



THE DEVELOPMENT OF ROBOT WORK CELL OPERATIONS AND THE CREATION OF STUDENT EXERCISES

Thesis

**Jarno Auvinen
Miia Jalkanen**

**Degree Programme in Automation Technology
Automation Technology**

Accepted ____ . ____ . ____ _____

SAVONIA-AMMATTIKORKEAKOULU, VARKAUDEN YKSIKKÖ

Koulutusohjelma

Automaatiotekniikan koulutusohjelma

Tekijä

Jarno Auvinen, Miia Jalkanen

Työn nimi

The Development of the Robot Work Cell and the Creation of Student Exercises

Työn laji

Päiväys

Sivumäärä

Insinöörityö

12.4.2010

121 + 91

Työn valvoja

Yrityksen yhdyshenkilö

Yritys

Risto Niemi

Risto Niemi

Savonia-amk

Tiivistelmä

Savonia-ammattikorkeakoulun Varkauden yksikköön hankittiin konenäöllä varustettu robottisolun vuonna 2006. Päätötoita lukuunottamatta sitä ei juurikaan ole käytetty opetuksessa, koska oppilaskäyttöön soveltuvia harjoituksia on ollut hyvin vähän. Lisäksi robotin nivelten paikoitustiedot olivat kadonneet varmistusparistojen loputtua.

Tällä työllä oli kolme tavoitetta. Työn pääasiallisena tavoitteena oli kehittää lisää oppilasharjoituksia robotiikan kursseja varten. Ennen robotin käyttöönottoa sen paristot täytyi vaihtaa ja koordinaatisto uudelleenkalibroida. Kolmantena tavoitteena oli kehittää robottisolun toimintoja monipuolisemmiksi ja ottaa solussa olevat kaksi web-kameraa käyttöön.

Työn aikana asetettiin myös kolme lisätavoitetta. Ensimmäinen oli TicTacToe, perinteisen ristinollapelin robottia vastaan pelattava verkkopeliversio. Ideana oli, että Savonia-ammattikorkeakoulun Varkauden yksikkö voisi käyttää peliä esittelytarkoituksessa. Toinen tavoite oli robottisolun rakenteellinen muutos, joka sisälsi harmaasävykameran siirron hyödyllisempään paikkaan ja lisäakselilla varustetun pöydän korvaamisen uudella työtasolla. Kolmantena tavoitteena oli kehittää robottisolun toimintoja havainnollistava esitys, jota voitaisiin esitellä vierailijoille.

Kaikki asetetut tavoitteet saavutettiin. Paristojen vaihto onnistui. Koordinaatiston uudelleenkalibrointi suoritettiin käyttäen menetelmää, jossa akselit käännettiin mekaanisesti rajoitettuihin ääripisteisiinsä. Oppilasharjoituksia kehitettiin kaikkiaan 20 ja niistä koottiin harjoitusmoniste yhdessä niihin liittyvien ohjeiden kanssa. Robottisolun toimintojen kehittäminen sisälsi rakenteellisen muutoksen ja uutta ohjelmistoa PC:n liittämiseksi robottisolun paikallisesti tai etäohjatuksi. Uusi kiinteästi asennettu työtaso rakennettiin lisäakselilla ohjatun kääntyvän pöydän yläpuolelle ja harmaasävykamera siirrettiin kattoon uuden työtason yläpuolelle. Web-kameroille kirjoitettiin uusi katseluohjelma. TicTacToe-peliä kokeiltiin lähiverkossa ja se todettiin toimivaksi, joskin lisätestaamista vaativaksi. Robottisolulle ohjelmoitiin helppokäyttöinen demonstraatio-ohjelma, joka esittelee robottisolun toimintoja käänneltävään telineeseen sijoitetulta näytöltä.

Avainsanat

Robotiikka, Teollisuusrobotti, Robottisolu, Konenäkö, Ohjelmointi

Luottamuksellisuus

Julkinen

SAVONIA UNIVERSITY OF APPLIED SCIENCES, BUSINESS AND ENGINEERING, VARKAUS

Degree Programme

Automation Technology

Author

Jarno Auvinen, Miia Jalkanen

Title of Project

The Development of the Robot Work Cell and the Creation of Student Exercises

Type of Project

Date

Pages

Final Project

12.4.2010

121 + 91

Academic Supervisor

Company Supervisor

Company

Risto Niemi

Risto Niemi

Savonia UAS

Abstract

Savonia University of Applied Sciences acquired a robot work cell for educational use in 2006. The work cell is equipped with a small articulated arm robot and a machine vision system. Excluding some final projects, the work cell has not been used very much by students because there has been only few suitable exercises. Also, the absolute position encoder data had been lost so the robot has been completely out of use.

