DEVELOPMENT OF A USER INTERFACE TO COMMUNICATE WITH VIRTUAL DELTA ROBOT USING MODBUS TCP/IP AND C#

A Thesis

Presented to

the Faculty of the College of Business and Technology

Morehead State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

Armin Maraghehmoghaddam

April 22, 2016

Accepted by the faculty of the College of Business and Technology, Morehead State University, in partial fulfillment of the requirements for the Master of Science degree.

_____
Dr. Jorge Alberto OrtegaMoody
Director of Thesis

Master's Committee: _____, Chair
Dr. Jorge Alberto OrtegaMoody

_____
Dr. Ahmad Zargari

_____
Dr. William R. Grise

_____
Date:

DEVELOPMENT OF A USER INTERFACE TO COMMUNICATE WITH VIRTUAL DELTA ROBOT USING MODBUS TCP/IP AND C#

Armin Maraghehmoghaddam
Morehead State University, 2016

Director of Thesis: _____
Dr. Jorge Alberto OrtegaMoody

Today, industrial robots are an integral part of the automotive assembly lines due to their higher speed, quality, reliability and productivity. Considering the wide applications of robots in almost all various fields in the modern world, there is always need for engineers, operators, technicians and experts to perform tasks of programming, maintenance, operating and troubleshooting of robots. Training professionals and experts in the robotic areas is necessary due to the critical processes in which robots are involved. However, the main constraints regarding providing continuous training in the field of robotic technology include lack of training resources, unsatisfactory training processes, high costs of equipping robotic laboratories, high sensitivity of working with robots for unskilled individuals, as well as high risks of making mistakes and damaging robots. As a solution, virtual robot laboratories are developed to resolve such issues. The whole concept of virtual robot training is based on implementing virtual robot laboratory and virtual robot with the exact behavior of the actual robot. Applying both virtual robots and virtual

robot laboratories, trainees are made able to implement various scenarios and code various sequences. As a result of applying such scenarios and sequences, trainees can monitor the real behavior of the robots. Trainees would also find the opportunity to practice all possible conditions as a result of scripting various scenarios.

For implementation of virtual delta robot with the actual behavior of a real delta robot, game engines came into the account. The capabilities of game engines such as physic engines inside them, real time simulation, and availability helped the simulation of the virtual robot. For the users' effective interaction with the virtual delta robot, development of a user interface is required. The topic of this thesis started at this point. Using the virtual robot, mathematical modeling and mathematical kinematics of delta robot, and communication methods based on Modbus TCP/IP, a user interface for facilitating user's interaction with the delta robot has been developed.

The design of the user interface is based on the basic needs of users which are control of the robot considering kinematic models, positions of the end effector and saving these positions, communication setting, robot dimensions setting, and programming and scripting.

As the output of the thesis, a virtual delta robot, which was maintained in a game engine, can be easily manipulated through the developed user interface in which robot kinematic models and communication based on Modbus TCP/IP came into the account.

Accepted by: _____, Chair
Dr. Jorge Alberto OrtegaMoody

_____
Dr. Ahmad Zargari

_____
Dr. William R. Grise

Table of Contents

Table of Figures

1 Introduction

The high level of popularity of the robotic technology is mainly due to its vast applicability across great number of fields. Robots have been applied in industry, healthcare and medical surgeries, military, agriculture, oceanographic explorations, education, and for aerospace purposes. The advances in computers as the brain of robots leads to more improvements in robotic technology in that robots are not simply mechanical machines, but they are turning into more intelligent machines with the ability to process information much faster and more efficiently in comparison to earlier versions.

1.1 Industrial Robots

In recent years, manufacturing processes have become more autonomous which require less operator intervention and higher flexibility for specific applications to meet specific market demand. Industrial robots play a crucial role in various fields of automated industry. Industrial robots are electronically controlled, and are programmable and reprogrammable so as perform specific and varied duties in industrial and manufacturing lines. Robots have brought about innovations and efficiency in manufacturing sectors and play a crucial role in industrial innovations. Despite the fact that the programming of industrial robotic systems for a specific application in industry is difficult, time-consuming and can be expensive, industrial robots are considered the best solution for both productivity and flexibility which is crucial in production diversification in the era of globalization (Pan, Z., Polden, J., Larkin, N., Duin, S. V., & Norrish, J., 2012). Today, industrial robots are an inescapable part of the automotive assembly lines since they have the ability to perform and accomplish specific tasks with higher speed, and result in higher quality, reliability and productivity. In addition to the higher level of flexibility, quality and

reliability, industrial robots play an important role in balancing production costs, time, and quality constraints (Papakostas, N., Michalos, G., Makris, S., Zouzias, D., & Chryssolouris, G. , 2011).

1.2 Robots and Hazardous Tasks

One of the greatest justifications for robotic applications is performing tasks that are dangerous for human beings to do. Robots are ideal for working in hazardous environments, so that people can be relieved from performing dangerous tasks in unfriendly conditions. According to the Robotic Industrial Association, robots are commonly used in sites where chemicals and hazardous materials are handled on a daily basis (Brumson, Robotic Industrial Association, 2007). By reducing workforce exposure to hazardous environments by taking advantage of chemical and material handling robots, companies aim to decrease potential liability for workers' compensation or other related costs. In the process of handling hazardous materials, there are irritating or toxic fumes- such as fumes created during ultrasonic welding of some kinds of plastic parts- that bother human workers especially if there is close or prolonged exposure. There are also explosion-proof robots that are employed in environments where ignitable fumes and vapors exists. Also, robots are applied in handling non-hazardous materials that can produce potentially explosive dust including bag palletizing of grain products. There is a growing demand for use of robots in handling hazardous materials as the number of chemicals used in industry continues to increase, with the resulting increase in the potential environmental and health risks. It is predicted that the future robotic technology will have improved vision and sensory equipment in order to handle such hazardous chemicals safely, and successfully (Brumson, Robotic Industrial Association, 2007). In addition, robots are capable of lifting heavy loads repetitively without tiring or injury. They increase worker safety and prevent accidents by removing workers from hazards. Robots significantly save production time, since they are able to produce great numbers of products. They

also save companies in the long term by decreasing the volume of wasted materials, reducing workforce injuries, using less materials, and thus generating a quicker return on investment. In addition, robots have created and developed new occupations. The production line moving over to programming tasks, thus reducing of monotonous jobs and adding them to more challenging ones. In the USA, robots allow companies to remain competitive, keeping local jobs (ScottCompany, n.d.).

1.3 Robots and Technology Development

Robotics has continued to have a profound influence on technology developments in various fields, some of which are presented here. One of the impressive technology developments in robotics is assistive robotics. Assistive robots generally include robots designed to aid people with special needs (Brian Scassellati, Henny Admoni, Maja Mataric, 2012). The primary application of assistive robotics is to provide hands-on treatment or support for physical disabilities by helping a patient perform repetitive therapeutic motions as a physical therapist would. Another technology is called social robotics, which involves robots that engage in some form of social interaction with humans, through speech, gestures, or other modality. One of the recent developments in this field is Social and Assistive Robotics (SAR), which lies at the intersection of social robotics and assistive robotics. SAR refers to the design of robots to help through social interaction rather than physical interaction. This technology is now used to conduct research for treatment of disorders such as autism (Brian Scassellati, Henny Admoni, Maja Mataric, 2012). There are many advances in the field of human- robot interactions in various industries. Military robots are also another development. They are autonomous remote-controlled robots that are used with military applications in transport, search, rescue, attack and so forth. Although there have been lots of development in the field of robotics so far, many researchers are involved in making robots more

efficient, more widely applicable, and more intelligent in various field, with more specifications to their systems (Brumson, Robotic Industries Association, 2011) (Grabianowski, 2016).

1.4 Robots and Training

As previously mentioned, robots are applicable in various fields from agriculture, manufacturing, arc welding, healthcare and treatment, to aerospace applications and space exploration. Thus, there is always need for engineers, operators, technicians, and other experts to perform the tasks of programming, maintenance, operating and troubleshooting of robots.

In order to have a well-prepared and skillful workforce in the field of robotics, continuous and updated training is essential. Training professionals and experts in the robotic areas is necessary due to the critical industrial and other processes in which robots are involved. However, the main constraint on providing continuous training is the lack of resources. Inadequate training resources leads to a not very fulfilling training process. The high costs of providing robots for the purposes of training individuals is a major obstacle. In addition, the higher risk presented by unskilled individuals working with robots increases the risk of making mistakes and damaging robots limits the availability of robots for training and also imposes extra costs.

Ideally, all trainees and students who are earning educations in fields of robotics and robotic technology should have access to adequate training resources such as a robotics lab in which they can apply different scenarios to the robot and monitor the behavior of the robot. Being able to do so, trainees will develop skills in applying instructions, programming orders and scenarios in programming, reprogramming, controlling, and maintaining robots and understanding robotic procedures and configurations. In addition, trainees should be able to implement intentionally some various types of errors so that they can see the behavior of the robot and also learn the troubleshooting process.

All these conditions above define the ideal situation for the training. However, not all of these conditions are not possible. Even if possible, the process of practicing troubleshooting can cause serious damages to the robots as well as extra expenses. So, the question is how to deal with this problem considering the fact that training programs in robotic technology need robotics equipment. To clarify the constraints in providing training programs with robotics equipment, the drawbacks of using robots in training sessions are discussed in the following.

There are three main drawbacks for using robots in training sessions.

In the first place, robots are expensive. Considering the number of trainees and students who need such training sessions, significant financial resources are required to buy robots. Also, proper laboratories need other equipment as well which cannot be avoided. Besides, the laboratory buildings themselves will require significant resources to build and/or retrofit, and maintain.

In the second place, the vulnerability of robots can lead to technical problems during the training sessions. Students and trainees are usually working with robots in laboratories during training sessions; at the same time, they are not fully aware of the capabilities of, and restraints on, robots. Consequently, using wrong codes and instructions can sometimes cause serious problems and damages to the laboratory equipment. In addition to the extra costs incurred for repair or replacement of damaged equipment, while the equipment has problems, it cannot be used by other trainees and students until it is repaired, which is a waste of time. If this downtime occurs in the beginning sessions, it might even impair the learning and training process for a while. Such a loss of valuable training time is hard to compensate for, as the impact of theoretical sessions can be best realized when done in conjunction with practical sessions in the laboratory.

Most importantly, safety issues are serious problems in robotic technology laboratories. Such safety issues arise comes from a lack of adequate professional supervision while working

with robots in the laboratory. In other words, most robotic training sessions are run and supervised by only one instructor or trainer. This means that the instructor cannot simultaneously be present for each and every trainee while he or she is using the robots and performing laboratory tasks. This inadequacy of supervision can be lead to serious safety issues, since trainees are mainly beginners who are totally new to running robotic programs while working with robotic systems and processes.

The process of training students in robotic technology is a complicated process. One set of complications arises due to trainees' often inadequate preparatory knowledge in relevant fields. In other words, students being trained in the robotic laboratory are required to have gained general knowledge in fields of mechanical design and electrical design. They also need to have understanding of the concept of mathematical modeling of robots and behavior of robots in addition to a profound grasp and knowledge of programming skills and procedures. The robots used in laboratories are almost always offered in compact packages. Thus, teaching the concepts of mechanical and electrical design as well as programming and mathematical modeling in the training period is almost impossible for the instructor. Considering the fact that trainees are not able to see the insides, i.e., the design of the robots they are working with during their robotic training, trainees will not be easily able to successfully program robots, figure out and analyze various outputs and behaviors of robots for different scenarios. In addition, programming actual robots is a very complicated process which needs extra attention and precision because any wrong program or sequence can lead to serious damages to the robots and also can cause safety issues for operators and trainees.

1.5 Virtual Reality

Although early elements of virtual reality can be traced back to the 1860's and long before the development of digital technology, it is usually considered to have started in earnest. During the 1920's, Edwin Link developed the world's first flight simulator as a training device for new and novice pilots. Later on, the first kind of multimedia device in the form of an interactive theatre experience was devised by Morton Heilig in 1957 as an early form of virtual reality which was not patented until 1962. The development of virtual reality continued with the technology of head mounted display (HMD) that was designed to be used by helicopter pilots so that it made them able to see their surroundings during night flights. The HMD was attached to a computer in 1968 to enable the wearer to see a virtual world. However, it had to be attached to a suspension device due to its weight. In 1970's, an innovative form of multimedia as the first interactive map of Aspen was developed in Massachusetts Institute of Technology (MIT) enabling people to walk through the town of Aspen. During 1980's, virtual reality was used for NASA projects along with other techniques derived from researches in new forms of human – computer interaction (HCI). In 1990's, virtual reality became public and grabbed public awareness as Jaron Lanier and Tom Zimmerman marketed a range of virtual reality gear. Although the hypes surrounding virtual reality technology in 1990's had an initially adverse effect on the popularity of such technology, there are now various advantages that can be obtained from its application (Virtual Reality Society, 2016).

1.6 Virtual Environment and Training

Virtual environment and the concept of virtual reality came to fore when the idea of eliminating dangers and expenses in hazardous and expensive training procedures was created (Maurizio Rovaglio, Tobias Scheele, 2011) (Etienne van Wyk, 2014). The three main issues of

using actual robots in robotic technology laboratories including the high cost of equipment, lack of resources, the vulnerability of robots to wrong instructions and codes, the serious damages caused by unskilled users, and safety issues related to the application of actual robots, are all reasons for the increasing trend toward using virtual environment.

The whole concept of virtual robot training is based on implementing a virtual robot laboratory and virtual robot with the exact behavior of the actual robot. Applying both virtual robots and virtual robot laboratories, trainees become capable of implementing various scenarios and coding various sequences. Also, they can monitor the real behavior of the robots as a result of applying such scenarios and sequences. Trainees would find the opportunity to practice all possible conditions as a result of scripting various scenarios. In addition, trainees can observe how wrong codes can cause errors and faults on the robot without imposing any extra maintenance costs that are unavoidable if using actual robots. Thus, trainees can continue to practice at no costs and have the opportunity to run one set of instructions several times to master the problem which is not possible via applying actual robots. The ability of repeating and performing various scenarios, errors and codes also make the instructor more confident about the risks of safety issues. So, applying virtual robots instead of actual ones for the purpose of training can highly enhance the problems with the application of actual robots and make robotic technology training much less stressful for both trainees and trainers. The combination of platforms for virtual developments, physics engines, computer processors, and graphic processors are used to implement a virtual robot laboratory (Moody, J. O., Sánchez- Alonso, R., Yun, C., González- Barbosa, J., & Reyes- Morales, G., 2015).

1.7 Scope of the Thesis

There are various significant drawbacks to the application of actual robots in robotic technology laboratories. These include the high costs of building, and equipping robotic laboratory as well as the costs of actual robots, vulnerability of robots to wrong coding, programming and error instructions leading to serious damages to robots, and safety concerns related to lack of adequate supervision on trainees while performing relevant procedures.

The main issue of this research is to apply the combination of platforms for virtual developments, physics engines, computer processors, and graphic processors to implement a virtual robot laboratory. By applying virtual robot laboratory to a training or instructional setting, not only all the previously mentioned constraints of actual robotic technology are removed, but also trainees gain the opportunity to spend more time practicing and performing various coding and programming scenarios to make training sessions fruitful, safe, challenging and robotically successful. This study explains the process of creating such virtual reality environment by presenting the mathematical modeling, programming and coding for a virtual parallel delta robot.

2 Review of Literature

2.1 Kinematics of Delta Robot

2.1.1 Delta Robot Configuration

The Delta robot is a type of parallel robot. The Delta robot is an ideal candidate for pick and place operations of light objects in manufacturing and production lines due to its fast pace and movement flexibility. The key design feature in the Delta robot is the use of parallelograms. A parallelogram maintains the orientation of an end effector. In other words, a parallelogram allows an output link to maintain a fixed orientation considering an input link (Brogardh, 2000).

The Delta robot is depicted in Figure 2-1. As shown in Figure 2-1, it consists of various parts as follows (Williams, 2016):

- A fixed platform
- Three active revolute joints
- Three upper links
- Six passive universal joints
- Three parallelogram links
- A mobile platform
- The end effector

In the case of this study, the fixed platform consists of an equilateral triangle. The fixed platform is affixed to a base and thus, does not move at all.

As pointed in Figure 2-1, the three revolute joints allow single-axis rotation around the axis. These revolute joints are called “active” since we placed motors on these joints in order to provide the source of motions for the robot (Williams, 2016).

The three upper links are connected to the fixed platform through the revolute joints from the top and are connected to the parallelogram links via universal joints from the bottom (Williams, 2016).

Universal joints, as showed on the Figure 2-1, allow the transmission of power by providing rotation at the joints. Applying the universal joints, the power of motors is transmitted to the mobile platform that makes the mobile platform move freely in the robot working space (Williams, 2016).

Parallelogram links consist of two parallel links. These parallel links shape a chain between the upper links and the mobile platform. A parallelogram allows an output link to maintain a fixed orientation with regard to an input link (Williams, 2016).

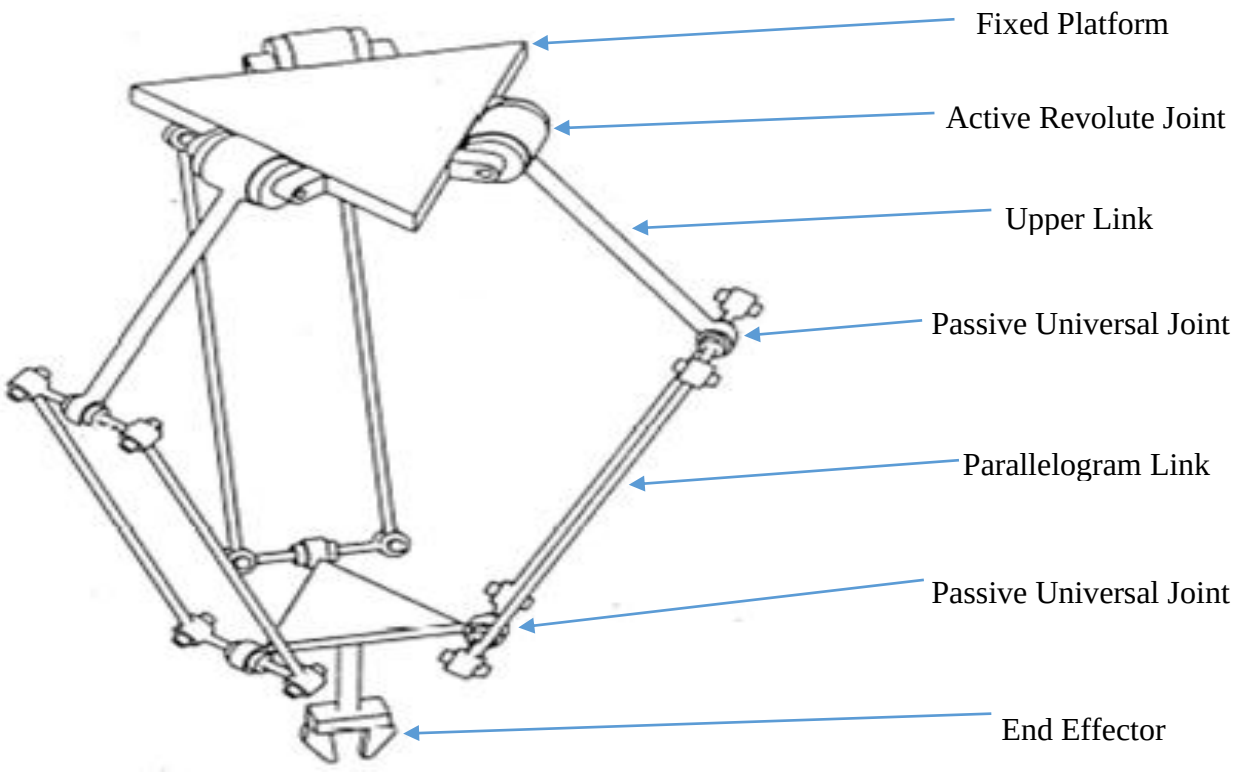

Figure 2-1: Delta Robot Configuration (Williams, 2016)

2.1.2 Mathematical Modeling of the Delta Robot

In this study, the effort is made to design and implement virtual robot by taking advantage of the combination of platforms for virtual developments, physics engines, computer processors, and graphic processors to implement a virtual robot laboratory. This can be done by creating the virtual reality environment through presenting the mathematical modeling, programming and coding for a virtual parallel delta robot.

