 2010. Robotics. Springer Netherlands.

9 Gallardo-Alvarado, Jaime. 2016. Kinematic Analysis of Parallel Manipulators by Algebraic Screw Theory. Springer International Publishing.

10 Collins, Danielle. 2005. Sizing and Selecting Linear Motion Systems: How “LOSTPED” Can Help. Web reference. <https://www.eetimes.com/sizing-and-selecting-linear-motion-systems-using-lostped/> . Read 12.3.2019.

11 Finkenzeller, Klaus. 2010. RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Indentification and Near-Field Communication (Third Edition). Wiley.

12 EPC Radio-Frequency Identity Protocols Generation-2 UHF RFID Standard: Specification for RFID Air Interface Protocol for Communications at 860 MHz – 960 MHz, Release 2.1 . 2018. EPCglobal Inc.

13 WidowX 200 Robot Arm. Trossen Robotics. <https://www.trossenrobotics.com/widowx-200-robot-arm.aspx>. Read 19.5.2019.

14 Mok, Kimberley. 2020. Robotics Operating System Brings Open Source Approach to Robotics Development. Web reference. <https://thenewstack.io/robotics-operating-system-brings-open-source-approach-to-robotics-development/> . Read 5.8.2019.

15 MoveIt Tutorials. Web reference. <http://docs.ros.org/en/melodic/api/moveit_tutorials/html/index.html>. Read 19.4.2020.

16 McCarthy, J Michael; Gim, Song Soh. 2010. Geometric Design of Linkages. Springer Science & Business Media.

17 Gonzales, Carlos. 2016. What's the Difference Between Industrial Robots? Web reference. <https://www.machinedesign.com/markets/robotics/article/21835000/whats-the-difference-between-industrial-robots>. Read 7.10.2020.