This thesis had three distinct goals. First, the robot's position encoder batteries had to be replaced and the coordinate system recalibrated in order to get the robot working. The main object of this project was to create student exercises about the work cell to be used in robotics courses. Additional development of the more versatile operations of the work cell was also desired. It was also aimed at utilizing the web cameras in a better way.

During the study, three additional goals were set. The first new subproject was TicTacToe, the traditional pencil-and-paper game played against the robot on the Internet. The idea was that Savonia UAS could use the game for exhibit purposes. The second project was the structural modification of the work cell. It included relocating of the grayscale camera and replacing of the additional axis with a new worktop. The third additional goal was to create a demonstration to be shown to the visitors.

All the goals were achieved. The batteries were changed successfully. The robot origin data was recorded using the mechanical stopper method. Total of 20 exercises and the necessary instructions were gathered into a handout. The development of the robot cell included both structural modifications and new software for connecting the PC to the work cell in local or remote control purposes. A new, solid worktop was built above the additional axis operated turntable and the grayscale machine vision camera was relocated to the ceiling above the table. A viewing software was written for the web cameras in the work cell. The TicTacToe game was tested in the local network and verified operational. A user-friendly demonstration was programmed to show the features of the robot work cell using a display installed to the display stand.

Keywords

Robotics, Industrial Robot, Robot Work Cell, Machine Vision, Programming

Confidentiality

Public

Foreword

This thesis has been a diverse process with all the programming and structural changes. The planning of the student exercises gave us a profound understanding about robotics and machine vision which we think will be very valuable in the future.

We would like to express our gratitude to the organizations and individuals who have supported and assisted in this study.

First, we would like to thank Savonia University of Applied Sciences for giving us the opportunity to work on the robot work cell. We are most grateful to the supervisor of this work, M. Sc. Risto Niemi, who offered this interesting project to us.

We would also like to thank the laboratory technician Pauli Tuovinen for supplying all the materials when the modifications of the work cell were planned. Also the help of Savo Vocational College is greatly appreciated.

We also appreciate the support of student colleagues, teachers and other personnel of Varkaus Campus.

In Varkaus 12.4.2010

Jarno Auvinen

Miia Jalkanen

Terminology

Accuracy

The difference between the point the robot is trying to achieve and the actual result point. See repeatability.

CCD

Charge-couple device. An analog shift register commonly used for serializing parallel analog signals in photoelectric light sensors (in digital cameras).

CMOS

Complementary metal-oxide-semiconductor. A constructing technology of integrated circuits used in microprocessors, microcontrollers, image sensors etc.

DOF

Degree of freedom. The number of independent position variables needed to be specified in order to locate all parts of the mechanism. The number of different ways in which a robot arm can move. Typical industrial robot is an open kinematic chain and thus, the number of joints equals the number of degrees of freedom.

Dot Matrix

A 2-dimensional array of dots used to represent characters, symbols and images.

DSP

Digital Signal Processor. A specialized microprocessor optimized for needs of digital signal processing.

Ethernet

Family of frame-based computer network technologies for local area networks.

Flash (memory)

Non-volatile memory that can be electronically erased and reprogrammed.

Forward kinematics

Computation of the position and orientation of robot's end effector from the known joint angles.

Inverse kinematics

The process of determining the parameters of a jointed flexible object (a kinematic chain) in order to achieve a desired pose.

Kinematics

The description of the motion of objects without consideration of the causes leading to the motion.

MDI

Multiple Document Interface. A software engineering term that describes an application which has the ability to work on multiple documents simultaneously.

OCR

Optical Character Recognition. Electronic translation of images into machine-editable text.

Payload

Maximum payload is the weight the robot is able to carry at a reduced speed while maintaining the rated precision. The nominal payload is the amount of weight that can be carried at a maximum speed while maintaining the rated precision. Both ratings depend on the size and shape of the payload.

PLC

Programmable Logic Controller. A computer designed for handling multiple inputs and outputs. Used in automation applications.

RAM

Random Access Memory. Computer data storage that allows the data to be accessed in any order (at random).

Reachability

The maximum horizontal distance between the center of the robot base to the end of the wrist.

Repeatability

The system's or mechanism's ability to, when receiving the same control signals, repeat the same motion or to achieve the same points. Unlike (absolute) accuracy, repeatability is the cycle-to-cycle variation of the manipulators position compared to the target point.

Resolution

- a. image resolution. The number of picture elements (pixels) in an image.
- b. resolution in motion. The smallest increment of motion or distance that can be detected or controlled by the control system of a mechanism.

RISC

Reduced Instruction Set Computer. Microprocessor architecture that utilizes a small, highly-optimized set of instructions, rather than a more specialized set of instructions often found in other types of architectures.

SCARA

Selective Compliant Articulated Robot Arm. A robot type with two horizontal joints. Due to its structure, it is slightly compliant in x-y direction but rigid in z direction.