The main tool for implementing the virtual Delta robot and programming the compatible user interface is the mathematical modeling of the Delta robot. When it comes to the user interface, the key element is modeling of the robot movements and the behavior of the virtual robot during the time the virtual robot is operated by the students, trainees, instructors or operators. The implementation of the mathematical modeling behind the user interface scene is highly significant due to the fact that such implementation provides the ground for the operator instructions to be transferred to the virtual robot movements.

Movements of the Delta robot are based on two major concepts of inverse kinematics and forward kinematics. This generally refers to the relations between the joint angles and the end effectors. Simply put, in the concept of forward kinematics, the input consists of the joint angles while the output is the coordinates of the end effectors. On the other hand, the input consists of the coordinates of the end effectors, and the joint angles calculations are the output (Msavatsky, 2009).

2.1.3 Inverse Kinematic

The first concept to discuss here is the inverse kinematics method. Considering the desired position of the end effector to be known, the purpose is to find the joint angles. In other words, while we know the coordinates of the end effector, we need to figure out how we should change the motors angles so that we reach the desired known position. So, figuring out the calculations of

the three angles of the motors in the way that we can reach out to desired position of the end effector is the goal.

In the process of mathematical modeling of a robot, knowing the precise measurements of the robot is the most important part. As previously mentioned, there are two platforms in the configuration of the Delta robot called the fixed and the mobile platform. Both of the fixed and mobile platforms in the scope of this study are considered as two equilateral triangles. There are also three kinematic chains, each of which consists of two links. Figure 2-2 shows the angles of θ and the coordinates of X_0, Y_0, and Z_0. As it is clearly depicted in Figure 2-2, the variables are the angles of the three motors as θ_1, θ_2, θ_3 and the position of the end effector which is pointed as E_0. The coordinate of the end effector is X_0, Y_0, and Z_0 (Msavatsky, 2009).

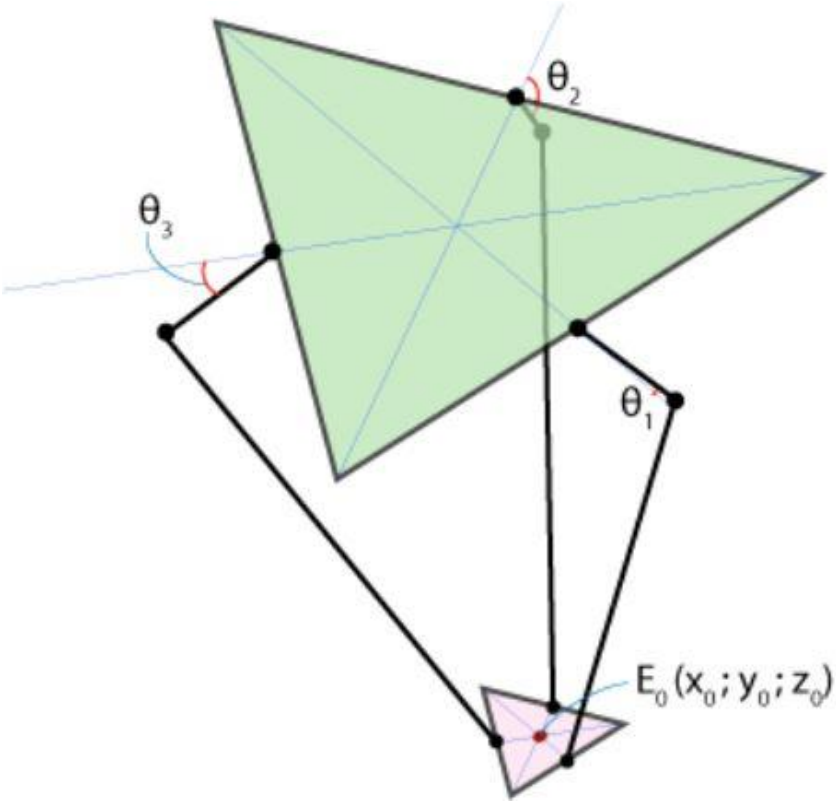

Figure 2-2: Delta Robot Variables (Msavatsky, 2009)

In the next step, the goal is to determine the key elements of the robot geometry. Finding the robot geometry is the important path to figuring out the motor angles based on the coordinates of the end effector position.

Figure 2-3 displays other specifications about both of the fixed and the mobile platforms. It explains “f” as the side of the equilateral triangle of the fixed platform. Also, “e” is pointed as the side of the equilateral triangle of the mobile platform. Other specifications are depicted as “r_f” which is the length of the upper link, and “r_e” which is the length of the parallelogram joint.

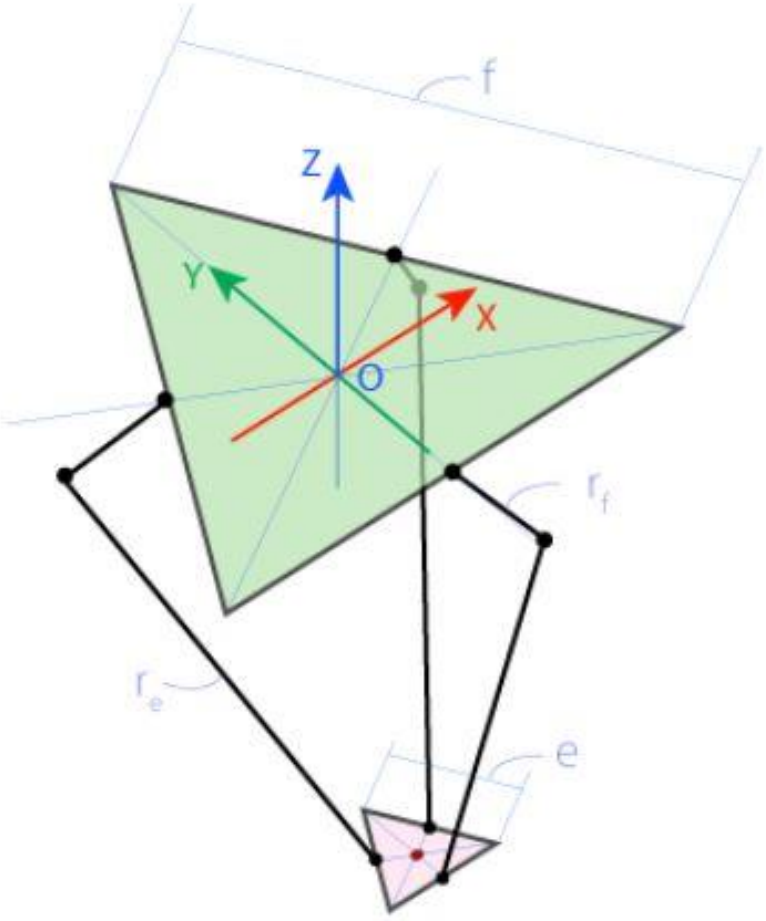

Figure 2-3: Robot Dimensions (Msavatsky, 2009)

So far, we have defined and determined all the variables that are important in doing the mathematical modeling for the virtual robot as desired. So, we are now able to shape the required equations and do the proper calculations which define the movements of the robot and its behaviors with regards to the determined robot geometry.

In order to properly perform and complete the mathematical modeling of the Delta robot, defining all possible movement of the robots is the first step. Referring to Figure 2-1, it is seen that all movements of the robot begins from the active revolute joints, where the motors are placed. Revolute joins allow single- axis movements and going forward, universal joints allow rotational movements. In other words, revolute joints shape a circle moving around the one axis whereas universal joints shape a sphere with rotational movements around two axes. Both the circle-shaped area created by the revolute joints single-axis movements and the sphere-shaped area resulted from the rotational movements allowed by the universal joints are depicted in Figure 2-4 (Msavatsky, 2009).

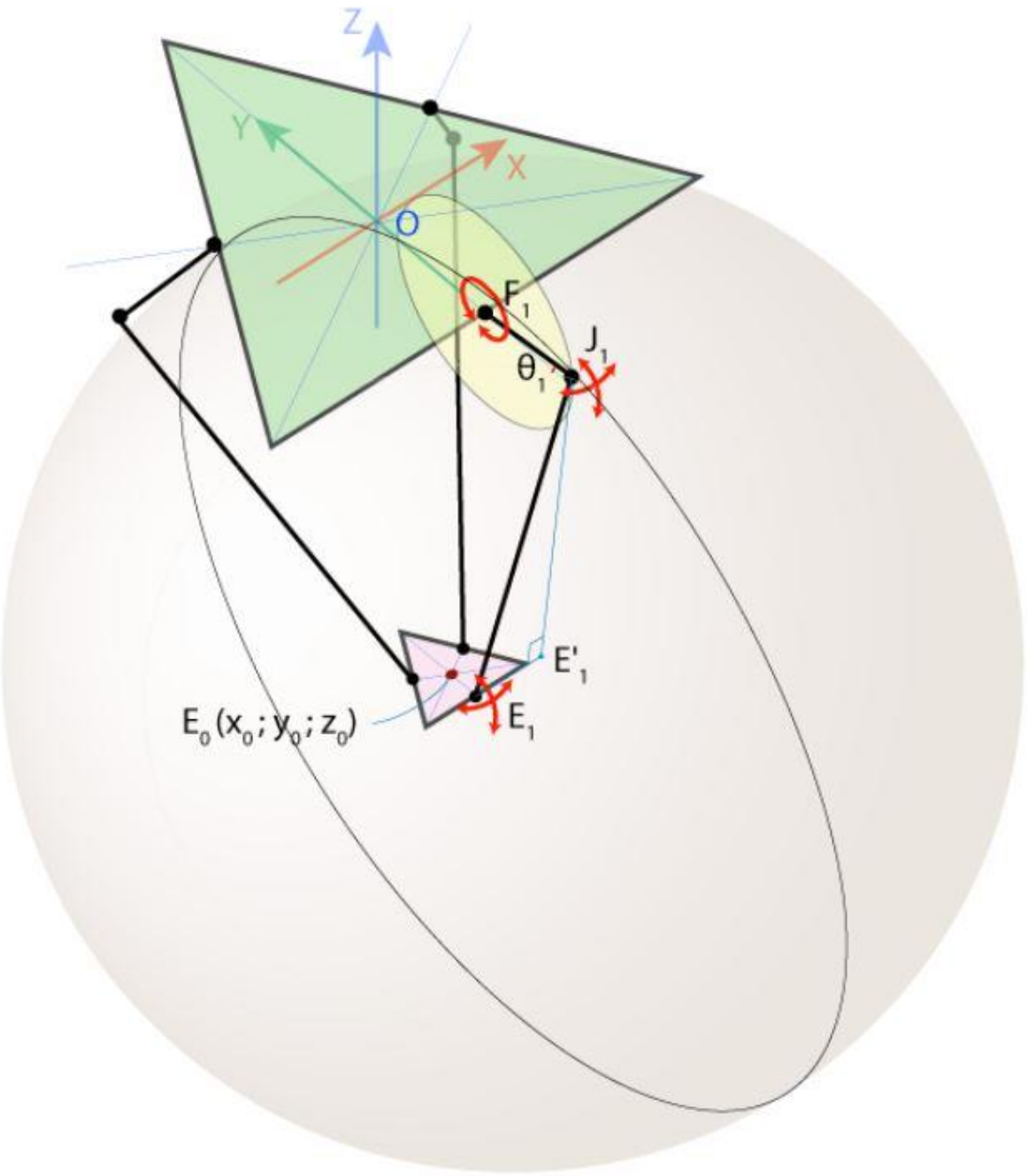

Figure 2-4: Kinematic Chain Movements (Msavatsky, 2009)

As shown in Figure 2-4, the calculation for one kinematic chain, the circle with the center at F_1 and the radius of F_1J_1 shows the path of moving link r_f around the center of F_1. This movement

is the first step toward figuring out the inverse kinematic equation. By placing the zero point of the Cartesian system at the center of the fixed platform, the circle will be defined at YZ plane.

The second step toward forming the inverse kinematic equation is moving forward on the kinematic chain to point E_1. The universal joint at the point E_1 provides a free movement of the link E_1J_1 around the center (E_1). This free movement around the center shapes a sphere with these specifications: the center at E_1 and radius of E_1J_1 or link r_e (Msavatsky, 2009).

Finding the equation of the point J_1 is the purpose of this model. Since we have the coordinate of the point J_1, the calculation of θ_1 -which is the motor angle- would be easy. To do so, the intersection of the circle with center F_1 and the sphere with center E_1 is needed. The sphere is shaped in three dimensions whereas the circle is just on YZ plane. The ultimate solution is to work on the image of the sphere on the YZ plane which would be a circle in two dimensional space. With a transformation of the center E_1 to E'_1 and drawing a circle with the center at E'_1 and radius of E'_1J_1, the image of sphere on YZ plane appears. At this point, the model of the robot path is extracted. Figuring out the equations for this path would result in finding the ultimate J_1 coordinate which is the main element of motor angle calculation (Msavatsky, 2009).

The third step mainly consists of looking at the whole system considering YZ plane. The view of looking at the whole system with respect to YZ plane is shown in Figure 2-5. In addition to this view, Figure 2-5 contains the mobile platform geometry which is necessary here because we are transferring the end point to find the image of the sphere on the YZ plane. This transformation is happening on the mobile platform which makes the mobile platform geometry important.

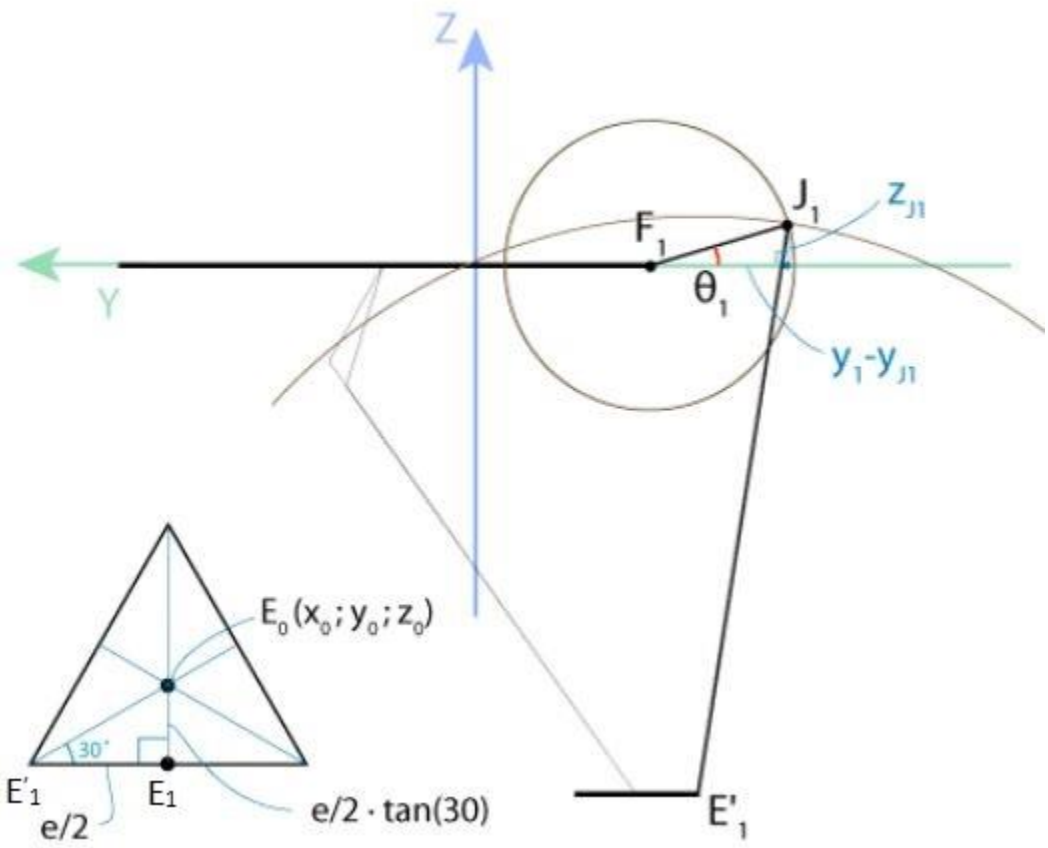

Figure 2-5: YZ Plane View (Msavatsky, 2009)

In order to do the calculation of the E'_1 position, a transformation is needed from our end point E_0 (end effector position) with the coordinates of x_0, y_0, and z_0 to the corner of the equilateral triangle.

As is defined on the top view of mobile platform in Figure 2-5, while transferring point E_0 to E'_1, a right triangle is shaped. The side of equilateral triangle (mobile platform) is e and as point E_1 is at the center of the side, the dimension of $E_1E'_1$ would be $e/2$. Now, we have three angles 90°, 60°, and 30°on the right triangle $E_0E_1E'_1$. Considering the E'_1 angle (30°), the following calculations show the measurement of E_0E_1.

$$\cos 30° = \frac{\frac{e}{2}}{hypotenuse} \quad \textit{(Equation 2-1)}$$

$$\sin 30° = \frac{E_0E_1}{hypotenuse} \quad \textit{(Equation 2-2)}$$

$$\frac{\sin 30°}{\cos 30°} = \tan 30° = \frac{\frac{E_0E_1}{hypotenuse}}{\frac{\frac{e}{2}}{haypotenuse}} = \frac{E_0E_1}{\frac{e}{2}} \quad \textit{(Equation 2-3)}$$

$$E_0E_1 = \frac{e}{2} \times \tan 30° = \frac{e}{2\sqrt{3}} \quad \textit{(Equation 2-4)}$$

The coordinate of E_0 is x_0, y_0, and z_0 and the coordinate of E_1 is x_0, $y_0 - \frac{e}{2\sqrt{3}}$, and z_0. As the coordinate of E'_1 is the same as the coordinate of E_1 with the shift in x to zero, we have the coordinate of E'_1 as 0, $y_0 - \frac{e}{2\sqrt{3}}$, and z_0. So, we have $E_1E'_1 = x_0$.

On the other side, on the fixed platform, the same calculations are proven in the same way as of the mobile platform. With the exact same calculation on the top equilateral triangle (fixed platform) with the side of f, the coordinate of point F_1 is 0, $-\frac{f}{2\sqrt{3}}$, and 0 (Msavatsky, 2009).

On the right triangle $J_1E'_1E_1$, applying Pythagorean Theorem, the following calculations are applied (Msavatsky, 2009).

$$(E_1J_1)^2 = (E_1E'_1)^2 + (E'_1J_1)^2 \quad \textit{(Equation 2-5)}$$

$$(\mathrm{E}_1'\mathrm{J}_1)^2 = (E_1J_1)^2 - (\mathrm{E}_1\mathrm{E}_1')^2 \qquad \textit{(Equation 2-6)}$$

$$\mathrm{E}_1'J_1 = \sqrt{E_1J_1 - \mathrm{E}_1\mathrm{E}_1'} = \sqrt{r_e^2 - x_0^2} \qquad \textit{(Equation 2-7)}$$

The point J_1 is the intersection of the two circles with the centers F_1 and E'_1 with the radiuses r_f and r_e respectively. Mathematical model of the two circles` equations considering the points` coordinate and previous calculations would generate the coordinate of point J_1 as the intersection point of the two circles. The output of the coordinate of the point J_1 and the geometry of the right triangle $F_1J_1Y_1$ will give us the angle θ_1.