TB

Teaching Pendant. Hand controller used for moving, programming and changing the parameters of a robot.

TCP/IP

Transmission Control Protocol / Internet Protocol. A set of communication protocols used for Internet and other similar networks.

Work envelope

The three-dimensional shape defined by the boundaries the robot is able to reach. Also referred to as reach envelope.

Table of Contents

Foreword.....	I
Terminology.....	II
1 Introduction.....	1
2 Industrial Robotics.....	3
2.1 The History of Robots.....	3
2.2 The Development of Robots.....	5
2.3 Industrial Robot Markets.....	7
2.4 Industrial Robot Types.....	8
2.4.1 Articulated Robot Arm.....	8
2.4.2 Linear Robot (Cartesian Robot).....	9
2.4.3 Cylindrical Robot.....	10
2.4.4 Parallel Robot.....	10
2.4.5 SCARA Robot.....	12
2.4.6 Polar Coordinate Robot.....	13
2.5 Robot Grippers.....	14
2.6 Robot Sensors.....	16
2.6.1 Range Sensors.....	17
2.6.2 Proximity Sensors.....	21
2.6.3 Touch.....	25
2.6.4 Force and Torque.....	29
2.6.5 Machine Vision.....	30
2.7 Robot Control.....	38
2.7.1 Robot Arm Kinematics.....	38
2.7.2 Robot Arm Dynamics.....	40
2.7.3 Robot Manipulator Control Techniques.....	43
2.7.4 Problems in Robot Control.....	45
2.7.5 The Work Environment of a Robot.....	46
2.8 The Robot Work Cell.....	50
2.8.1 The Equipment.....	51
2.8.2 The Software.....	54
3 The Modification and the Exercises.....	56
3.1 Battery Change and Calibration.....	56
3.2 Robot Work Cell Exercises.....	57
3.2.1 The Pedagogical Goals of the Exercises.....	57
3.2.2 The Exercises.....	58
3.3 Accessories.....	63
3.3.1 RobotCamera.....	63
3.3.2 Tool Change Program.....	72
3.3.3 Server Console.....	73
3.4 TicTacToe.....	77
3.4.1 Description of the System.....	78
3.4.2 TicTacToe Data Transfer Protocol.....	79
3.4.3 TicTacToe Server.....	80
3.4.4 TicTacToe Client.....	87
3.5 Modification of the Robot Work Cell.....	93
3.6 Demo of the Robot Work Cell.....	95
3.6.1 Planning and Problems.....	95

3.6.2	Explanation of the Operation.....	97
3.6.3	Slide Show.....	97
3.6.4	Graphics.....	100
3.6.5	Networking	101
3.6.6	Robot Program	101
3.6.7	Hardware Configuration.....	102
4	Results.....	103
4.1	Results of the Main Objectives.....	103
4.1.1	Battery Change and Calibration.....	103
4.1.2	Exercises.....	103
4.1.3	The Web Cameras.....	104
4.2	Results of the Additional Goals.....	105
4.2.1	TicTacToe.....	105
4.2.2	Modification of the Robot Work Cell.....	106
4.2.3	Demo of the Robot Work Cell.....	107
4.3	Development ideas.....	108
5	Conclusions.....	110
6	References.....	111
7	Appendixes.....	115

1 Introduction

Savonia University of Applied Sciences acquired a robot work cell for educational use in 2006. Several theses have been completed with the robot work cell. In two of them, the basic use of the robot work cell is introduced [1], [2]. The projects can be seen as an excellent starting point for the learning of the use of the work cell. However, the work cell has not been used by students in the robotics courses. Using the robot work cell in robotics courses has been complicated because of the lack of student exercises. The studying of the use of all the equipment in the work cell has been seen as too large a challenge.

The study of Mika Korhonen teaches how to start and shut down the work cell, program the robot and how to control the conveyor. The study of Anssi Kauppinen extends the previous by explaining the use of the machine vision system. Together with the things learned from Korhonen's programming exercise, the student gets the needed knowledge to create fully functional applications for the work cell.

After completing these exercises, the student has learned the basics of the robot work cell functions, like moving the arm around the robot work cell and picking items. Real solutions used in industry are not only about moving the robot from a point to another, programs have to optimize the work cycle time, communicate with other devices, handle multiple products with different methods and send statistics to be archived. Thus more exercises needed to be designed.

While familiarizing with the work cell, there were several problematic situations with the devices. When it was searched for the manuals, instructions or notes of the devices, it was noticed that pieces of useful information were scattered all over the laptop computer. The instructions were partly insufficient and vague. There had been some problems in previous project, when the robot accidentally lost its calibration during the installation of the other web camera to the ceiling, and the robot has been out of order since then. [2]

This thesis project had three distinct goals. The main objective was to create more student exercises and instructions about the robot work cell to be used in robotics courses. In order to get the robot working, the dead absolute position data encoder batteries had to be replaced and the lost position data had to be re-recorded by calibrating the robot. This was the second objective. The third objective was to develop the operations of the robot work cell. This included the deployment of the two web cameras. For this, the work cell's PC had to be replaced due to stability problems and lack of computing power required by the live video processing.

During the study, also three additional goals were set. The first started project was TicTacToe, a traditional pencil-and-paper game played against the robot over the Internet. The second additional goal was the structural modification of the robot work cell. Because it was possible to modify the equipment of the robot work cell, a renovation was planned to improve the usability of the work cell. The modification included designing and manufacturing of a larger, solid worktop to replace the additional axis and the mounting of the grey scale machine vision camera to the ceiling above the new worktop. The third additional goal was to develop a demonstration that could be shown to the visitors. The demonstration was thought to be interactive so that anyone passing by could view it without assistance. For this, a slide show explaining the actions was designed.

2 Industrial Robotics

2.1 The History of Robots

The idea of an autonomous human-like robot has existed for thousands of years. As early as in 750 BC, the Iliad of Homer portrays golden mechanical handmaidens, female slaves, made by master craftsman Hephaestus [3]. Probably the first actual mechanical human automaton (a self-operating machine) was designed by Leonardo da Vinci in the 15th century. It is unclear, whether it was actually built during his lifetime. The sketches were discovered in the 1950s and in 2002 the complete physical model was built [4]. The cable-and-pulley-driven "mechanical knight" was able to move its arms and head, to sit down and to open and close its mouth [5]. The word "robot" was introduced by a Czech author Karel Capek in 1921 in his play R.U.R. (Rossum's Universal Robots). The term is derived from the Czech word "robota", which means serf labour [6].

The development of modern robot mechanisms began in the mid-1940s at the Oak Ridge and Argonne National Laboratories, where mechanical master-slave manipulators were designed for handling radioactive material. These systems reproduced the movements of the hand and arm of a human operator. The mechanical coupling was later replaced by electric and hydraulic manipulators. [7]

In the mid-1950s more sophisticated systems quickly replaced master-slave manipulators. These systems were capable of autonomous, repetitive operations. Inspired by Isaac Asimov's science fiction writings, George C. Devol developed a manipulator which was able to follow the instructions of a program and, what's most important, which could be reprogrammed by changing only the software. The further development of this device, "programmable articulated transfer device" as Devol called it, led to the creation of the first industrial robot, Unimate, shown in Figure 1. Devol and Joseph F. Engelberger founded the Unimation, Inc., the first robot manufacturing company. [7], [8]

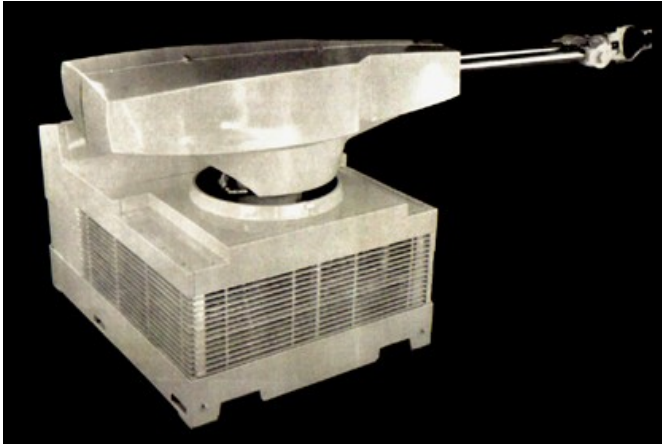


Figure 1: Unimate, the first industrial robot [9]

Unimate was a electronically controlled, programmable lifting arm powered with hydraulics [10]. The program sequence was stored to a magnetic drum which was a ferromagnetically coated cylinder rotating with a constant speed. The drum had magnetic tracks around its circumference and each track had its read and write heads. [11] The robot arm had six programmable axes and it could lift loads as heavy as 200 kilograms [12]. Unimate was first used in 1961 to do die casting and welding duties on a General Motors assembly line [13].

The first successful computer controlled and electrically powered robot arm was Stanford robot arm which was designed by the mechanical engineer Victor Scheinman who was working in the Stanford Artificial Intelligence Lab [14],[15]. The Stanford arm had 6 degrees of freedom. The robot's predecessor, Stanford hydraulic arm, was dangerous and difficult to control and the new Stanford arm was a great improvement providing the compatibility with the existing computer systems and easier controlling. [16] All joints were powered by DC motors. Potentiometers provided feedback of the positions and analog tachometers measured the velocity and electromagnetic brakes locked the joints. To prevent damage in collisions there were slip clutches. Wrist had also a torque sensor to prevent overloading. Because of the low calculation speeds of the first computers, brakes held the joints still while the next trajectory was being calculated. [15] Stanford arm was used for over 20 years in teaching purposes and various projects. These consisted of fairly complicated tasks such as assembling a Ford Model A waterpump, partial assembly work of a chainsaw and solving the Instant Insanity color puzzle. [16]