The circle equation with the center at F_1 and the radius of r_f is (Msavatsky, 2009):

$$\left(y_{J1} - y_{F1}\right)^2 + \left(z_{J1} - z_{F1}\right)^2 = r_f^2 \qquad \textit{(Equation 2-8)}$$

Replacing the calculated points into the equation, we have (Msavatsky, 2009):

$$\left(y_{J1} + \frac{f}{2\sqrt{3}}\right)^2 + z_{J1}^2 = r_f^2 \qquad \textit{(Equation 2-9)}$$

The circle equation with the center at E'_1 and the radius of r_e is (Msavatsky, 2009):

$$\left(y_{J1} - y_{\mathrm{E}\prime 1}\right)^2 + \left(z_{J1} - z_{\mathrm{E}\prime 1}\right)^2 = r_f^2 \qquad \textit{(Equation 2-10)}$$

Replacing the calculated points into the equation, we have (Msavatsky, 2009):

$$\left(y_{J1} - y_0 + \frac{e}{2\sqrt{3}}\right)^2 + \left(z_{J1} - z_0\right)^2 = r_e^2 - x_0^2 \qquad \textit{(Equation 2-11)}$$

Combining the two circles equations, the final coordinate of point J_1 will be extracted. The elements for the J_1 position would be 0, y_{J1}, and z_{J1}.

Considering the right triangle $F_1J_1Y_1$, following calculations result in ultimate θ_1 (Msavatsky, 2009).

$$J_1Y_1 = z_{J1} \qquad \textit{(Equation 2-12)}$$

$$F_1Y_1 = y_{F1} - y_{J1} \qquad \textit{(Equation 2-13)}$$

$$\theta_1 = \tan^{-1}\frac{z_{J1}}{y_{F1} - y_{J1}} \qquad \textit{(Equation 2-14)}$$

For the θ_2 calculation, a 120° rotation of the coordinate system around the Z-axis is needed. This rotation is needed because the motors on the Delta robot are located at the center of the fixed platform sides. This means that with putting zero point at the center of the fixed platform, the motors have 120° differences with each other. Considering θ_1 as zero value, θ_2 and θ_3 would have 120° counterclockwise and clockwise differences respectively.

Using rotation matrix, for the 120° rotation counterclockwise (finding θ_2 coordinate), the following equations show the new coordinate system (Msavatsky, 2009).

$$x' = x\cos 120 + y\sin 120 \qquad \textit{(Equation 2-15)}$$

$$y' = -x \sin 120 + y \cos 120 \qquad \textit{(Equation 2-16)}$$

The Figure 2-6 shows the coordinate system rotation and the new coordinate system.

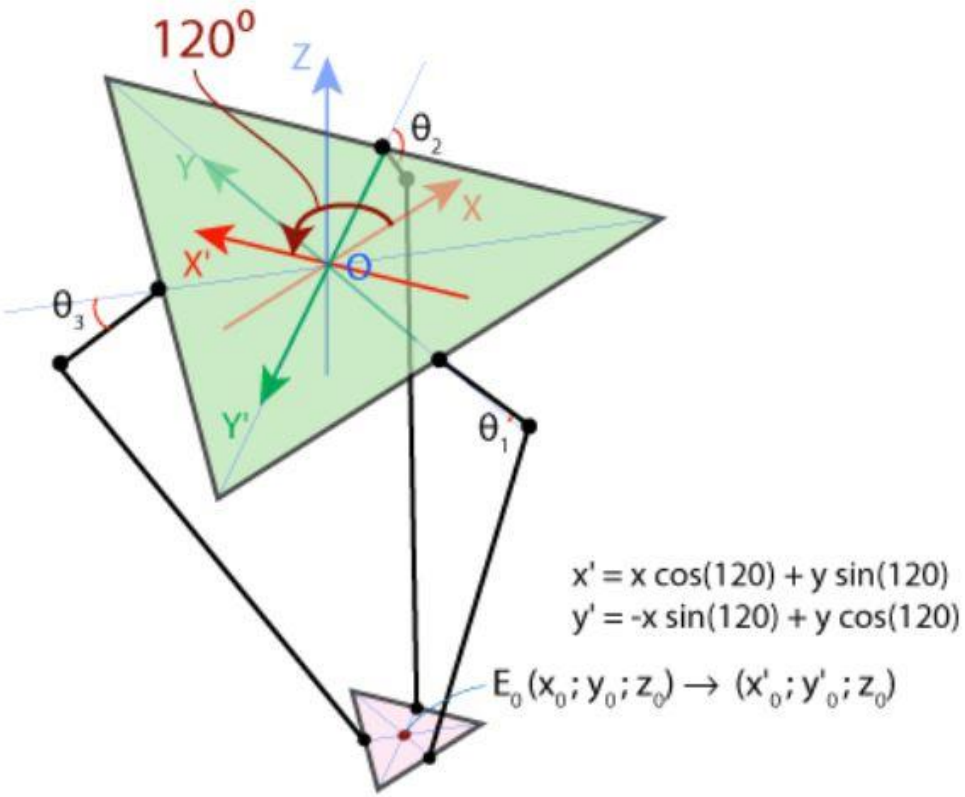

Figure 2-6: Coordinate Rotation (Msavatsky, 2009)

By repeating exactly the same calculations used for calculating θ_1 with respect to the new coordinate system, the θ_2 is gained as (Msavatsky, 2009):

$$\theta_2 = \tan^{-1} \frac{z'_{J1}}{y'_{F1} - y'_{J1}} \qquad \textit{(Equation 2-17)}$$

In order to do the calculation of θ_3, the rotation of coordinate system for 120° clockwise is needed. Using rotation matrix and the initial position of the coordinate system, the following equations show the new coordinate system.

$$x'' = x\cos 120 - y\sin 120 \qquad \textit{(Equation 2-18)}$$

$$y'' = x\sin 120 + y\cos 120 \qquad \textit{(Equation 2-19)}$$

Following the previous calculation, θ_3 would be:

$$\theta_3 = \tan^{-1}\frac{z''_{J1}}{y''_{F1} - y''_{J1}} \qquad \textit{(Equation 2-20)}$$

At this point, the three formulas for calculation of three angles enable us to find out about the exact motor angles base on the position of the end effector on the coordinate system. So, completing the calculations of the universe kinematics having the coordinates of end effectors, the joint angles are found as:

$$\theta_1 = \tan^{-1}\frac{z_{J1}}{y_{F1} - y_{J1}} \qquad \textit{(Equation 2-21)}$$

$$\theta_2 = \tan^{-1}\frac{z'_{J1}}{y'_{F1} - y'_{J1}} \qquad \textit{(Equation 2-22)}$$

$$\theta_3 = \tan^{-1}\frac{z''_{J1}}{y''_{F1} - y''_{J1}} \qquad \textit{(Equation 2-23)}$$

2.1.4 Forward Kinematic

For the inverse kinematics modeling and calculation, the final position of the end effector is known and the unknown variables are the motor angles θ_1, θ_2, and θ_3. The forward kinematic works in the opposite way. In the other words, the motor angles θ_1, θ_2, and θ_3 are considered as known variables while the aim of the calculations is to figure out the unknown variable which is the position of the end effector. The position consists of three elements x, y, and z. The three angles and the position of the end effector are shown in the Figure 2-7 (Msavatsky, 2009).

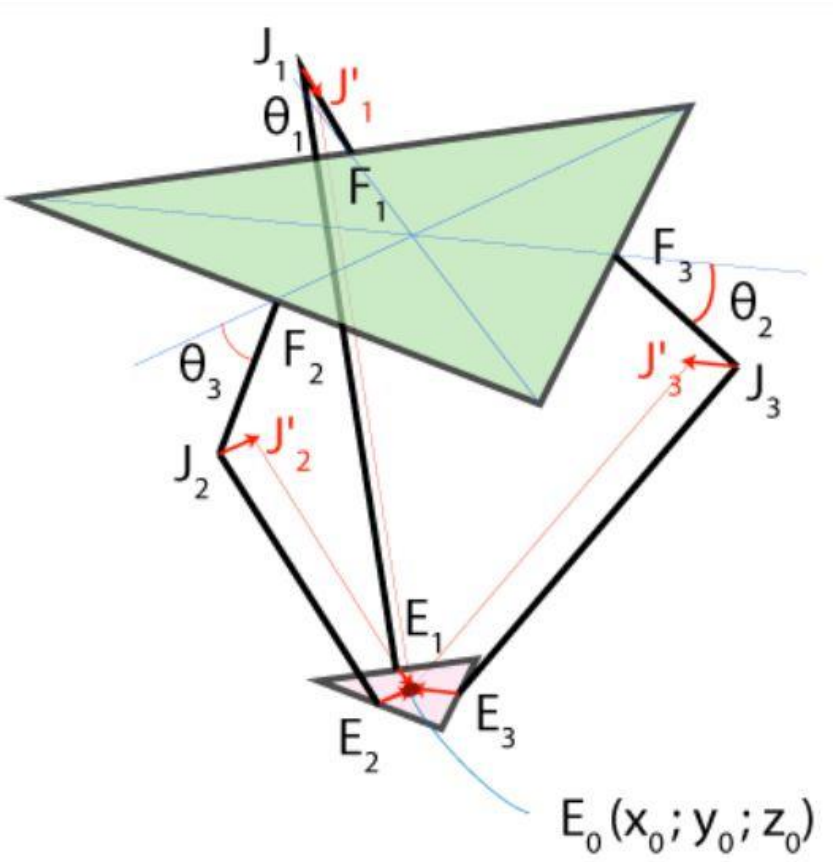

Figure 2-7: Joints Configuration (Msavatsky, 2009)

As the three revolute joints at the contact points of kinematic chains and the fixed platform can only transmit the motor movements, the three universal joints which indicated as J_1, J_2, and J_3 on the Figure 2-7 would play the role of transmitting rotations. In the other words, with the calculation of J_1, J_2, and J_3 positions on the coordinate system and a transmission from them to the mobile platform, the final position of the end effector would be extracted.

The movements of the three links (J_1E_1, J_2E_2, and J_3E_3) around the three centers (J_1, J_2, and J_3) will shape three spheres. Similar to the invers kinematic calculations, figuring out the equations for the three spheres should determine the final position of the end effector. However, we encounter one important issue here. The issue is that the three spheres do not have precisely the same intersection points which leave the position of the end effector unknown.

In order to solve this problem, the transitions of the three sphere centers are needed. As it is shown on the Figure 2-7, the three joint positions (the three centers) are moved to the points J'_1, J'_2, and J'_3 using the transitions vectors E_1E_0, E_2E_0, and E_3E_0. These transitions would help to shape three spheres with the intersection at the position of the end effector. The important point is considering these transitions in doing the final calculation (Msavatsky, 2009).

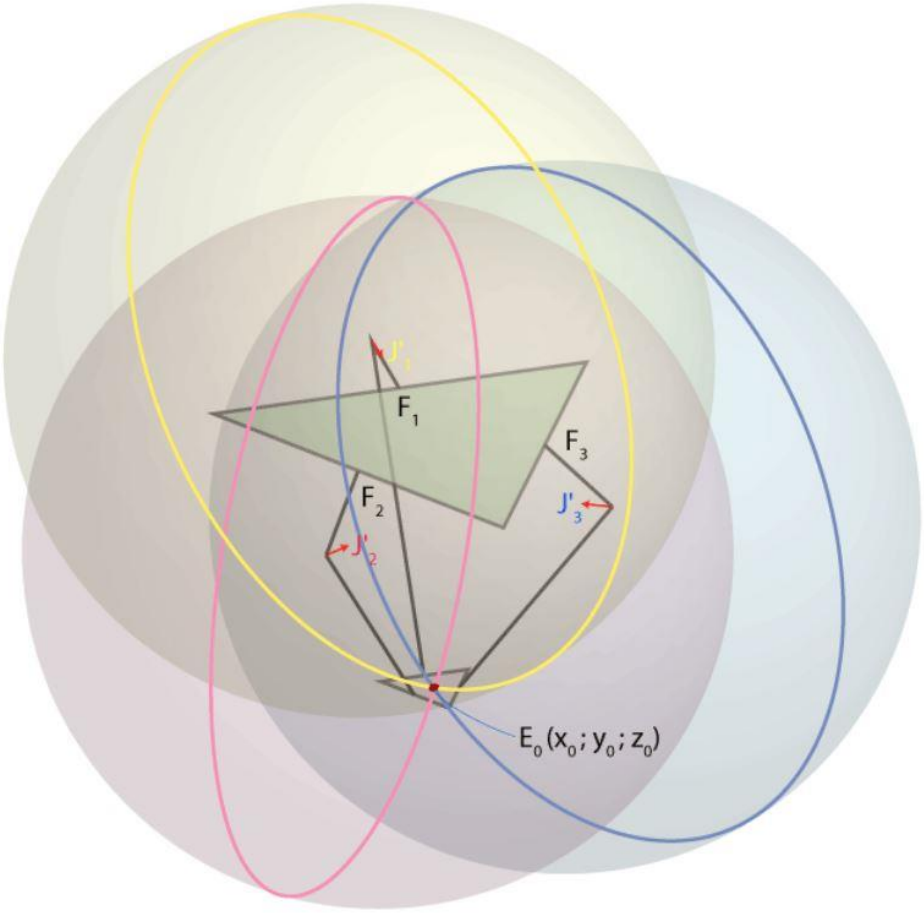

Figure 2-8: Kinematic Chains Movement (Msavatsky, 2009)

As the Figure 2-8 displays, the three spheres with the centers at J'_1, J'_2, and J'_3 and the radiuses of J'_1E_0, J'_2E_0, and J'_3E_0 are shaped. The length of the radiuses are equal to the length of the bottom link which is called r_e. The circles at the center of these three spheres have intersection at point E_0 which is the end effector position. In other words, the intersection of the three circles should be found in order to figure out the position of the end effector. For this purpose, the equation of the circles with the initial coordinate of contributed points are needed. This means that the initial coordinate of J'_1, J'_2, and J'_3 as the centers of the circles and the bottom link length r_e are the main elements to make the set of circles` equations. The output of the set of the equations would be the desired intersection point (Msavatsky, 2009).

The first step in this process is finding the coordinate of the centers J'_1, J'_2, and J'_3. The Figure 2-9 shows the geometry of the fixed platform and the position of points and links from the top view. The important point is the coordinate system position which is placed in a way that zero of the system is at the center of the equilateral triangle. It means the point zero on the z-axis is on the fixed platform (Msavatsky, 2009).

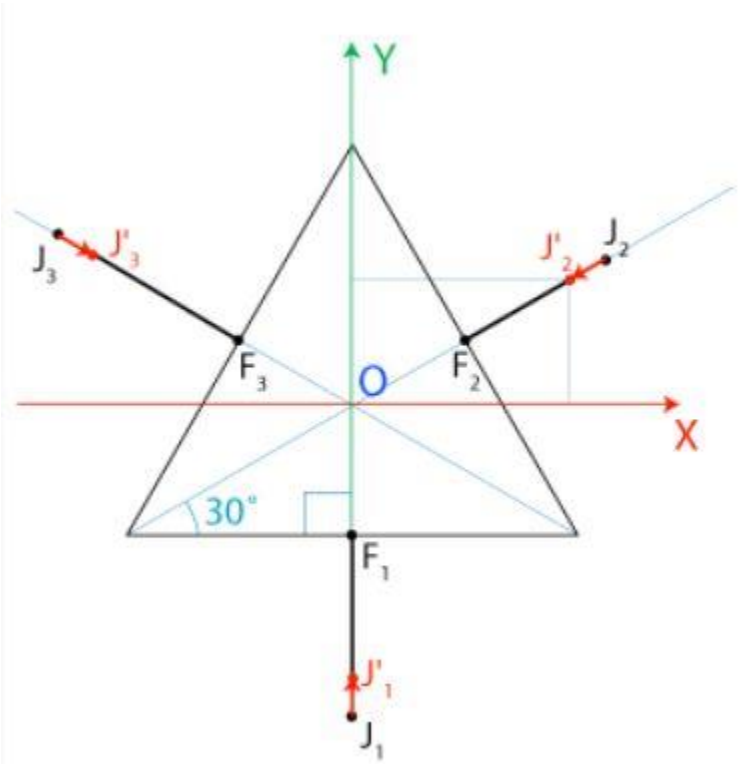

Figure 2-9: Fixed Platform Geometry (Msavatsky, 2009)

Considering the geometry of the top view of the robot which is shown on the Figure 2-9, the lines between the center and the motor places would have the following equations (Msavatsky, 2009).

$$OF_1 = OF_2 = OF_3 = \frac{f}{2} \times \tan 30 = \frac{f}{2\sqrt{3}} \qquad \textit{(Equation 2-24)}$$

The f would be the side of the equilateral triangle (the fixed platform).

The images of the sphere centers transition from points J to points J′ (J_1 to J'_1, J_2 to J'_2, and J_3 to J'_3) on the mobile platform (equilateral triangle) could be calculated as in the following.

$$J_1J'_1 = J_2J'_2 = J_3J'_3 = \frac{e}{2} \times \tan 30° = \frac{e}{2\sqrt{3}} \quad \textit{(Equation 2-25)}$$

The e would be the side on mobile platform.

The length of the motor positions to the top universal joints are as the following.

$$F_1J_1 = r_f \times \cos\theta_1 \quad \textit{(Equation 2-26)}$$

$$F_2J_2 = r_f \times \cos\theta_2 \quad \textit{(Equation 2-27)}$$

$$F_3J_3 = r_f \times \cos\theta_3 \quad \textit{(Equation 2-28)}$$

The second step is the calculation of the J'_1, J'_2, and J'_3 positions on the coordinate system. Considering the positions of the fixed platform, links, joints, and the mobile platform on the coordinate system and the above calculations, the x, y, and z components of J'_1, J'_2, and J'_3 are as follows (Msavatsky, 2009):

$$J'_1: 0, \frac{-(f-e)}{2\sqrt{3}} - \left(r_f \times \cos\theta_1\right), -r_f \sin\theta_1 \quad \textit{(Equation 2-29)}$$

$$J_2'\colon\left(\frac{(f-e)}{2\sqrt{3}}+\left(r_f\times\cos\theta_2\right)\right)\times\cos 30\,,\left(\frac{(f-e)}{2\sqrt{3}}+\left(r_f\times\cos\theta_2\right)\right)\times\sin 30\,,-r_f\sin\theta_2 \qquad \textit{(Equation 2-30)}$$

$$J_3'\colon\left(\frac{(f-e)}{2\sqrt{3}}+\left(r_f\times\cos\theta_3\right)\right)\times\cos 30\,,\left(\frac{(f-e)}{2\sqrt{3}}+\left(r_f\times\cos\theta_3\right)\right)\times\sin 30\,,-r_f\sin\theta_3 \qquad \textit{(Equation 2-31)}$$

Considering all the above calculations, the set of the three circles equations would be as (Msavatsky, 2009):

$$x^2+(y-y_1)^2+(z-z_1)^2=r_e^2 \qquad \textit{(Equation 2-32)}$$

$$(x-x_2)^2+(y-y_2)^2+(z-z_2)^2=r_e^2 \qquad \textit{(Equation 2-33)}$$

$$(x-x_3)^2+(y-y_3)^2+(z-z_3)^2=r_e^2 \qquad \textit{(Equation 2-34)}$$

Expanding the (Equation 2-32), we have (Msavatsky, 2009):

$$x^2+y^2+z^2-2y_1y-2z_1z=r_e^2-y_1^2-z_1^2 \qquad \textit{(Equation 2-35)}$$

Expanding the (Equation 2-33), we have (Msavatsky, 2009):

$$x^2 + y^2 + z^2 - 2x_2x - 2y_2y - 2z_2z = r_e^2 - x_2^2 - y_2^2 - z_2^2 \qquad \textit{(Equation 2-36)}$$

Expanding the (Equation 2-34), we have (Msavatsky, 2009):

$$x^2 + y^2 + z^2 - 2x_3x - 2y_3y - 2z_3z = r_e^2 - x_3^2 - y_3^2 - z_3^2 \qquad \textit{(Equation 2-37)}$$

For making the calculation easier, the following equation is considered (Msavatsky, 2009).

$$w_i = x_i^2 + y_i^2 + z_i^2 \qquad \textit{(Equation 2-38)}$$

Subtracting (Equation 2-36) from the (Equation 2-35) and replacing (Equation 2-38)in the output, we have (Msavatsky, 2009):

$$x_2x + (y_1 - y_2)y + (z_1 - z_2)z = \frac{(w_1 - w_2)}{2} \qquad \textit{(Equation 2-39)}$$

Subtracting (Equation 2-37) from (Equation 2-35) in the same way, we have (Msavatsky, 2009):

$$x_3x + (y_1 - y_3)y + (z_1 - z_3)z = \frac{(w_1 - w_3)}{2} \qquad \textit{(Equation 2-40)}$$

By subtracting (Equation 2-37) from the (Equation 2-36), we have (Msavatsky, 2009):

$$(x_2 - x_3)x + (y_2 - y_3)y + (z_2 - z_3)z = \frac{(w_2 - w_3)}{2} \qquad \textit{(Equation 2-41)}$$

The following equations are extracted from (Equation 2-39) and (Equation 2-40) (Msavatsky, 2009).

$$x = a_1 z + b_1 \qquad \textit{(Equation 2-42)}$$

$$y = a_2 z + b_2 \qquad \textit{(Equation 2-43)}$$

Where a_1, b_1, a_2, and b_2 would be (Msavatsky, 2009):

$$a_1 = \frac{1}{d}\left[\left((z_2 - z_1) \times (y_3 - y_1)\right) - \left((z_3 - z_1) - (y_2 - y_1)\right)\right] \qquad \textit{(Equation 2-44)}$$

$$a_2 = \frac{-1}{d}\left[\left((z_2 - z_1)x_3\right) - \left((z_3 - z_1)x_2\right)\right] \qquad \textit{(Equation 2-45)}$$

$$b_1 = \frac{-1}{2d}\left[\left((w_2 - w_1) \times (y_3 - y_1)\right) - \left((w_3 - w_1) - (y_2 - y_1)\right)\right] \qquad \textit{(Equation 2-46)}$$

$$b_2 = \frac{1}{2d}\left[\left((w_2 - w_1)x_3\right) - \left((w_3 - w_1)x_2\right)\right] \qquad \textit{(Equation 2-47)}$$

Also, the d in the above equations would be (Msavatsky, 2009):

$$d = [(y_2 - y_1)x_3 - (y_3 - y_1)x_2] \qquad \textit{(Equation 2-48)}$$

With substituting equations (Equation 2-42) and (Equation 2-43) in (Equation 2-35) we have (Msavatsky, 2009):

$$(a_1^2 + a_2^2 + 1)z^2 + 2(a_1 + a_2(b_2 - y_1) - z_1)z + (b_1^2 + (b_2 - y_1)^2 + z_1^2 - r_e^2) = 0 \quad \textit{(Equation 2-49)}$$

Solving this quadric equation will give the answers for the z at the output. The smallest negative number would be the desirable answer. The final answer is the z component of the end effector position (z_0).