2.2 The Development of Robots

The early robotic mechanisms were powered either pneumatically or hydraulically. Today, most robots are equipped with electrical motors. [17] These are either servo or stepper motors powered with either AC or DC current. Servo motors are proportionally controlled DC motors in a closed loop - feedback of the servo position is provided back to the servo controller with a potentiometer [18]. Stepper motors are brushless, pulse powered motors which move in discrete steps and do not provide feedback, but they are a cheaper solution compared to servo motors and in slow speeds stepper motors have a higher torque. [19]

From the beginning of the industrial robot revolution in the 1970s, the research of robotics was mainly concentrated on industrial robots until 1990. While the assigned tasks became more complicated, robots had to be more adaptive and flexible. This change brought robots also to the new industries such as food and pharmacy industries. [20] The rapid development of computer systems in the 1980s made it possible to calculate more complex movements. As the semiconductor technology developed further and lowered the prices of microprocessors, it was realized that robot markets could be profitable also in other sectors than industry. [21],[22]

Today, industrial robot controllers are small and smart real time process computers capable of controlling the workcell equipment thousands of times per second and reacting in milliseconds. Multiple programs can also be executed in parallel. Industrial robots can lift payloads from a few kilograms to several hundred kilograms. Also precision has increased greatly, almost all robots can reach the precision of ± 1 mm and the industrial robots used in assembly tasks have to be able to reach a precision of $\pm 0.05 - 0.1$ mm. [23]

As Unimate was the first robot used in a dangerous environment in 1961, industrial robots are now commonly replacing workers in tasks that are either too dangerous, dull or difficult for humans [10],[20]. They are used in a variety of tasks ranging from light assembly work in the electronics industry to heavy-duty work in the metal industry. In the past 30 years, automotive lines have been fully automated and taken over by robots

[20]. In laboratories, robots are identifying, measuring and preparing samples, only requiring a little of staff attention [24]. A common thing for all robot applications is the need to optimize the productivity, when human workers are not fast or accurate enough [20].

As a result of increasing numbers of utilizations, different types of robots exist today beside the stationary industrial robot. Mobile robots use mechanisms that allow them to move freely around a real world terrain. Mobile robots are developed to increase the range of the work area. Telerobots are mobile machines that follow instructions from a human operator in a remote location. [20]

2.3 Industrial Robot Markets [25]

Somewhat surprisingly, there are only a little over one million industrial robots in the world. In 2008, the number of shipped industrial robots was about 113 300 units worldwide. During the recent years, there have been dramatic differences between the regions of the world. In Asia, the number of supplied units has increased, in Europe it has stagnated and in America it has decreased. In China, where the robot investments are booming, the number of units supplied increased by 20 % from the previous year being 7 900 units. China is the third largest robot market in Asia after Japan (33 100 units) and the Republic of Korea (11 600 units). In the Republic of Korea, the rise was 28 % and in Taiwan 40 %. In other Asian markets, including Indonesia, Malaysia, the Philippines, Singapore, Thailand and Vietnam, the sales rose by 10 %.

The global financing crisis has had a strong influence on the automotive industry in the United States. The cutting and relocating of the production capabilities and the restraining of foreign investments lead to a 12 % decrease in the supply of industrial robots in the Americas.

In Europe, the total number of units supplied stagnated at about 35 100 units. In Italy, France and United Kingdom, the supply decreased but on the other hand, the sales to Central and Eastern Europe – remarkably to Hungary, Poland and Slovakia - went up by 22 %. In Germany, 4 % more robots were sold than in 2007. In the first half of 2008, almost all industries invested in robots, but the start of the economic crisis dropped the sales dramatically. The supplies increased in food and beverage, metal products and machinery industries and decreased in automotive and electrical/electronics industries, as well as in rubber and plastics industry.

At the end of 2008, the number of operational industrial robots in the world was estimated to be in the range between 1 036 000 and 1 300 000 units. The lower number is based on the assumption that the average service life of an industrial robot is 12 years. The latter number is based on the assumption that the average service life might be as long as 15 years.

2.4 Industrial Robot Types

Industrial robots are divided into six main types: articulated, linear, cylindrical, parallel, SCARA and polar. These types are introduced in the following chapters. Different robot types have been developed to fit more specific tasks. The maximum payloads, reachabilities, numbers of axes and maximum movement speeds are varying depending on the robot type and manufacturer. [26]

2.4.1 Articulated Robot Arm

An articulated arm robot, also called a jointed-arm robot, uses rotary joints typically arranged in a chain, where one joint supports the next in the chain. The arm of a vertically articulated robot is connected to the base [26]. The arms typically have 6 degrees of freedom. They are the x, y and z axes and yaw, pitch and roll. Pitch means the wrist's movement up and down. Yaw is the hand's movement left and right and roll is the rotation of the whole fore arm. [17] The work envelope of an articulated robot arm is a partial sphere as shown in Figure 2.