Substituting z_0 in equations (Equation 2-42) and (Equation 2-43) will give x_0 and y_0 at the output. The final coordinate of the end effector position would be x_0, y_0, and z_0.

2.2 Communication

2.2.1 Modbus

The Modbus protocol is a master-slave/ client-server base protocol which was developed in 1979 by Modicon. The Modbus protocol is mostly used in industry. That is why the Modbus protocol is categorized as an industrial communication standard. It is an open protocol which is mostly applied to transfer discrete and analog I/O information and register data between industrial control and monitoring devices (Acromag, 2005).

To have a simple interpretation of the client-server method for transferring data, we can consider that one device would be the master and responsible for transaction initiation in a network. The other devices in the network would just respond and provide data which is requested by the master. Simply put, the process involves requesting data from a slave device (valve, I/O transducer, network drive, or other measuring devices) or sending an instruction to the same slave devise. Sending query (the request from mater to slave/s) can address individual slave or can propagate

through the whole network. Slaves return a response to all the queries addressed to them individually. However, they do not answer to broadcast queries (Acromag, 2005) (Moody, J. O., Sánchez- Alonso, R., Yun, C., González- Barbosa, J., & Reyes- Morales, G., 2015).

A slave address, a function code, required data, and an error checking field are the forming formative components of the query. The response from the slave involves the confirmation of taken action, returned data, and an error checking field. The master would send a query with the specific slave address to the specific slave device. The query includes a function code and required data, for example asking for the end effector condition. If no error occurs, the slave`s data response contains the requested data. If an error occurs in the query received, or if the slave is unable to perform the action requested, the slave will return an exception message as its response (Acromag, 2005).

2.2.2 Modbus TCP/IP

Modbus protocol is a worldwide standard industrial protocol. In order to make the Modbus protocol more user friendly, more compatible to other networks and communications and add more features to it, the Modbus TCP/IP was developed in 1999 (Acromag, 2005).

The IEEE 802.3 Ethernet is an office network protocol that has gained universal worldwide acceptance and became popular due to its capability for data transactions. It is also an open standard that is supported by many manufacturers and its infrastructure is widely available and largely installed (Acromag, 2005).

The combination of IEEE 802.3 and Modbus has generated a powerful protocol called Modbus TCP/IP in which the positive capabilities of two protocols gathered together. This standard uses the client-server method and takes into account the structure bed of IEEE 802.3 to shape the query frames. Compatibility to the installed Ethernet infrastructure of cables, connectors,

network interface cards, hubs, and switches is the major positive point of the new protocol (Acromag, 2005) (RTA, n.d.).

In order to profoundly explain and create better understanding of the frame structure of Modbus TCP/IP, the OSI general model of frame structure will be examined first. In the next step, a thorough study on the Modbus TCP/IP frame structure will be conducted that leads to fundamental understanding of this networking method.

2.2.3 OSI Network Model

The Open System Interconnect (OSI) model was developed by the International Standards Organization in 1983. The Open System Interconnect (OSI) adopted as a common reference for the development of data communication standards. The general model of the OSI model is being used as the foundation of the structure of many communication protocols such as Modbus TCP/IP, HTTP, HTTPS, Ethernet, and a lot of other communication protocols (Acromag, 2005).

The OSI communication structure model consists of seven layers. Each layer in this concept is responsible for a specific role in the whole communication movie. Figure 2-10 illustrates the position of each layer based on the priority on making the communications.

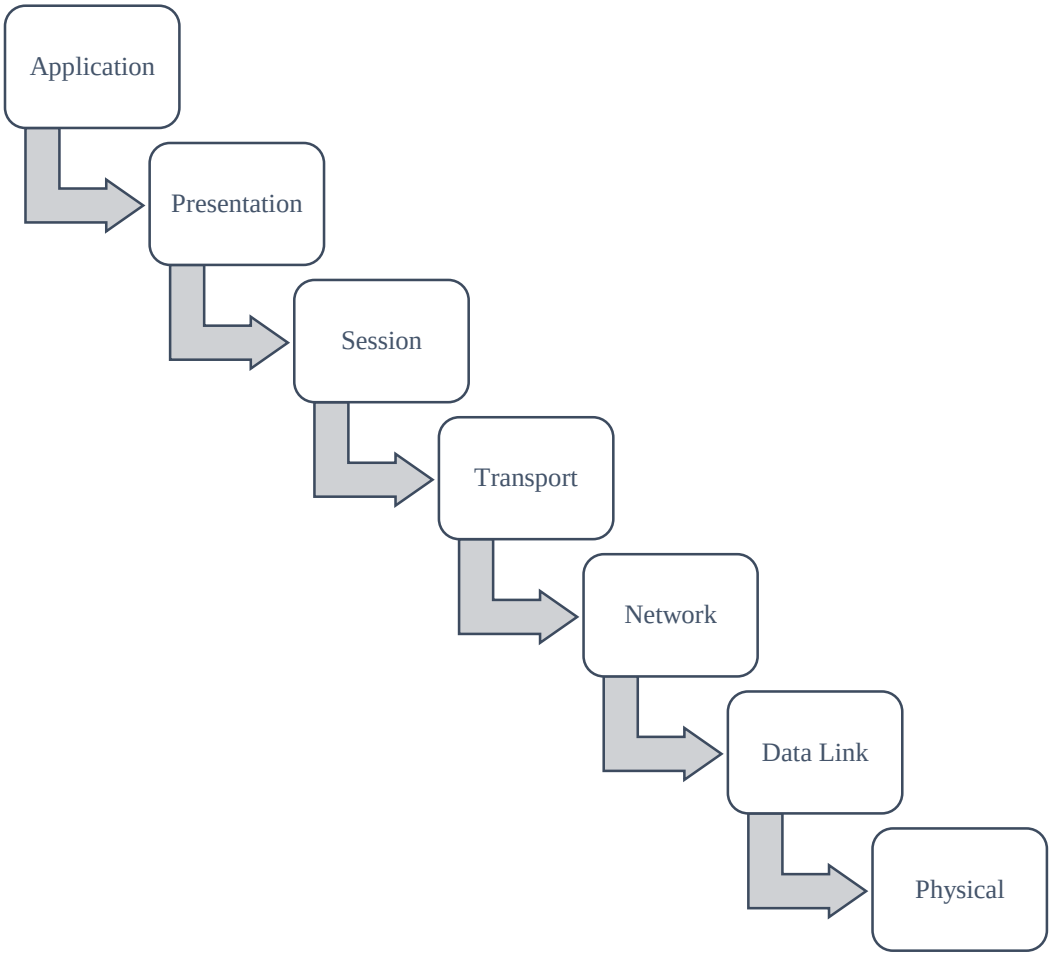

Figure 2-10: OSI Model

At the bottom place of the OSI model is located the Physical layer. The Physical layer defines the electrical, mechanical, functional, and procedural attributes used to access and send a binary data stream over a physical medium which can be RJ-45 connector or CAT5 cable (Acromag, 2005) (RTA, n.d.).

Being located above the Physical layer, the Data Link layer is responsible for ensuring reliable delivery at the lowest levels, including data frame, error detection and correction, sequence control and flow control. The protocols like Ethernet (IEEE 802.2) and MAC are defined at this level (Acromag, 2005) (RTA, n.d.).

The Network layer, placed on top of the Data Link layer and the Physical Layer, provides controls routing, prioritization, network setup, release of connections, and flow control. The main responsibilities of this layer involve establishing and maintaining connections over a network and providing addresses, routing, and delivery packets to hosts. The protocols such as IP, PPP, and IPX are all offered at this level (Acromag, 2005) (RTA, n.d.).

Next comes the Transport layer. The Transport layer is responsible for sequencing of application data, controlling start/end of transmission, providing error detection, data correction, end to end recovery, and clearing the communication. In other words, providing flow control of data between networks is the main responsibility of the Transport layer. The TCP and UDP protocols are both defined at this level (Acromag, 2005) (RTA, n.d.).

The Session layer is places above the Transport layer. The connections between applications and networks and establishing and managing sessions are all implemented in the Session layer in OSI model. Dialing control and synchronization of session connections are occurring in this layer. Windows WinSock socket API is considered as one of the most popular Session layer managers (Acromag, 2005) (RTA, n.d.).

The data compression and encryption are defined as the responsibilities of the Presentation layer. The layer offers the representation format of data, coding type and used characters. This layer performs data and protocol negotiation and conversion to ensure that data may be exchanged between hosts and transportable across the network (Acromag, 2005) (RTA, n.d.).

The Application layer is considered as the last layer in OSI model. This layer is being used by applications to prepare and interpret data for use by other layers. This layer provides the application interface to the network. The important protocols which are defined at this level include HTTP, FTP, SMTP, POP3, CIP, and SNMP (Acromag, 2005) (RTA, n.d.).

As explained above, the OSI model of communication makes a data frame that consists of different layers. In each layer, it defines specific role for communication, error detection, and so forth. The data frame will be transferred to the physical layer which is usually called communication channel and transfer to the destination. To set an example, consider the Ethernet. The Ethernet provides the communication bed in layer one and two which are Physical and Data Link layers. The TCP/IP is covering layers three and four, Network and Session. The applications using TCP/IP standard follow the same way of communication at layers three and four. At higher levels such as the Application layer, the connection will be made between software which share the same Application protocol.

For the Modbus TCP/IP, a reduction on OSI model results in a five- layer communication standard. As the Figure 2-11 shows, the structure of Modbus TCP/IP model has five layers. The layers Session, Presentation, and Application are combined into one layer called "application" with almost all capability of the three layers. As previously mentioned, the applications, software, or devices which are intended to build a connection considering the Modbus TCP/IP protocol should share the same Application layer. Using the same Application layer helps devices to code and decode the data they received from the network. Adding TCP/IP to this protocol made it capable of taking advantage of the compatibility between the applications which are using this standard. Thus, using Modbus TCP/IP protocol makes using an industrial protocol standard possible while the users benefit from the capability of TCP/IP which is the foundation for the World Wide Web (Acromag, 2005).

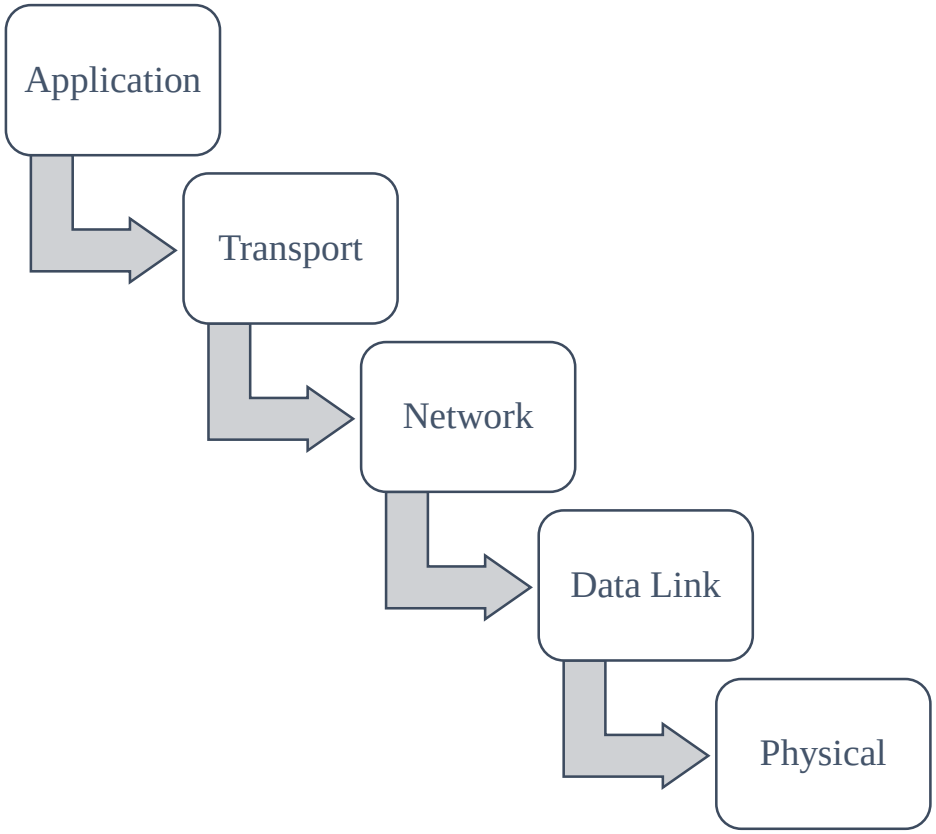

Figure 2-11: Modbus TCP/IP Model

Considering the TCP/IP protocols, as they are introduced in Network and Transport layers, the function of TCP (Transport layer) is to ensure that all packets of data are received correctly, while IP (Network layer) has the responsibility to make sure that messages are correctly addressed and routed. The important point worth considering here is that the data is made in the Application layer and TCP and IP do not make any changes in the original data. They are only playing the role of communication protocols (Acromag, 2005) (RTA, n.d.).

The application protocols which are usually defined in the Application layer are the protocols carrying the responsibility for organizing and interpreting data. The structure of the data frame is shaped and coded in this layer at the sender `s side and is then sent to the destination. At

the receiver `s side, the frame would be decoded and the original data is extracted out of it (Acromag, 2005) (RTA, n.d.).

The following table shows the protocol stack, or simply put, Modbus TCP/IP communication layers. As it was mentioned above, the protocol stack of Modbus TCP/IP is following the standard OSI communication model, but with the combination of the layers of Session, Presentation, and Application in one layer called “Application”. This combination makes the protocol stack a five-layer structure.

5	Application	Specifies how an application uses a network
4	Transport	Specifies how to ensure reliable data transport
3	Network/Internet	Specifies packet format and routing
2	Host-to-Network	Specifies frame organization and transmittal
1	Physical	Specifies the basic network hardware

Table 2-1: Modbus TCP/IP (Acromag, 2005)

The same protocol stack exists at the both sides for all the applications, software, network, and devices communicating together. Both sender and receiver have the same structure for the communication. The data (request/query) is generated at the Application layer at the sender. Then it moves through each layer down and a header is added at each and every step. In other words, each layer adds its own identifier to the data. In this way, when the data gets to the receiver, each layer identifies and decodes the relevant header. Base on the table, at level 2, the data is encapsulated in a frame with each layer header and thus, is ready to proceed to level one (Physical layer) to be sent into the channel. Conversely, this header information is removed by the corresponding layer at the receiver. In this way, the headers are essentially peeled off as the data packet moves up the receiving stack to the receiver Application. The following table illustrates the Modbus TCP/IP communication stack and protocols which are used in every level.

Modbus TCP/IP Communication Stack			
#	**Model**	**Protocols**	**References**
7	Application	Modbus	
6	Presentation		
5	Session		
4	Transport	TCP	
3	Network	IP, ARP, RARP	
2	Data Link	Ethernet, CSMA/CD, Mac	IEEE 802.3 Ethernet
1	Physical	Ethernet Physical Layer	

Table 2-2: Modbus TCP/IP Communication Stack (Acromag, 2005)

For better understanding of the aforementioned explanations, Figure 2-12 displays the great journey from layer five (Application layer) to layer two (Data Link layer). As it is shown in the Figure 2-12, at each step, the layer adds its own header to the data by which it can be distinguished at the receiver side (Acromag, 2005) (RTA, n.d.).

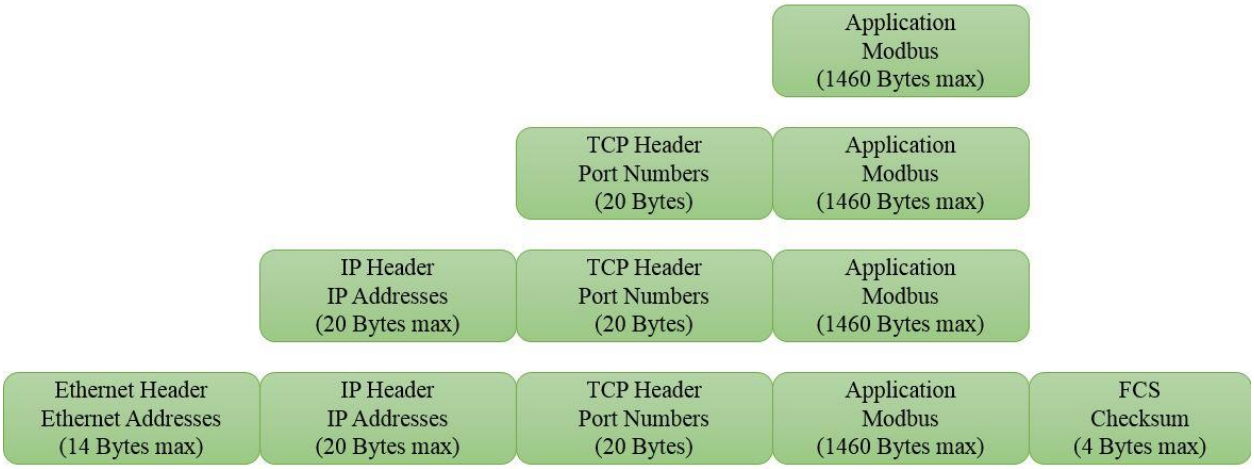

Figure 2-12: Modbus TCP/IP Frame Structure

2.3 Virtual Reality

Virtual reality is a fully immersive computer simulated environment that gives the user the feeling of being in that environment. A lot of video games have already developed the technology to put the user in an interactive world such as a driver `s seats in a car, a warrior in a first person shooter game, or even in a town that the gamers build themselves. When the users are able to freely

move within the virtual environment and interact with the objects in it, the user `s brain can truly perceive that the virtual world as real.

Virtual reality is considered to have begun in the 1950's but early elements of it can be traced back to the 1860's and long before the development of digital technology (Virtual Reality Society, 2016). The main concept of virtual reality is providing a bed for human brain and convince it to accept the virtual situation as real. With the emergence of power computers, graphic cards, coding languages, and 3D design software, the invention of virtual reality was upgraded to next level that has dramatically altered the world games and training in such virtual environments.

The main capability of the virtual environments and game engines is the implementation of physics engines inside them. It means that the rules and regulations of real world which come from rules of physics can be simulated in this environment. It has provided the capability of making the virtual environments based on the and with the real behavior of the objects in the real world. With respect to such great capability, the implementation of a game or a laboratory in virtual environment can result in achieving compelling output for human brains that has the power to convince it to believe the virtual environment as the real one (Moody, J. O., Sánchez- Alonso, R., Yun, C., González- Barbosa, J., & Reyes- Morales, G., 2015).

In addition to gaming, the virtual reality has a great deal of other practical purposes with respect to its capability of simulating real behavior of objects. As a result of such capability, virtual reality has been applied for practices and purposes other than gaming namely training simulators for soldiers, pilots, doctors, and engineers. Big companies are looking for interactive, safe, and inexpensive training programs. And since virtual reality environment offers options including moving within the virtual plants, making operational decisions, and investigating processes at a glance, it is highly important and popular for such big companies looking for it (Maurizio

Rovaglio, Tobias Scheele, 2011) (Etienne van Wyk, 2014).Using virtual reality and gaming to implement an environment for educational and training purposes is mainly what the concept of virtual laboratory explains. The virtual laboratories in general and virtual robot laboratories, in the case of this thesis, offer an environment in which the implemented robots simulate the exact behavior of real robots. In this way, the students and trainees can manipulate the virtual robots by monitoring the robots behavior and observing the responses sent from the robot to the users as a result of applying different structures, codes, and scenarios (Moody, J. O., Sánchez- Alonso, R., Yun, C., González- Barbosa, J., & Reyes- Morales, G., 2015).

The main reason for implementation of virtual laboratories is providing more resources for students in academic areas and trainees in different industries. Since the main obstacle facing users in the actual training environments is the lack of sufficient resources considering high costs of equipment, applying the method of the implementation of virtual robotic laboratories in place of actual ones can provide more opportunities for students and trainees to work and learn. The other advantage of using such robotic laboratories is that immense amounts of financial resources can be saved for companies. There are still more upsides to the application of virtual laboratories instead of the actual robotic laboratories. To illustrate, consider the pressing issues of robots vulnerability to damages and potential safety issues, all of which will be solved by using virtual robot laboratories. Programming wrong codes and sequences is inevitable during training session since participants are mainly untrained or novice learners. Students and trainees always make some mistakes at different levels. Sometimes, some of these mistakes can cause damages to robots and themselves. The high expenses paid on the maintenance of robots due to damages and irreparable human hurts are crucial challenges facing users in actual robot laboratories. However, taking advantage of the technology of virtual robot laboratories can eliminate all these expenses by

simulated hazardous situations in a safe, highly visual, and interactive way (Etienne van Wyk, 2014) (Maurizio Rovaglio, Tobias Scheele, 2011).

Considering the aforementioned issues with conducting training sessions in actual robotic laboratories along with the specific capabilities of the virtual reality environments have developed the idea and trend of creating virtual worlds with the application of running training sessions in both academic environments and industrial preparation practices. Such a trend has led into generating the concept of serious gaming which is defined as using game engines, with the capability of implementation of the rules of physics in them, in the real world environments.

2.3.1 Design

As it mentioned in the previous section, the availability of tools with high performance such as the platforms for virtual development, physics engines, computer processors and graphics cards has provided appropriate bed for the development of virtual laboratories. The implementation of virtual laboratories requires several steps from designing elements to programming. These steps include designing each element, exporting designed elements to virtual environment, and programing to achieve the real behavior of objects. Figure 2-13 depicts the procedure and algorithm of objects behavior in the virtual environment (Moody, J. O., Sánchez-Alonso, R., Yun, C., González- Barbosa, J., & Reyes- Morales, G., 2015).

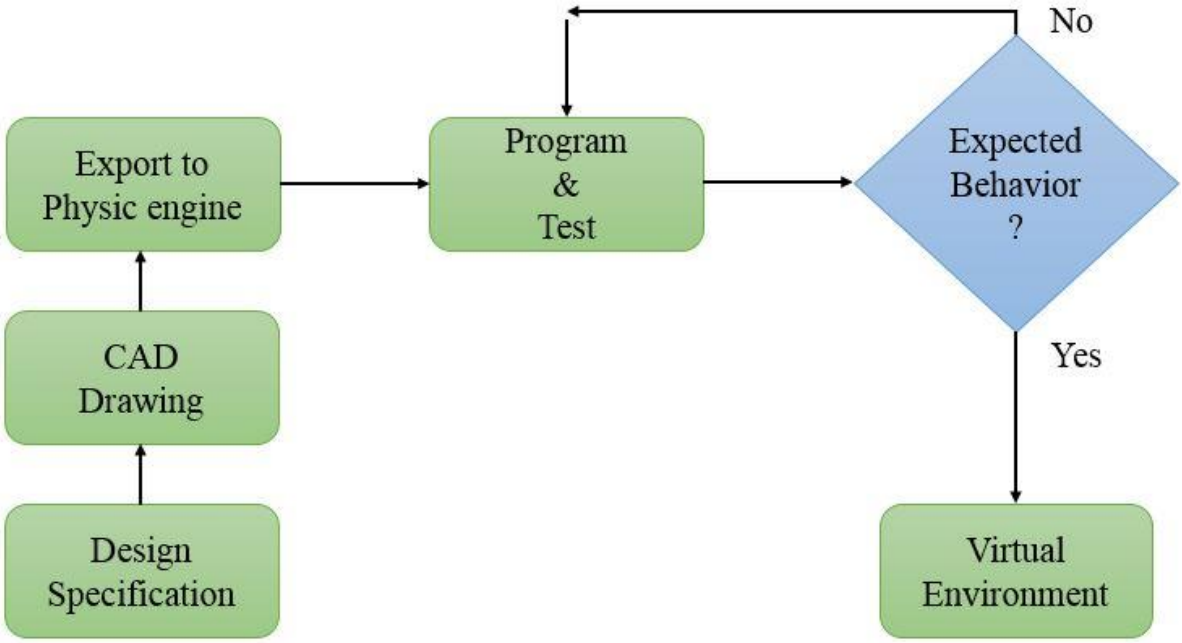

Figure 2-13: The Whole System Concept (Moody, J. O., Sánchez- Alonso, R., Yun, C., González- Barbosa, J., & Reyes- Morales, G., 2015)

The first step is CAD drawing. In order to draw the elements, there are numerous software. The exact geometry of objects are the critical points at this step as it is shown in the Figure 2-13. The ability of moving around freely based on the design is the key point in the first step in order to simulate behavior with higher precision of similarity to the real objects.

There are also a large number of game engines and software which offer physics engines. The Unity software has been used, in the case of this thesis, to simulate the behavior of rigid bodies such as gravity, collision, detection, mass and center of mass, angular velocities, acceleration, forces, and torques in the virtual Delta robot.

The reason why the author has chosen to work with Unity refers to the capabilities of this software. Some of the capabilities of Unity are explained in the following (Unity, n.d.):

- Scripting with C#, JavaScript or Boo (.NET-based).
- Action-Packed Physics (built-in NVIDIA PhysX 3™ and Box2D physics engines)

- Life-Like Animation.
- Supporting numerous platforms such as Windows, Mac, Linux/Steam OS, iOS, Android, Windows Phone 8, Windows Store, BlackBerry 10, Tizen, Xbox 360, Xbox One, and PlayStation 3.
- Optimized Graphics.
- 64-bit Editor.
- Inverse Kinematics (use IK rigs to move your character to a pre-determined point on an object in a natural way – position feet on the ground or hands on the edge of a wall).
- Sync Layers and Additional Curves (attach animation curves to animation clips to ensure proper encapsulation of the game code).
- Static batching (create geometry batches for static meshes at build-time so the CPU does not spend time recreating the same batches).

The virtual Delta robot studied in the case of this thesis has been developed by Dr. Ortega-Moody using all of the mentioned software and extensive mechanical and electrical design knowledge. Figure 2-14 illustrates the virtual robot in Unity virtual environment.

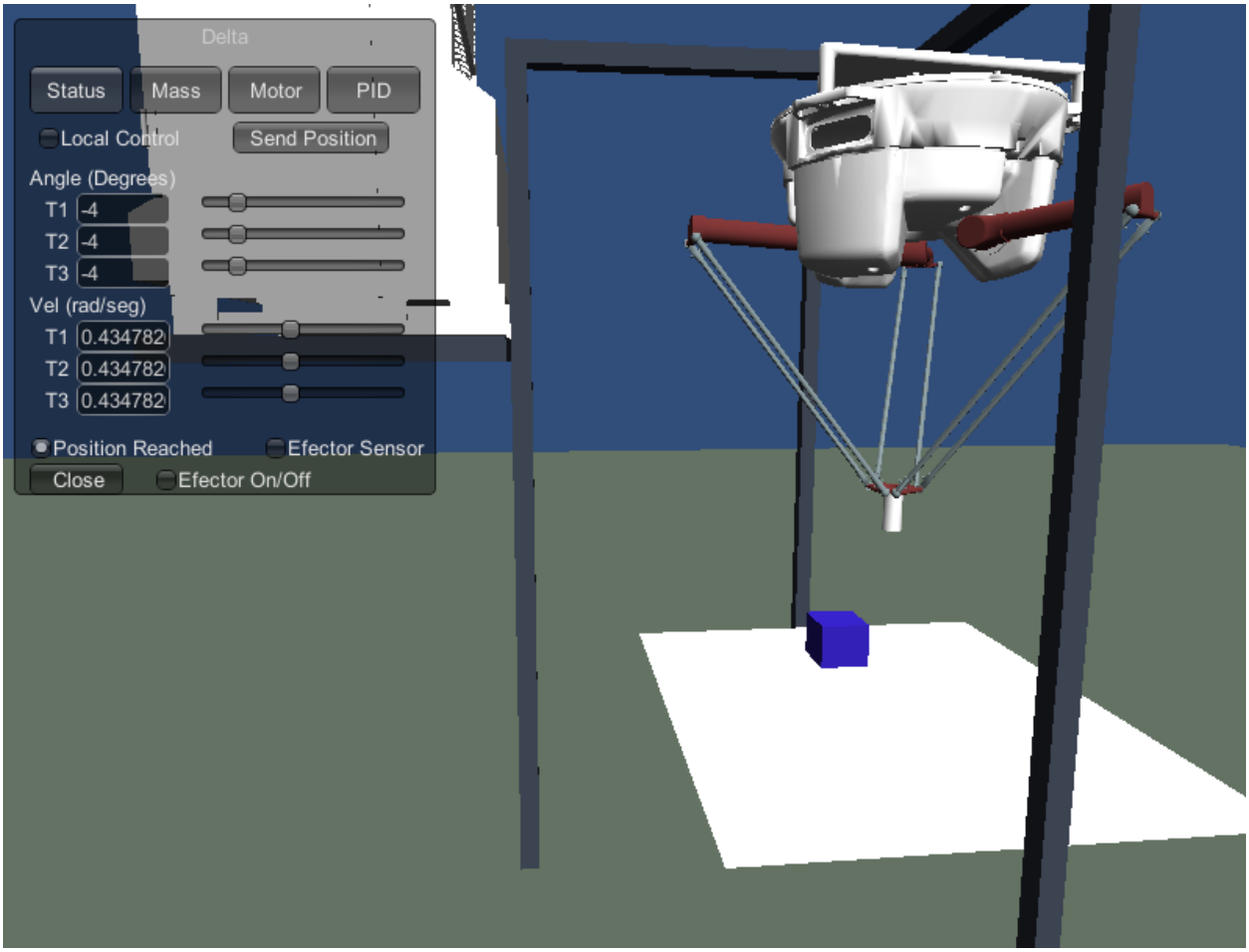

Figure 2-14: Virtual Environment

All the pieces are designed based on the real world objects. The effect of gravity on robot `s parts act in the same way that it affects objects in real world. For instance, if the motors used in the robot body are not engaged, the effect of gravity will result in moving all links to lower positions.

The investigation of robot components and abilities seems necessary at this point. The virtual Delta robot consists of the exact components of the original Delta robot concept which is explained at the beginning of this chapter. A fixed platform at the top is responsible for holding the whole system. The zero point of the Cartesian system is designed to be placed at the center of this platform. Three motors have been fixed at the three fixed platform and kinematic chains joint

points. Each kinematic chain includes two links. The upper link and the bottom link which are called r_f and r_e respectively. Three revolute joints connect the r_fs to the fixed platform and three universal joints connect r_fs to r_es. The motion of motors cause movements on all the system as discussed before in the section of mathematical modeling. The kinematic chains are followed by a mobile platform. The end effector is placed at the center of the mobile platform. The position of the end effector in the Cartesian system (point zero at the center of the fixed platform) is the ultimate purpose of the robot operation. The angles of the motors determine the final position of the end effector. In other words, if a user intends to move an object with this robot, they should manipulate the robot by changing the motor angles to get to the position of the object. Figure 2-15 shows different parts of the robot and the external object.

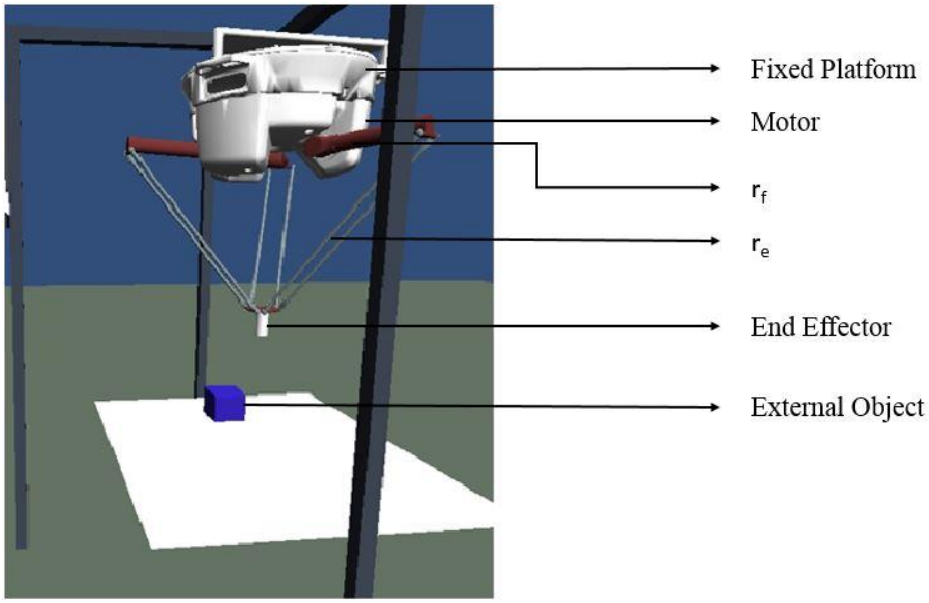

Figure 2-15: Virtual Robot `s Parts

Moving to the next step, the specifications of the elements in the virtual environment have been defined. Those specifications include the mass of each piece, the structure of the robot, and the specifications of the motors, and so forth. As it is shown on the top left side of Figure 2-14, a

window with four tabs is designed for changing physical options. The four tabs designed to make alterations in the physical options are Status, Mass, Motors, and PID.

The tab Status is designed with the purpose of showing the angles of the motors. The three angles, as Figure 2-16 illustrates, are the three motor angles. For manipulating the robot and moving the end effector around, changing these angles is required.

Figure 2-16: Status Tab

The next tab is for changing different parts of the robot `s masses. The three upper links, three bottom links, and the mobile platform are the user `s option in this window.

Figure 2-17: Mass Tab

The third tab is motors` specifications. This tab enables users to change the three motors` specifications of the resistance, torques, voltages, amperes, and gearbox ratios. The Figure 2-18 shows these options in the virtual environment.

Figure 2-18: Motor Tab

And the last tab is PID. The tab of PID provides the users with the options of changing the gain of proportional, integral, and derivative controllers. In addition to manipulating and programing the virtual robot, the students and trainees can get familiar with the concept of PID controllers and observe the output of each set of gains. The Figure 2-19 illustrate the PID control options.

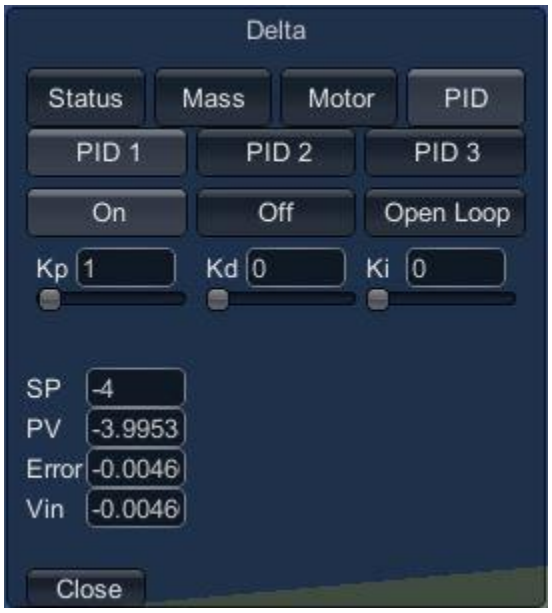

Figure 2-19: PID Tab

Moreover, the three coils have been designed aiming at monitoring the position of the end effector, the end effector sensor, and the condition of the end effector (ON/OFF conditions). These coils have the option of “On” or “Off”. Both of such options can be used as inputs of the user interface design.

Figure 2-20: Coils

3 Methodology

3.1 Problem Statement

There are various significant downsides to the application of actual robots in robotic technology laboratories. Obstacles such as the high expenses of building, and equipping robotic laboratory as well as the costs of actual robots, vulnerability of robots to wrong coding, programming and error instructions leading to serious damages to robots, and safety concerns related to lack of adequate supervision on trainees while performing relevant procedures are among those downsides which definitely impairs the use of real robots by novice trainees and users.

To address such an existing shortage, the combination of platforms for virtual developments, physics engines, computer processors, and graphic processors have been worked on in order to implement a virtual robot laboratory.

Having defines and illustrated all concepts of mathematical modeling, virtual reality, physics engines and computer processor in making a virtual robot laboratory, there is still one major requirement for the users to be able to benefit from such technology. This requirement is having access to an appropriate user interface that makes the communication possible between the virtual robot laboratory and the virtual robot with the trainee or user.

3.2 Purpose of the Study

The purpose of this study is to develop such a user interface having the capability to meet all those expected requirements in a user friendly, straightforward manner that is also beneficial for all the end users, both in academic and industrial training environments, in order to provide them with a profound robotic training experience and simultaneously, removes the challenges and potential risks of robot vulnerability to damage by wrong coding and instruction, and safety issues.

By applying virtual robot laboratory and making the communication between it and the intended users, not only all the previously mentioned constraints of actual robotic technology are removed, but also trainees gain the opportunity to spend more time practicing and performing various coding and programming scenarios to make training sessions fruitful, safe, challenging and robotically successful.

4 Procedure and Findings

4.1 User Interface

The works done so far regarding the history of robots and robotic knowledge, the training trend in robotic laboratory, the virtual environment and virtual robot, the inverse and forward kinematics behind the robot movements, and the Modbus TCP protocol for communication were discussed and presented in all previous chapters. Also, the mathematical modeling of delta robot and its implementation in virtual environment (applying Unity) were used to build a virtual robot laboratory.

Having the virtual robotic laboratory and the virtual robot, a user interface is required to provide the main connection between the users and the virtual robot in order to take advantage of the virtual robotic laboratory. As discussed earlier in the introduction part, the effort of this thesis is to meet the necessity of such a user interface to be installed on students/ trainees` computers to provide them with the ability to manipulate the virtual robot. What has been done in the scope of this thesis is the development of the aforementioned user interface so that the user interface can communicate with the virtual Delta robot via applying the Modbus TCP as the communication protocol. The whole concept of the work done in this work is depicted in the Figure 4-1. It shows that the virtual robot laboratory is installed on a computer counted as a server in this case. Also, it depicts that the designed user interface is installed on the user's computer, and finally shows the communication between the installed virtual robotic laboratory on the server and the user interface.