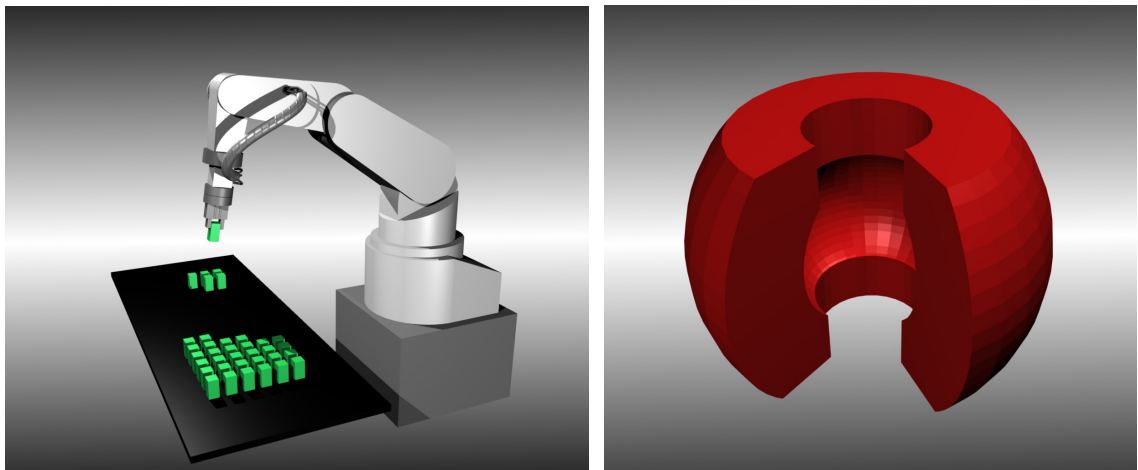


Figure 2: Articulated robot arm and its work envelope

The parts of a robot arm are named base, shoulder, upper arm, forearm and wrist just as in a human arm. In a 6-axis type robot, there is also an elbow block between the upper arm and the forearm. The end effector of a robot arm is often referred to as a hand or tool. The end effector's installation surface is referred to as mechanical interface. [7]

2.4.2 Linear Robot (Cartesian Robot)

The movements of a linear robot are limited to x, y and z axes. Typically the robot is mounted on a linear track supported by pillars over the work area. The track is the robot's axis 1, while axis 2 is another linear drive perpendicular to the axis 1. See figure 3. The up and down motion is provided by the axis 3.

Linear robots are ideally suited for handling and palletizing applications, since they provide a more costefficient solution than articulated robots when only simple manipulation is required. Another advantage is that linear robots can work over a large area. Linear robots can also be combined with articulated robots to produce combined kinematics. [27]