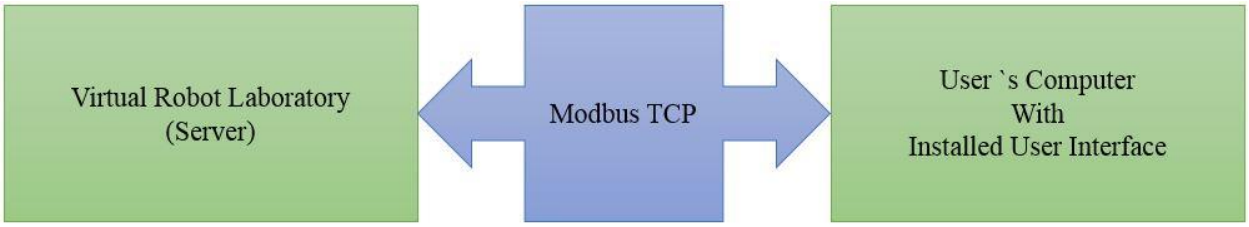

Figure 4-1: Concept

There are various steps in the development of the user interface (software) for the virtual robot laboratory. In the first step, we need to investigate the users' needs through the process of using the user interface in order to develop the software. The author has gained such requirements through the brainstorming and monitoring of the robot laboratory training sessions` procedures at Morehead State University robotics laboratory. Those requirements/ expectations of the software would be as the Figure 4-2 shows.

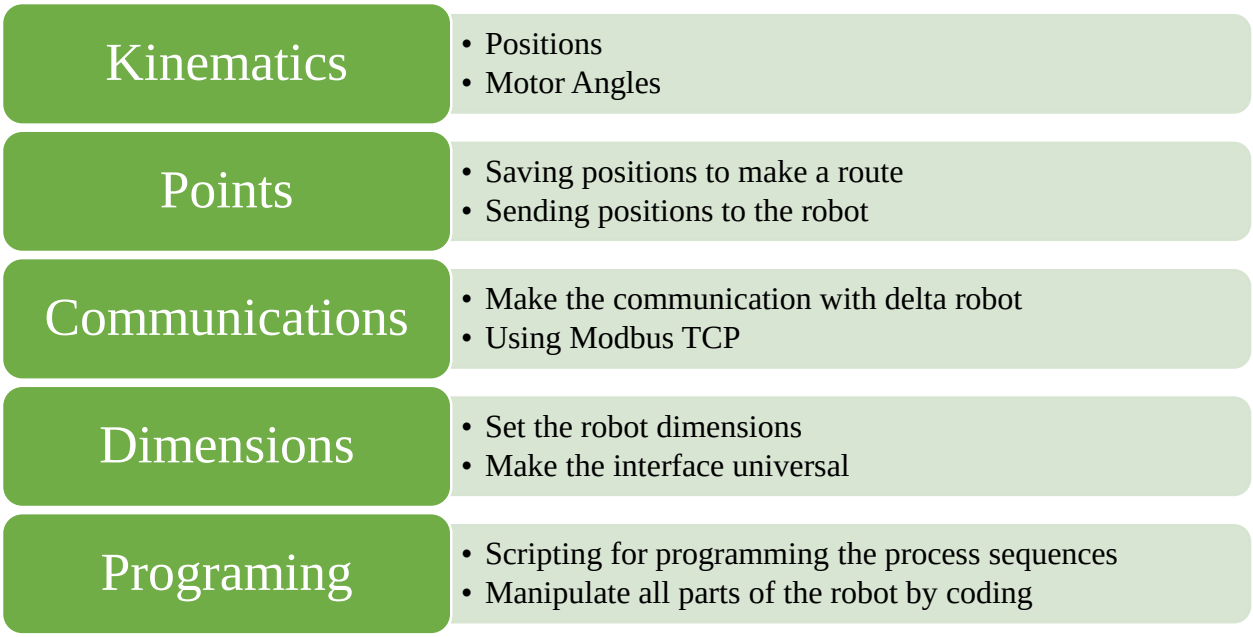

Figure 4-2: User Interface Requirements

As Figure 4-2 shows, the first requirement for the development of the user interface is Kinematics. By Kinematics, we mean that users should have the ability of sending the positions or motor angles to the virtual robot. As explained earlier in the mathematical modeling section, there are two main concepts discussed as inverse kinematics and forward kinematics.

In the concept of inverse kinematics, having a coordinates of a specific position of the end effector, there would exist three motor angles. In other words, when the user intends to move the virtual robot to a specific end effector position, how the three motors should move in order to reach

that specific point. For this purpose, repeating the formulas (Equation 2-21), (Equation 2-22), and (Equation 2-23) , we have the three motor angles as the following in which J_1 is the universal joint point between upper and bottom links. Calculating the position on this point would give the motor angles (Msavatsky, 2009).

$$\theta_1 = \tan^{-1}\frac{z_{J1}}{y_{F1} - y_{J1}} \qquad \textit{(Equation 4-1)}$$

$$\theta_2 = \tan^{-1}\frac{z'_{J1}}{y'_{F1} - y'_{J1}} \qquad \textit{(Equation 4-2)}$$

$$\theta_3 = \tan^{-1}\frac{z''_{J1}}{y''_{F1} - y''_{J1}} \qquad \textit{(Equation 4-3)}$$

Accomplishing the inverse kinematic calculations and coding the procedures of extracting angles will give the capability of motor angles` calculations for every chosen positions. Thus, applying the user interface, the trainee can practice working with the virtual robot inverse kinematics by applying the given end effector position and having the calculated motor angles.

On the other hand, forward kinematics has the opposite direction for calculation. In other words, when users choose the motor angles as the input variables in the user interface, they can see the calculated specific end effector position as the desired output variable. Such options provided in the user interface makes the users able to monitor the movement path of the robot with respect to the motor angles. By altering the input variable values for the motor angles, the altered positions of the end effector of the virtual robot can be monitored by the users. For this purpose,

repeating the formula (Equation 2-49) and solving the quadric equation will give the z component of the ultimate position (Msavatsky, 2009).

$$(a_1^2 + a_2^2 + 1)z^2 + 2(a_1 + a_2(b_2 - y_1) - z_1)z + (b_1^2 + (b_2 - y_1)^2 + z_1^2 - r_e^2) = 0 \quad \textit{(Equation 4-4)}$$

Substituting z_0 in equations (Equation 4-5) and (Equation 4-6) will give x_0 and y_0 at the output. The final coordinate of the end effector position would be x_0, y_0, and z_0 (Msavatsky, 2009).

$$x = a_1 z + b_1 \quad \textit{(Equation 4-5)}$$

$$y = a_2 z + b_2 \quad \textit{(Equation 4-6)}$$

In which, a_1, b_1, a_2, and b_2 would be (Msavatsky, 2009):

$$a_1 = \frac{1}{d}\left[\left((z_2 - z_1) \times (y_3 - y_1)\right) - \left((z_3 - z_1) - (y_2 - y_1)\right)\right] \quad \textit{(Equation 4-7)}$$

$$a_2 = \frac{-1}{d}\left[\left((z_2 - z_1)x_3\right) - \left((z_3 - z_1)x_2\right)\right] \quad \textit{(Equation 4-8)}$$

$$b_1 = \frac{-1}{2d}\left[\left((w_2 - w_1) \times (y_3 - y_1)\right) - \left((w_3 - w_1) - (y_2 - y_1)\right)\right] \quad \textit{(Equation 4-9)}$$

$$b_2 = \frac{1}{2d}\left[\left((w_2 - w_1)x_3\right) - \left((w_3 - w_1)x_2\right)\right] \quad \textit{(Equation 4-10)}$$

4.1.1 Control Window

Accomplishing the calculation of the forward kinematics and coding the procedures of extraction positions for the given angles will give the capability of manipulating the virtual delta robot by changing motor angles. Thus, the user can practice various values of the motor angles as the input variables in order to see various relevant output variables and continually observe the movement paths by manipulating variables to manipulate the virtual robot, and consequently, master such skills and knowledge.

Figure 4-3 illustrates the design of Kinematics window in the developed user interface. There are two modes for forward and inverse kinematics designed in the user interface called the Joint mode and the World mode. The Joint mode offers the forward kinematics options while the World mode offers the inverse kinematics options.

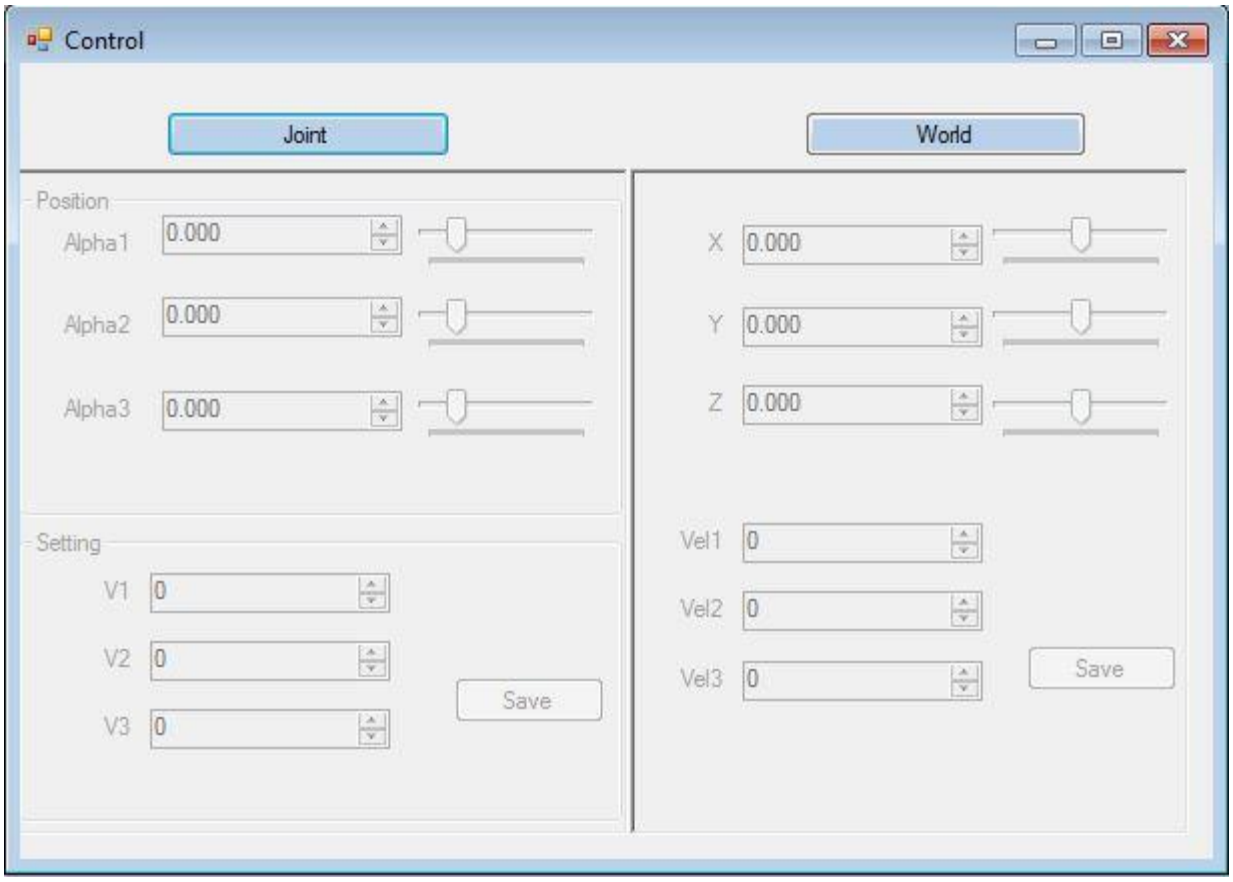

Figure 4-3: Control Window

By selecting the Joint mode, user can change the motor angles and simultaneously, the interface calculates the correspondence position. The Figure 4-4 shows one example of this process.

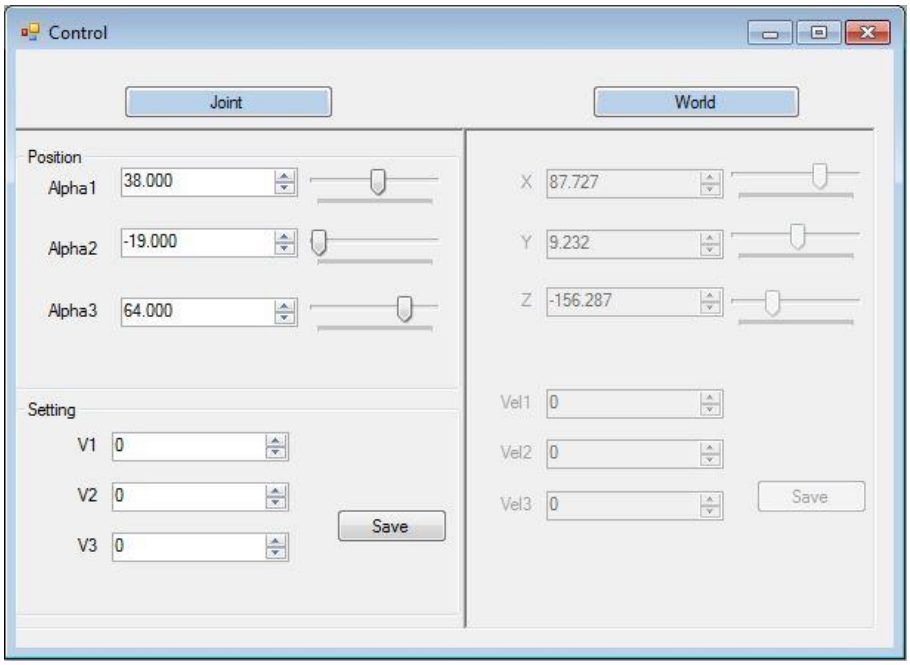

Figure 4-4: Joint Mode

By choosing the World mode, user can alter the position and simultaneously, the interface calculates the correspondence motor angles. The Figure 4-5 shows one example of this process.

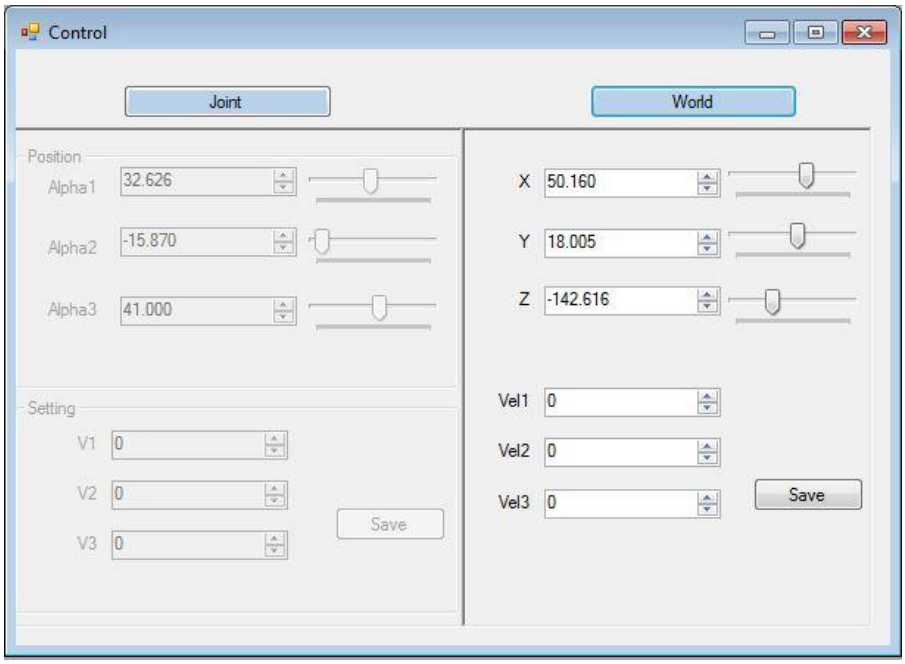

Figure 4-5: World Mode

4.1.2 Save Position Window

When the users find the desirable end effector position, either by manipulating the robot in Joint or World mode, they should be able to save that position. As each path includes some points, saving each position make the user enable of programing a specific route for the robot. To meet this requirement, a window was designed which has communication with a .csv file outside the user interface. This .csv file is called in a datagridview through this window and by saving each position, the correspondence x, y, and z will be sent to this file. The important consideration about saving points is that since the virtual robot has no understanding about positions and just works with motor angles, the saving points in the .csv file are based on the motor angles. Figure 4-6 illustrates the designed window in the user interface.

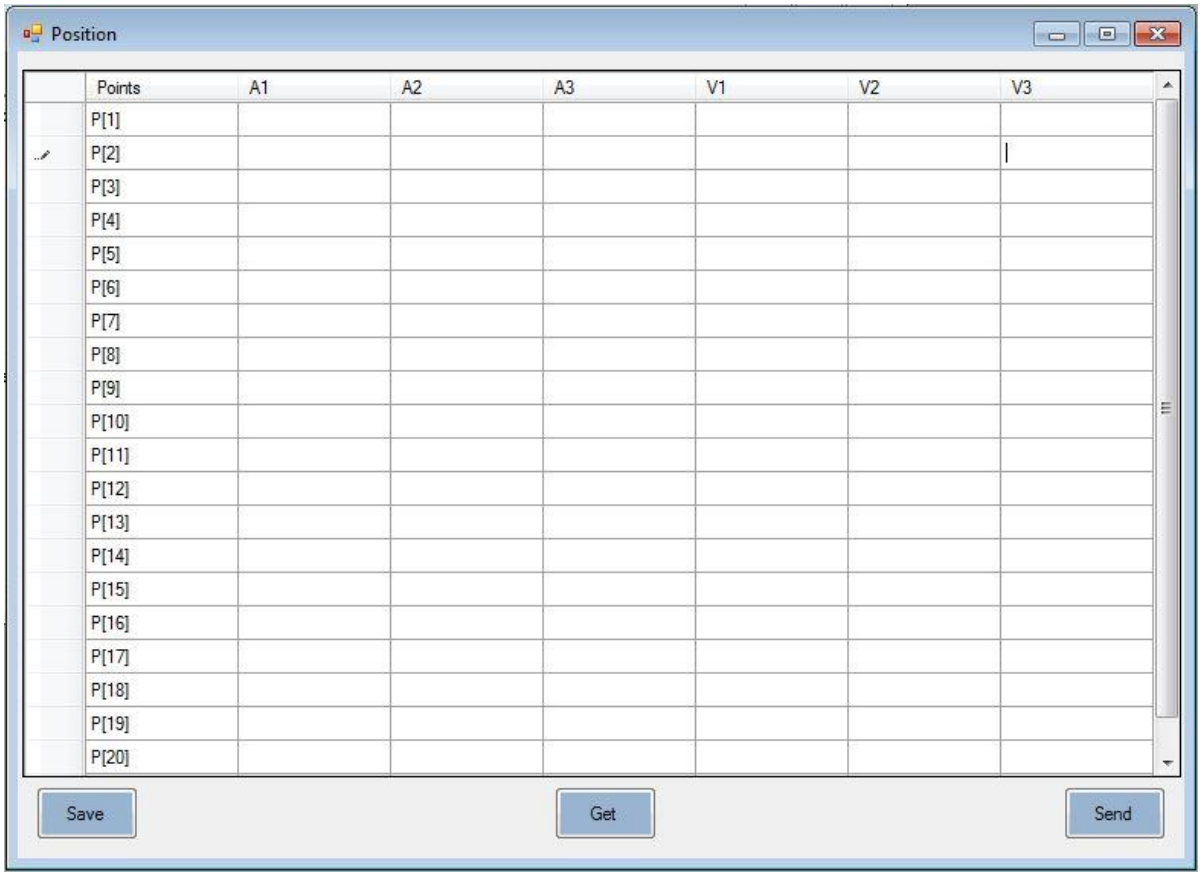

Figure 4-6: Save Positions

4.1.3 Communication

In addition to invers and forward kinematics which are the fundamental parts of robot operation, communication plays the critical role. For implementing virtual robot laboratory, the virtual robot will be installed on the university or company `s computer server. The user interface would be installed on student/trainee `s computer. The communication between the virtual laboratory and the user interface would be based on Modbus TCP. As mentioned before in the communication part, the Modbus TCP is a five layer industrial protocol in which the communication uses TCP frame format to build a compatible data exchange over Ethernet. For making this communication happen, there should be a window that students/trainees can enter the server IP address and the Modbus point. The communication window was designed as shown in Figure 4-7 to fulfil such purpose.

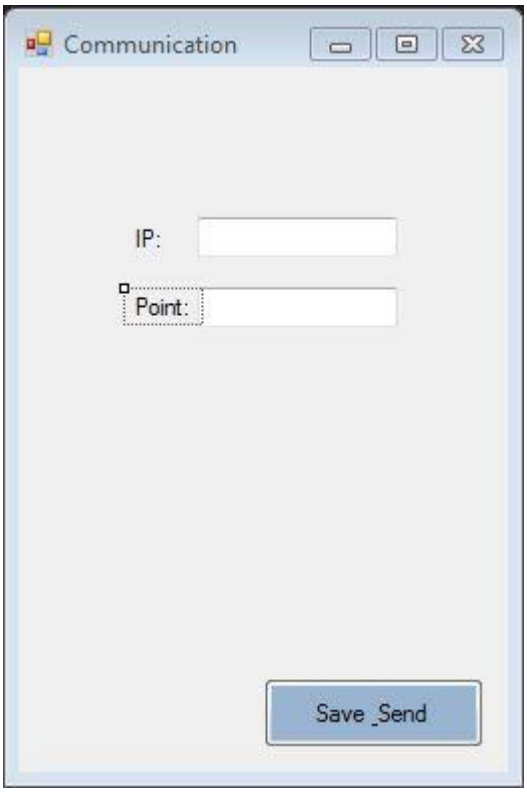

Figure 4-7: Communication

4.1.4 Dimensions

Forward and inverse kinematics are based on the dimensions of the virtual Delta robot. The calculation of the end effector position depends on the length of the kinematic links and the platforms sides. The fixed dimensions on the user interface programing make it special for manipulating just one robot. To make the user interface universal and to make its application possible for any Delta robots, users need a window to set the measurements. Figure 4-8 shows the dimensions which are used for kinematics calculations.

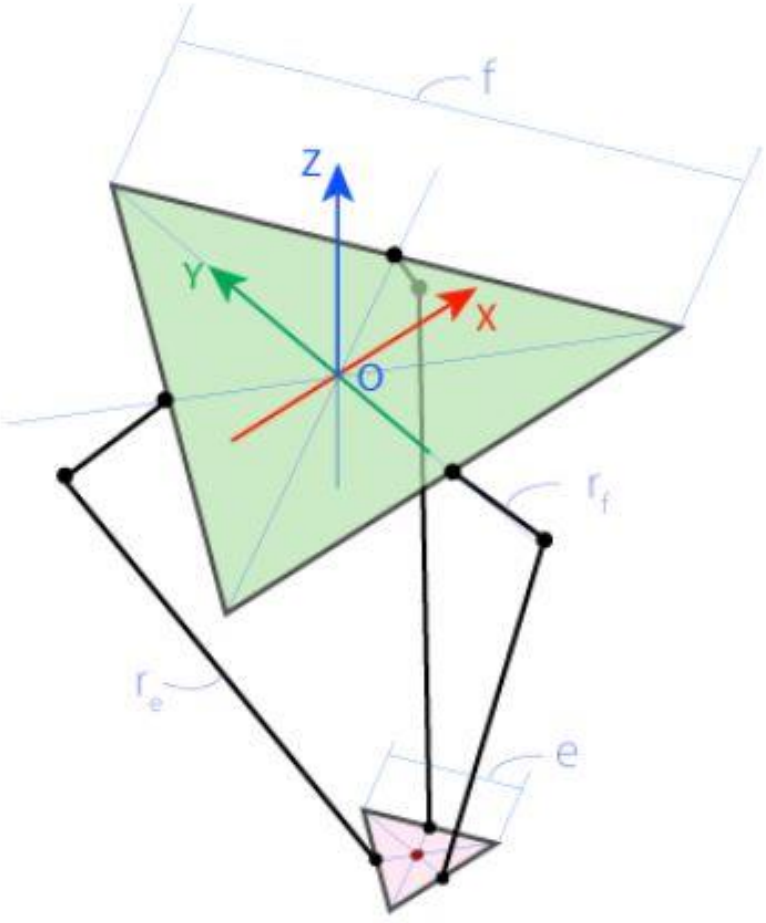

Figure 4-8: Delta Robot Dimensions (Msavatsky, 2009)

And the design window for setting these dimensions in the user interface is depicted in the Figure 4-9.

Figure 4-9: Communication

The final appearance of the user interface including various windows of Control of Kinematics, Position, Communication, Dimensions, and Programming is shown in the Figure 4-10.

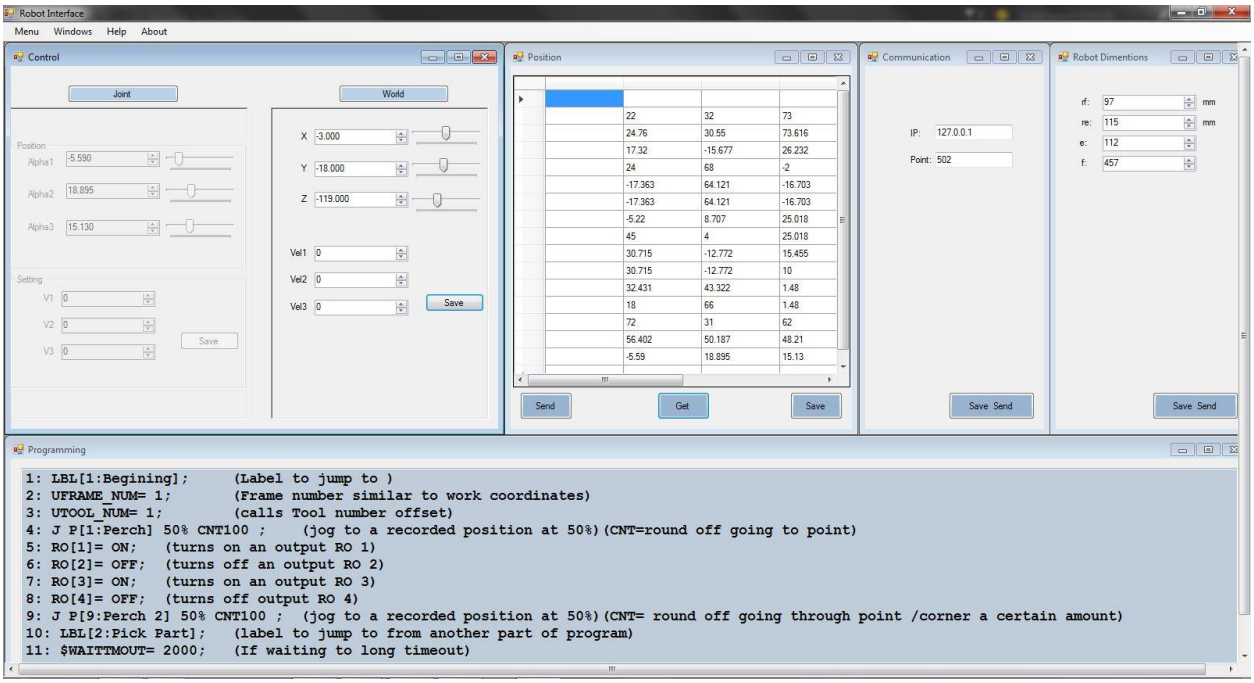

Figure 4-10: Final View

As Figure 4-10 shows, the final look of the developed user interface in the scope of this thesis includes various windows. The simple and clear look of the windows as well as its being self-explanatory provide the users and trainees of the designed software with a great opportunity to learn how to work with different scenarios relevant to the inverse and forward kinematics. Doing so, the users are able to monitor different movement paths of the virtual robot and analyze the behavior of the virtual robot resembling the actual robot with high precision in details. In addition, the software enables the learners to save the practiced points, perform the communication protocols, observe the dimensions and programming windows and in general, develop a much deeper grasp of robotic technology in an available, safe and straightforward virtual environment that resembles the specifications of the real robotic technology laboratories and actual Delta robots with high accuracy and precision.

5 Conclusion and Future Works

5.1 Conclusion

Robots have been applied in industry, healthcare and medical surgeries, military, agriculture, oceanographic explorations, education, and for aerospace purposes. Robots are turning into more intelligent machines with the ability of processing information much faster and more efficiently in comparison to their early versions. Today, industrial robots are an integral part of the automotive assembly lines due to their higher speed, quality, reliability and productivity. Considering the wide applications of robots in almost all various fields in the modern world, there is always need for engineers, operators, technicians and experts to perform tasks of programming, maintenance, operating and troubleshooting of robots. Training professionals and experts in the robotic areas is necessary due to the critical processes in which robots are involved. However, the main constraints regarding providing continuous training in the field of robotic technology include lack of training resources, unsatisfactory training processes, high costs of equipping robotic laboratories, high sensitivity of working with robots for unskilled individuals, high risks of making mistakes and damaging robots. As a solution, virtual robot laboratories are developed to resolve such issues. The whole concept of virtual robot training is based on implementing virtual robot laboratory and virtual robot with the exact behavior of the actual robot. Applying both virtual robots and virtual robot laboratories, trainees are made able to implement various scenarios and coding various sequences and they can monitor the real behavior of the robots as a result of applying such scenarios and sequences. Trainees would find the opportunity to practice all possible conditions as a result of scripting various scenarios.

In order to take advantage of virtual robotic laboratory, a user interface (software) is developed during this study. The straightforwardness of the designed software provides novice

users with a great opportunity to develop a profound understanding and knowledge of robotic technology in an available, safe and straightforward virtual environment that resembles the specifications of the real robotic technology laboratories and actual Delta robots with high accuracy and precision. In addition, applying the user interface makes companies able of reducing their training expenses significantly and assures them of having well-trained operators due to the specifications of the developed software.

5.2 Future Works

Referring to the previous chapters, one can see how the developed user interface consists of five various windows of Control, Position, Communication, Dimension and Programming. However, in the scope of this thesis, only the first four windows are developed for the use by trainees. Future research can be conducted on developing the programming window. In other words, future work can be done in order to make it possible for the users to do the processes of scripting for programming the process sequences as well as manipulating all parts of the robot by coding. This added feature will further strengthen the training processes by enabling the users to gain a mastery level in performing various tasks with the virtual robot and figuring out its behaviors in different scenarios defined by themselves.

6 References

Acromag. (2005). Retrieved from Acromag: https://www.acromag.com/sites/default/files/Acromag_Intro_ModbusTCP_765A.pdf

Brian Scassellati, Henny Admoni, Maja Mataric. (2012). *Annual Reviews*. Retrieved from Annual Reviews: http://www.annualreviews.org/doi/full/10.1146/annurev-bioeng-071811-150036

Brogardh, T. (2000). Design of high performance parallel arm robots for industrial application. *A Symposium Commemorating the Legacy, Works, and Life of Sir Robert Stawell Ball.*

Brumson, B. (2007). *Robotic Industrial Association*. Retrieved from Robotic Industrial Association: http://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/Chemical-and-Hazardous-Material-Handling-Robotics/content_id/614

Brumson, B. (2011). *Robotic Industries Association*. Retrieved from Robotic Industries Association: http://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/Robotics-in-Security-and-Military-Applications/content_id/3112

Etienne van Wyk, R. d. (2014). Applying design-based research for developing virtual reality training in the South African mining industry.

Grabianowski, E. (2016). *How Miliraty Robots Work*. Retrieved from How Sruff Works-Science: http://science.howstuffworks.com/military-robot.htm

Maurizio Rovaglio, Tobias Scheele. (2011). Virtual reality improves training in process industries. *Automation IT*, 32-36.

Moody, J. O., Sánchez- Alonso, R., Yun, C., González- Barbosa, J., & Reyes- Morales, G. (2015). Virtual labratory of idustrial scenarios for training in the areas of automation and control. *International Mechanical Engineering Congress & Exposition*, (p. 5). Houston.

Msavatsky. (2009). *Delta Robot Kinematics*. Retrieved from Trossen Robotics Community: http://forums.trossenrobotics.com/tutorials/introduction-129/delta-robot-kinematics-3276/

Pan, Z., Polden, J., Larkin, N., Duin, S. V., & Norrish, J. (2012). Recent progress on programming methods for industrial robots. *Robotics and Computer-Integrated Manufacturing*, 87-94.

Papakostas, N., Michalos, G., Makris, S., Zouzias, D., & Chryssolouris, G. . (2011). Industrial applications with cooperating robots for the flexible assembly. *International Journal of Computer Integrated Manufacturing*, 24(7).

RTA. (n.d.). Retrieved from RTA: http://www.rtaautomation.com/technologies/modbus-tcpip/

ScottCompany. (n.d.). *RobotWorx*. Retrieved from RobotWorx: https://www.robots.com/articles/viewing/benefits-of-robots

Unity. (n.d.). Retrieved from Unity: https://unity3d.com/

Virtual Reality Society. (2016). Retrieved from Virtual Reality Society: http://www.vrs.org.uk/virtual-reality/beginning.html

Williams, R. (2016). The Delta Parallel Robot: Kinematics Solutions. Ohio. Retrieved from www.ohio.edu/people/williar4/html/pdf/DeltaKin.pdf

7 Appendix A

Interface Form Main Source Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RobotInterface
{
    public partial class FormMain : Form
    {
        public FormMain()
        {
            InitializeComponent();
        }
        FormMain f1;
        private void Form1_Load(object sender, EventArgs e)
        {

        }


        frmControl fControl;
        public void controlToolStripMenuItem_Click(object sender, EventArgs e)
        {

            if (fControl == null)
            {
                fControl = new frmControl();
                fControl.MdiParent = this;
                fControl.FormClosing += F2_FormClosing;
                fControl.Show();
            }
            else
            {
                fControl.Show();
                fControl.Activate();
            }
          }

        public void F2_FormClosing(object sender, FormClosingEventArgs e)
        {
            e.Cancel = true;
            fControl.Hide();
        }

        void f2_FormClosed(object sender, FormClosedEventHandler e)
        {
            fControl = null;
        }
```

```
public frmPosition fPosition;
private void positionToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (fPosition == null)
    {
        fPosition = new frmPosition();
        fPosition.MdiParent = this;
        fPosition.FormClosing += F3_FormClosing;
        fPosition.Show();
    }
    else
    {
        fPosition.Show();
        fPosition.Activate();
    }
}

private void F3_FormClosing(object sender, FormClosingEventArgs e)
{
    e.Cancel = true;
    fPosition.Hide();
}
void f3_FormClosed(object sender, FormClosedEventHandler e)
{
    fPosition = null;
}

frmCommunication fComm;
private void communicationToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (fComm == null)
    {
        fComm = new frmCommunication();
        fComm.MdiParent = this;
        fComm.FormClosing += F4_FormClosing;
        fComm.Show();
    }
    else
    {
        fComm.Show();
        fComm.Activate();
    }
}

private void F4_FormClosing(object sender, FormClosingEventArgs e)
{
    e.Cancel = true;
    fComm.Hide();
}

void f4_FormClosed(object sender, FormClosedEventHandler e)
{
    fComm = null;
}

FormProgramming fProg;
public void programmingToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{

    if (fProg == null)
    {
        fProg = new FormProgramming();
        fProg.MdiParent = this;
        fProg.FormClosing += F5_FormClosing;
        fProg.Show();
    }
    else
    {
        fProg.Show();
        fProg.Activate();
    }
}

public void F5_FormClosing(object sender, FormClosingEventArgs e)
{
    e.Cancel = true;
    fProg.Hide();
}
void f5_FormClosed(object sender, FormClosedEventHandler e)
{
    fProg = null;
}

frmRobotDimentions fRobotDim;
private void robotDimentionsToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (fRobotDim == null)
    {
        fRobotDim = new frmRobotDimentions();
        fRobotDim.MdiParent = this;
        fRobotDim.FormClosing += F6_FormClosing;
        fRobotDim.Show();
    }
    else
    {
        fRobotDim.Show();
        fRobotDim.Activate();
    }
}

private void F6_FormClosing(object sender, FormClosingEventArgs e)
{
    e.Cancel = true;
    fRobotDim.Hide();
}
void f6_FormClosed(object sender, FormClosedEventHandler e)
{
    fRobotDim = null;
}

public void FormMain_SizeChanged(object sender, EventArgs e)
{
    //Console.WriteLine("Main form is resized....");

    fControl.AutoResize();
```

```
            fPosition.AutoResize();
            fProg.AutoResize();
            fComm.AutoResize();
            fRobotDim.AutoResize();

        }

        public void FormMain_FormClosing(object sender, FormClosingEventArgs e)
        {
            System.Environment.Exit(1);
        }