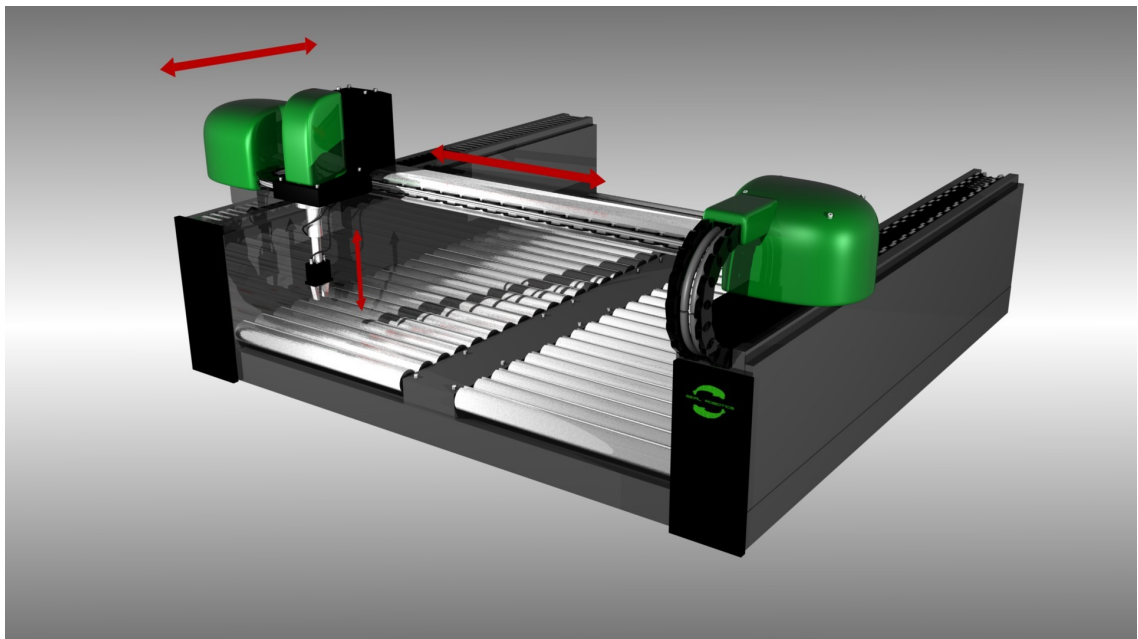


Figure 3: Linear robot

2.4.3 Cylindrical Robot

As the linear robot, also the cylindrical arm (Figure 4) has three degrees of freedom, but only two of them, the Y and the Z axes, are linear. The third is the rotation at the base. The work envelope of a cylindrical robot is cylinder-shaped. [17]

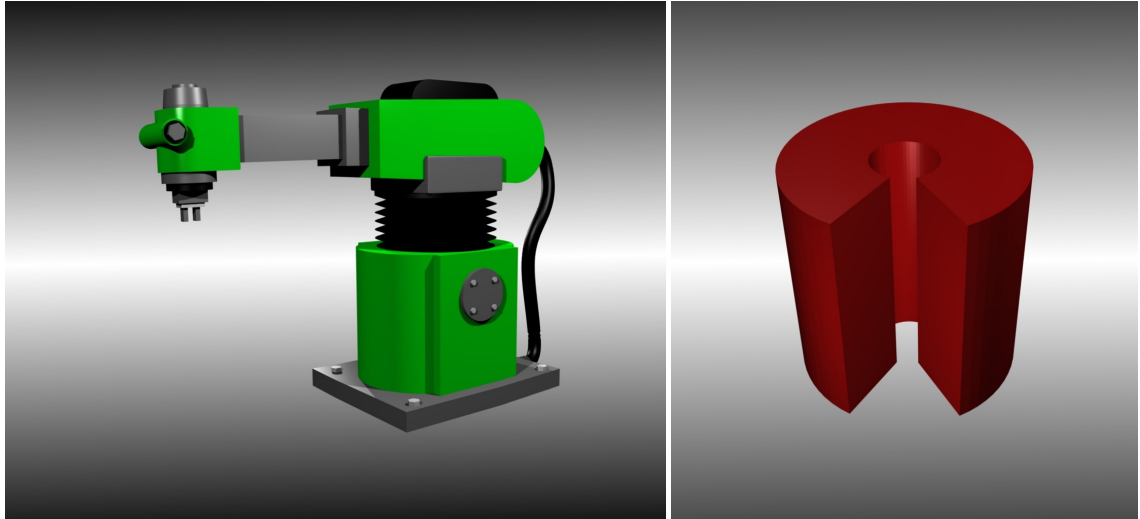


Figure 4: Cylindrical robot and its work envelope

2.4.4 Parallel Robot

Parallel robot, known also as parallel manipulator, consists of a fixed base, an end effector and multiple “legs” as shown in Figure 5. The term parallel means that there are more than one kinematic chain from the base to the end effector. A parallel robot has four degrees of freedom, three of them translational and one rotational. The use of parallelograms forces the end platform's movements to the x, y and z axes. The fourth degree of freedom is the end effector's rotational movement. The parallel robot is mounted above the workspace.