    }

}
```

8 Appendix B

Interface Form Control Source Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using Excel = Microsoft.Office.Interop.Excel;

using System.Reflection;
using System.IO;
using System.Data.OleDb;
using System.Web;
using EasyModbus;
using System.Threading;

namespace RobotInterface
{

    public partial class frmControl : Form
    {
        public bool running = false;
        public static bool newPosition = false;
        Thread mthread;
        public const int WM_NCLBUTTONDBLCLK = 0x00A3;
        public  string axe;


        public frmControl()
        {
            InitializeComponent();
            running = true;
            this.mthread = new Thread(new ThreadStart(this.modbusClient));
            this.mthread.Start();

        }

        public void frmControl_Load(object sender, EventArgs e)
        {

        }

        protected override void WndProc(ref Message m)
        {
            if (m.Msg == WM_NCLBUTTONDBLCLK)
            {
                //this.RelocatedForm();
                AutoResize();
                m.Result = IntPtr.Zero;
                return;
```

```
    }
    base.WndProc(ref m);
}


public void AutoResize()
{
    // Resize the form and relocated
    this.Width = (int)(MdiParent.ClientSize.Width * 0.4);
    this.Height = (int)(MdiParent.ClientSize.Height * 0.6);

    int x = 0;
    int y = 0;
    this.Location = new Point(x, y);

    // Resize the controls in the from
}
private bool v;
private void btnWorld_Click(object sender, EventArgs e)
{
    this.Enabled = true;
    this.panel2.Enabled = false;
    this.panel3.Enabled = true;
    v = true;

}

private void btnJoint_Click(object sender, EventArgs e)
{
    this.Enabled = true;
    this.panel3.Enabled = false;
    this.panel2.Enabled = true;
    v = false;



}
public static double A1 = 0;
public static double A2 = 0;
public static double A3 = 0;
// World
private void TrackBar5_Scroll(object sender, EventArgs e)
{


    A1 = positionTBX.Value;
    positionX.Value = (decimal)A1;

}
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{

    A1 = (double)positionX.Value;
    positionTBX.Value = (int)A1;
    if (v == true)
    {
        axe = "X";
        sendAngles();
```

```
        newPosition = true;
    }

}


private void textBox1_TextChanged(object sender, EventArgs e)
{

}
private void trackBar7_Scroll(object sender, EventArgs e)
{
    axe = "Y";
    A2 = positionTBY.Value;
    positionY.Value = (decimal)A2;
}
private void numericUpDown2_ValueChanged(object sender, EventArgs e)
{

    A2 = (double)positionY.Value;
    positionTBY.Value = (int)A2;
    if (v == true)
    {
        sendAngles();
        newPosition = true;
    }

}
private void TrackBar4_Scroll(object sender, EventArgs e)
{

    A3 = positionTBZ.Value;
    positionZ.Value = (decimal)A3;
}

private void numericUpDown3_ValueChanged(object sender, EventArgs e)
{

    A3 = (double)positionZ.Value;
    positionTBZ.Value = (int)A3;
    if (v == true)
    {
        axe = "Z";
        sendAngles();
        newPosition = true;
    }

}



// Joint
public static double B1;
public static double B2;
public static double B3;

public void TrackBar1_Scroll(object sender, EventArgs e)
{
```

```
    B1 = angleTBT1.Value;
    angleT1.Value = (decimal)B1;

}
private void angleT1_ValueChanged_1(object sender, EventArgs e)
{
    B1 = (double)angleT1.Value;
    angleTBT1.Value = (int)B1;
    if (v == false)
    {
        sendPositions();
        newPosition = true;
    }


}
private void TrackBar2_Scroll(object sender, EventArgs e)
{
    B2 = angleTBT2.Value;
    angleT2.Value = (decimal)B2;
}
private void angleT2_ValueChanged(object sender, EventArgs e)
{
    B2 = (double)angleT2.Value;
    angleTBT2.Value = (int)B2;
    if (v == false)
    {
        sendPositions();
        newPosition = true;
    }


}
private void TrackBar3_Scroll(object sender, EventArgs e)
{
    B3 = angleTBT3.Value;
    angleT3.Value = (decimal)B3;
}
private void angleT3_ValueChanged(object sender, EventArgs e)
{
    B3 = (double)angleT3.Value;
    angleTBT3.Value = (int)B3;
    if (v == false)
    {
        sendPositions();
        newPosition = true;
    }


}


public static double frmRobotDimentions.die;
public static double frmRobotDimentions.dif;
public static double frmRobotDimentions.dire;
public static double frmRobotDimentions.dirf;

private double s = 330;
```

```
private double sqrt3 = Math.Sqrt(3.0);

private double pi = 3.141592653;
private double sin120 = Math.Sqrt(3.0) / 2.0;
private double cos120 = -0.5;
private double tan60 = Math.Sqrt(3.0);
private double sin30 = 0.5;
private double tan30 = 1 / Math.Sqrt(3.0);

private double x0;
private double y0;
private double z0;

private double t;

private double x1;
private double x2;
private double x3;

private double y1;
private double y2;
private double y3;

private double z1;
private double z2;
private double z3;

private double w1;
private double w2;
private double w3;

private double a;
private double b;
private double c;

private double a1;
private double a2;

private double b1;
private double b2;

private double d;
private double dnm;

private double T1;
private double T2;
private double T3;

private double yj;
private double zj;

private double x20;
private double y20;
private double a20;
private double b20;
private double d1;
private double yj1;
private double zj1;
```

```
private double y30;

private double f1;
private double f2;
private double f3;
private double f4;

private double x40;
private double y40;
private double a40;
private double b40;
private double d2;
private double yj2;
private double zj2;
private double y50;



void sendPositions()
{
    T1 =  (double)angleT1.Value;
    T2 =  (double)angleT2.Value;
    T3 =  (double)angleT3.Value;

    T1 = T1 * pi / 180;
    T2 = T2 * pi / 180;
    T3 = T3 * pi / 180;

    x0 = 0.0;
    y0 = 0.0;
    z0 = 0.0;

    t = (f - e) * tan30 / 2.0;


    y1 = -(t + (rf * Math.Cos(T1)));
    z1 = -rf * Math.Sin(T1);

    y2 = (t + (rf * Math.Cos(T2))) * sin30;
    x2 = y2 * tan60;
    z2 = -rf * (Math.Sin(T2));

    y3 = (t + rf * (Math.Cos(T3))) * sin30;
    x3 = -y3 * tan60;
    z3 = -rf * (Math.Sin(T3));

    dnm = ((y2 - y1) * x3) - ((y3 - y1) * x2);

    w1 = (y1 * y1) + (z1 * z1);
    w2 = (x2 * x2) + (y2 * y2) + (z2 * z2);
    w3 = (x3 * x3) + (y3 * y3) + (z3 * z3);

    a1 = (z2 - z1) * (y3 - y1) - (z3 - z1) * (y2 - y1);
    b1 = -((w2 - w1) * (y3 - y1) - (w3 - w1) * (y2 - y1)) / 2.0;


    a2 = -(z2 - z1) * x3 + ((z3 - z1) * x2);
    b2 = (((w2 - w1) * x3) - ((w3 - w1) * x2)) / 2.0;
```

```
            a = (a1 * a1) + (a2 * a2) + (dnm * dnm);
            b = 2.0 * ((a1 * b1) + (a2 * (b2 - (y1 * dnm))) - (z1 * dnm * dnm));
            c = (b2 - (y1 * dnm)) * (b2 - (y1 * dnm)) + (b1 * b1) + (dnm * dnm * ((z1 *
z1) - (re * re)));

            d = (b * b) - (4.0 * a * c);

            z0 = (-0.5 * (b + Math.Sqrt(d))) / a;
            x0 = ((a1 * z0) + b1) / dnm;
            y0 = ((a2 * z0) + b2) / dnm;



            x0 = Math.Round(x0, 3);
            y0 = Math.Round(y0, 3);
            z0 = Math.Round(z0, 3);

            positionX.Value = (decimal)x0;
            positionY.Value = (decimal)y0;
            positionZ.Value = (decimal)z0;


            finalCal();

        }



        // Update is called once per frame
        void sendAngles()
        {
            x0 = (double)positionX.Value;
            y0 = (double)positionY.Value;
            z0 = (double)positionZ.Value;

            T1 = 0;
            T2 = 0;
            T3 = 0;

            y1 = -0.5 * 0.57735*f;
            y30 = y0 - (0.5 * 0.57735 * e);

            a = ((x0 * x0) + (y30 * y30) + (z0 * z0) + (rf * rf) - (re * re) - (y1 * y1))
/ (2.0 * z0);
            b = (y1 - y30) / z0;

            d = -(a + b * y1) * (a + b * y1) + rf * (b * b * rf + rf);

            yj = (y1 - a * b - Math.Sqrt(d)) / (b * b + 1);
            zj = a + b * yj;
            T1 = (Math.Atan(-zj / (y1 - yj)) * (180 / pi));
            if (yj>y1) {
                T1 = T1 + 180;
                }
            else
            {
```

```
            }

            f1 = x0 * cos120;
            f2 = y0 * sin120;
            f3 = y0 * cos120;
            f4 = x0 * sin120;

            x20 = (f1 + f2);
            y20 = (f3 - f4);

            y30 = y20 - (0.5 * 0.57735 * e);

            a20 = ((x20 * x20) + (y30 * y30) + (z0 * z0) + (rf * rf) - (re * re) - (y1 *
y1)) / (2.0 * z0);
            b20 = (y1 - y30) / z0;

            d1 = -(a20 + b20 * y1) * (a20 + b20 * y1) + rf * (b20 * b20 * rf + rf);

            yj1 = (y1 - a20 * b20 - Math.Sqrt(d1)) / (b20 * b20 + 1);
            zj1 = a20 + b20 * yj1;
            T2 = (Math.Atan(-zj1 / (y1 - yj1)) * (180 / pi));
            if (yj1 > y1)
            {
                T2 = T2 + 180;
            }
            else
            {

            }

            x40 = ((x0 * cos120) - (y0 * sin120));
            y40 = ((y0 * cos120) + (x0 * sin120));

            y30 =y40 - (0.5 * 0.57735 * e);

            a40 = ((x40 * x40) + (y30 * y30) + (z0 * z0) + (rf * rf) - (re * re)- (y1 *
y1)) / (2.0 * z0);
            b40 = (y1 - y30) / z0;

            d2 = -(a40 + b40 * y1) * (a40 + b40 * y1) + rf * (b40 * b40 * rf + rf);

            yj2 = (y1 - a40 * b40 - Math.Sqrt(d2)) / (b40 * b40 + 1);
            zj2 = a40 + b40 * yj2;
            T3 = (Math.Atan(-zj2 / (y1 - yj2)) * (180 / pi));
            if (yj2 > y1)
            {
                T3 = T3 + 180;
            }
            else
            {

            }

            T1 = Math.Round(T1, 3);
            T2 = Math.Round(T2, 3);
            T3 = Math.Round(T3, 3);
```

```
    finalCal();

    try
    {
        if (T1 > 89 || T1 < -20 || T2 > 89 || T2 < -20 || T3 > 89 || T3 < -20)
        {
            switch (axe)
            {
                case "X":
                    A1--;
                    break;
                case "Y":
                    A2--;
                    break;
                case "Z":
                    A3--;
                    break;

            }

            Console.WriteLine("Not in the range");
        }

        else
        {
            angleT1.Value = (decimal)T1;
            angleT2.Value = (decimal)T2;
            angleT3.Value = (decimal)T3;
        }
    }
    catch
    {
        Console.WriteLine("Not in the range");
    }




}

void finalCal()
{
    testT4.Text = x0.ToString();
    testT5.Text = y0.ToString();
    testT6.Text = z0.ToString();

    testT1.Text = T1.ToString();
    testT2.Text = T2.ToString();
    testT3.Text = T3.ToString();
}

private float c1;
private float c2;
private float c3;
public static bool reachPos;
void modbusClient()
{
    ModbusClient client = new ModbusClient("127.0.0.1", 502);
```

```
        client.Connect();

    while (running)
        if (newPosition)
        {
            c1 = Convert.ToSingle(B1);
            byte[] aux0 = BitConverter.GetBytes(c1);
            byte[] aux1 = new byte[] { aux0[0], aux0[1] };
            byte[] aux2 = new byte[] { aux0[2], aux0[3] };

            ushort pos1 = BitConverter.ToUInt16(aux2, 0);
            ushort pos2 = BitConverter.ToUInt16(aux1, 0);

            c2 = Convert.ToSingle(B2);

            byte[] aux3 = BitConverter.GetBytes(c2);
            byte[] aux4 = new byte[] { aux3[0], aux3[1] };
            byte[] aux5 = new byte[] { aux3[2], aux3[3] };

            ushort pos3 = BitConverter.ToUInt16(aux5, 0);
            ushort pos4 = BitConverter.ToUInt16(aux4, 0);

            c3 = Convert.ToSingle(B3);

            byte[] aux6 = BitConverter.GetBytes(c3);
            byte[] aux7 = new byte[] { aux6[0], aux6[1] };
            byte[] aux8 = new byte[] { aux6[2], aux6[3] };

            ushort pos5 = BitConverter.ToUInt16(aux8, 0);
            ushort pos6 = BitConverter.ToUInt16(aux7, 0);

            int[] total = new int[] { pos1, pos2, pos3, pos4, pos5, pos6 };
            //try {
            client.WriteMultipleRegisters(0, total);
            newPosition = false;

            Thread.Sleep(50);
        }

        else
        {
            bool[] coils = client.ReadCoils(0, 3);
            reachPos = coils[1];
            Thread.Sleep(50);
        }

}



private void btnWorldCal_Click(object sender, EventArgs e)
{
    //sendAngles();
}

private void btnJointCal_Click(object sender, EventArgs e)
{
    //sendPositions();
```

```
        }


        int j = 1;
        private void button1_Click_1(object sender, EventArgs e)
        {
            ((FormMain)this.MdiParent).fPosition.dataGridView1.Rows.Add();
            int i =j;


                {
                ((FormMain)this.MdiParent).fPosition.dataGridView1[1, i].Value =
(float)angleT1.Value;
                ((FormMain)this.MdiParent).fPosition.dataGridView1[2, i].Value =
(float)angleT2.Value;
                ((FormMain)this.MdiParent).fPosition.dataGridView1[3, i].Value =
(float)angleT3.Value;

                }


            j++;
        }

        private void btnJointSave_Click(object sender, EventArgs e)
        {
            ((FormMain)this.MdiParent).fPosition.dataGridView1.Rows.Add();
            //int i = ((FormMain)this.MdiParent).fPosition.dataGridView1.RowCount;
            int i = j;



                ((FormMain)this.MdiParent).fPosition.dataGridView1[1, i].Value =
(float)angleT1.Value;
                ((FormMain)this.MdiParent).fPosition.dataGridView1[2, i].Value =
(float)angleT2.Value;
                ((FormMain)this.MdiParent).fPosition.dataGridView1[3, i].Value =
(float)angleT3.Value;


        j++;
        }
    }
}
```

9 Appendix C

Interface Form Position Source Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using Excel = Microsoft.Office.Interop.Excel;

using System.Reflection;
using System.IO;
using System.Data.OleDb;
using System.Web;
using Modbus.Data;
using Modbus.Device;
using Modbus.Utility;
using System.Net.Sockets;
using System.Net;
using System.Threading;



namespace RobotInterface
{
    public partial class frmPosition : Form
    {
        private const int WM_NCLBUTTONDBLCLK = 0x00A3;

        public frmPosition()
        {

            InitializeComponent();
        }
        public DataGridView DataGrid
        {
            get
            {
                return this.dataGridView1;
            }
        }
        public void frmPosition_Load(object sender, EventArgs e)
        {

        }
        protected override void WndProc(ref Message m)
        {
            if (m.Msg == WM_NCLBUTTONDBLCLK)
            {
                // this.RelocatedForm();
                AutoResize();
```

```
            m.Result = IntPtr.Zero;
            return;
        }
        base.WndProc(ref m);
    }

    private void RelocatedForm()
    {
        int parentWidth = this.MdiParent.Width;
        //int width = this.Width;
        int x = 636;
        int y = 0;
        this.Location = new Point(x, y);
    }


    public void AutoResize()
    {
        // Resize the form and relocated

        this.Width = (int)(MdiParent.ClientSize.Width * 0.2);
        this.Height = (int)(MdiParent.ClientSize.Height * 0.6);

        int x = (int)(MdiParent.ClientSize.Width * 0.4);
        int y = 0;
        this.Location = new Point(x, y);
    }

    public void btnGet_Click(object sender, EventArgs e)
    {

        int counter = 0;
        string line;
        dataGridView1.Rows.Clear();
        dataGridView1.ColumnCount = 7;


        System.IO.StreamReader file =
            new StreamReader(@"C:\Users\Armin\Documents\Visual Studio
2015\Projects\RobotInterface\RobotInterface\bin\Debug\Positions.csv");
        while ((line = file.ReadLine()) != null)
        {

            dataGridView1.Rows.Add();
            string[] words2 = line.Split(',');
            for (var i = 0; i < words2.Length; i++)
            {

                dataGridView1[i, counter].Value = words2[i];


            }
            counter++;
        }

        dataGridView1.Rows.Add();

        file.Close();
```

```
        }


        public void btnSend_Click(object sender, EventArgs e)
        {
            int i0 = dataGridView1.Columns.Count;
            int j0 = dataGridView1.Rows.Count;
            textBox1.ResetText();
            for (int row_index = 0; row_index < j0; row_index++)
            {
                for (int column_index = 0; column_index < i0 - 1; column_index++)
                {
                    if (dataGridView1[column_index, row_index].Value != null)
                    {

                        textBox1.Text += dataGridView1[column_index,
row_index].Value.ToString().Trim(':') + ",";
                    }
                    else
                    {
                        textBox1.Text += " ,";
                    }
                }
                textBox1.Text += "\r\n";
            }
            File.WriteAllText(@"C:\Users\Armin\Documents\Visual Studio
2015\Projects\RobotInterface\RobotInterface\bin\Debug\Positions.csv", textBox1.Text);
            textBox1.ResetText();
        }

        public void dataGridView1_CellContentClick(object sender,
DataGridViewCellEventArgs e)
        {

        }
        private float c1;
        private float c2;
        private float c3;
        private float c4;
        private float c5;
        private float c6;

        private float o1;
        private float o2;
        private float o3;
        private float o4;
        private float o5;
        private float o6;


        private void btnSave_Click(object sender, EventArgs e)
        {



            modbusWriteRegister();
```

```
            }


        void modbusWriteRegister()
        {
            int j = 0;
            int n = dataGridView1.RowCount;

            while (j < n-1)

            {
                //bool reachPos = false;
                using (TcpClient client = new TcpClient())
            {

                    string temp1 = "";
                    string temp2 = "";
                    string temp3 = "";
                    string test1 = "";
                    string test2 = "";
                    string test3 = "";


                    try
                    {
                        test1 = ((FormMain)this.MdiParent).fPosition.DataGrid[1,
j].Value.ToString();
                        test2 = ((FormMain)this.MdiParent).fPosition.DataGrid[2,
j].Value.ToString();
                        test3 = ((FormMain)this.MdiParent).fPosition.DataGrid[3,
j].Value.ToString();

                    }
                    catch
                    {
                        test1 = "null";
                        test2 = "null";
                        test3 = "null";
                    }


                                    if(test1 != " " && test2 != " " && test3 != " " &&
test1 != "" && test2 != "" && test3 != "" && test1 != "null" && test2 != "null" && test3
!= "null")
                    {
                        temp1 = ((FormMain)this.MdiParent).fPosition.DataGrid[1,
j].Value.ToString();
                        temp2 = ((FormMain)this.MdiParent).fPosition.DataGrid[2,
j].Value.ToString();
                        temp3 = ((FormMain)this.MdiParent).fPosition.DataGrid[3,
j].Value.ToString();

                        frmControl.B1 = Convert.ToDouble(temp1);
                        frmControl.B2 = Convert.ToDouble(temp2);
                        frmControl.B3 = Convert.ToDouble(temp3);
```

```
                    frmControl.newPosition = true;
                    frmControl.reachPos = false;

                    Thread.Sleep(500);

                    while (!frmControl.reachPos)
                    {

                    }


                }
                j++;

                frmControl.reachPos = false;
            }


        }

    }

    void modbusReadRegister()
    {
        using (TcpClient client = new TcpClient("127.0.0.1", 502))
        {
            ModbusIpMaster master = ModbusIpMaster.CreateIp(client);

            // read five input values
            ushort startAddress = 100;
            ushort numInputs = 5;
            bool[] inputs = master.ReadInputs(startAddress, numInputs);

            for (int i = 0; i < numInputs; i++)
                Console.WriteLine("Input {0}={1}", startAddress + i, inputs[i] ? 1 :
0);
            client.Close();
        }
    }

    private void frmPosition_FormClosing(object sender, FormClosingEventArgs e)
    {
        int i0 = dataGridView1.Columns.Count;
        int j0 = dataGridView1.Rows.Count;
        textBox1.ResetText();
        File.WriteAllText(@"C:\Users\Armin\Documents\Visual Studio
2015\Projects\RobotInterface\RobotInterface\bin\Debug\Positions.csv", textBox1.Text);
        textBox1.ResetText();
    }


    }
}
```

10 Appendix D

Interface Form Robot Dimension Source Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RobotInterface
{
    public partial class frmRobotDimentions : Form
    {
        private const int WM_NCLBUTTONDBLCLK = 0x00A3;

        public frmRobotDimentions()
        {
            InitializeComponent();
        }


        private void frmRobotDimentions_Load(object sender, EventArgs e)
        {

        }
        protected override void WndProc(ref Message m)
        {
            if (m.Msg == WM_NCLBUTTONDBLCLK)
            {
                // this.RelocatedForm();
                AutoResize();
                m.Result = IntPtr.Zero;
                return;
            }
            base.WndProc(ref m);
        }


        public static double dif;
        public static double die;
        public static double dire;
        public static double dirf;


        private void dimentionLA_ValueChanged(object sender, EventArgs e)
        {
            dirf = (double)dimentionLA.Value;
            //sendDimentions();
        }
        private void dimentionLB_ValueChanged(object sender, EventArgs e)
        {
            dire = (double)dimentionLB.Value;
```

```
            //sendDimentions();
        }
        private void dimentionE_ValueChanged(object sender, EventArgs e)
        {
            die = (double)dimentionE.Value;
            //sendDimentions();
        }

        private void dimentionF_ValueChanged(object sender, EventArgs e)
        {
            dif = (double)dimentionF.Value;
            //sendDimentions();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            frmControl.e = (double)dimentionE.Value;
            frmControl.f = (double)dimentionF.Value;
            frmControl.re = (double)dimentionLB.Value;
            frmControl.rf = (double)dimentionLA.Value;
        }


        public void AutoResize()
        {

            this.Width = (int)(MdiParent.ClientSize.Width * 0.2);
            this.Height = (int)(MdiParent.ClientSize.Height * 0.6);

            int x = (int)(MdiParent.ClientSize.Width * 0.8);
            int y = 0;
            this.Location = new Point(x, y);
        }


    }
}
```