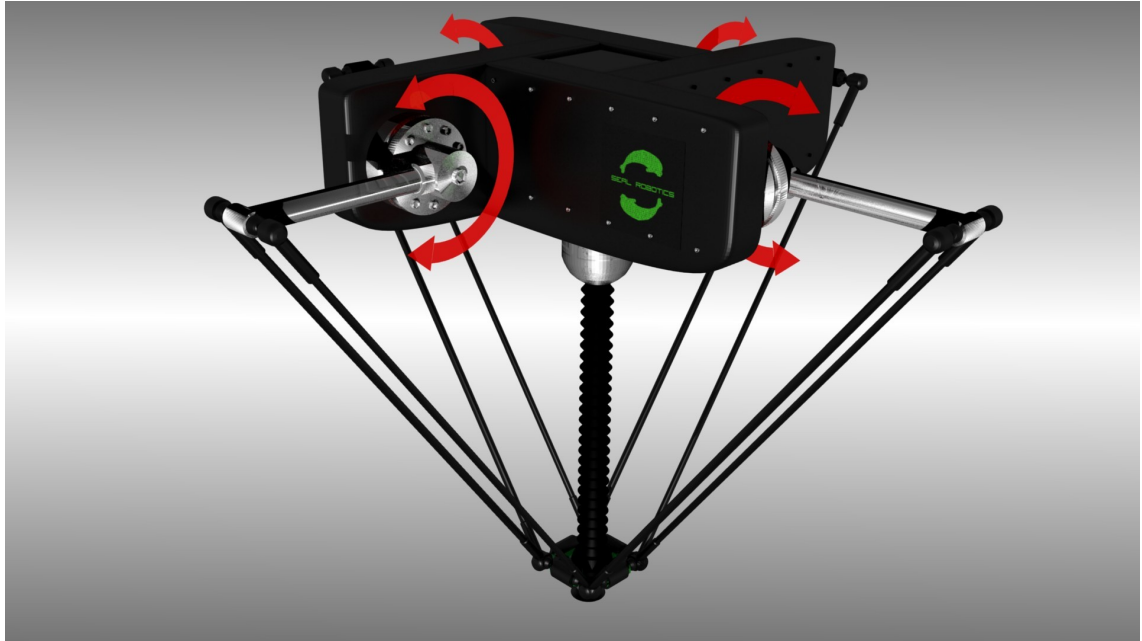


Figure 5: Parallel robot

The design of a delta robot makes it possible to use extremely high accelerations. The moving arms can be made of lightweight composite materials because the actuators are located in the base of the robot. The actuators can be either linear or rotational. Parallel robots are stronger than serial robots (e.g. articulated arm) because the end effector is connected to multiple links. This also makes the accuracy of a parallel robot better when compared to serial robots. In a serial robot, the total error of position is the sum of the errors. In parallel robot the error is averaged and thus smaller. [28],[29]

Of course, there are also some weaknesses in the design. The workspace of a parallel robot is more limited because its legs can collide. This makes them somewhat unable to reach around obstacles. The calculations needed for the forward kinematics used in parallel robots are usually more complex. [29]

2.4.5 SCARA Robot

The acronym SCARA stands for Selective Compliant Articulated Robot Arm. It is also called a horizontal articulated arm robot or horizontal multi-joint robot. The concept of the SCARA robot was introduced in 1981 by Sankyo Senki, Pentel and NEC. As the name implies, the robot has compliance only in some directions (x and y directions) and high rigidity in some (z) direction. There are two types of SCARA designs. One type rotates round all three axes, the other has a sliding motion along one axis. The SCARA design enables very high speeds in horizontal movement. SCARA robots are mainly used in assembly and pick & place solutions. The work envelope of a SCARA robot is in the shape of a cylinder similar to a cylindrical robot's but with round ends as shown in Figure 6.

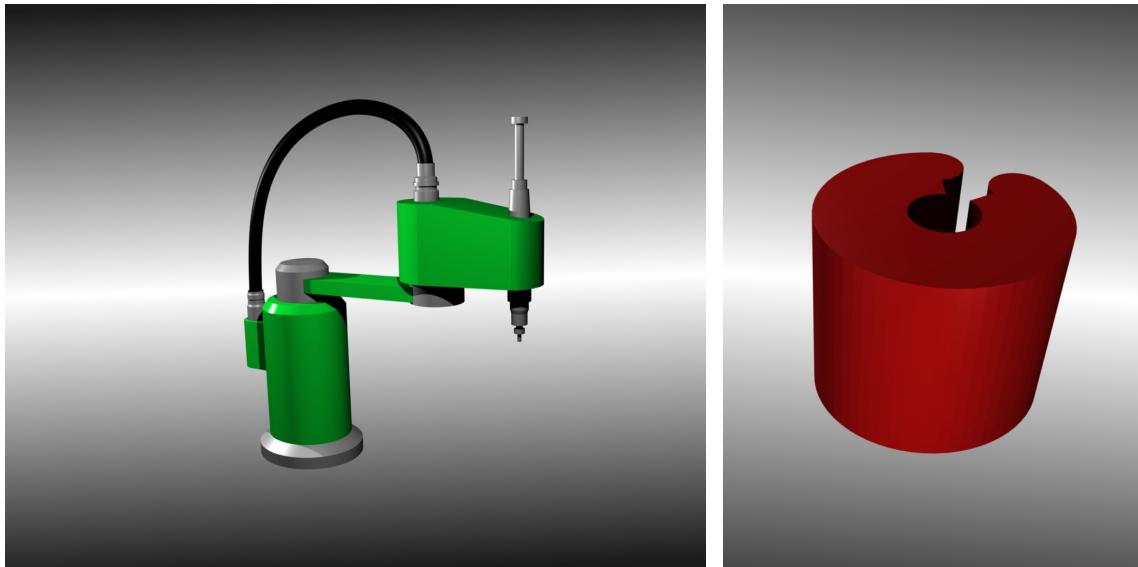


Figure 6: SCARA robot and its work envelope

2.4.6 Polar Coordinate Robot

The polar coordinate robot, also known as a spherical robot, has one sliding and two rotational motions as shown in Figure 7. The first rotation is around the vertical post and the other around the shoulder joint. The work envelope of a polar robot is a partial sphere with varying radius. [17]

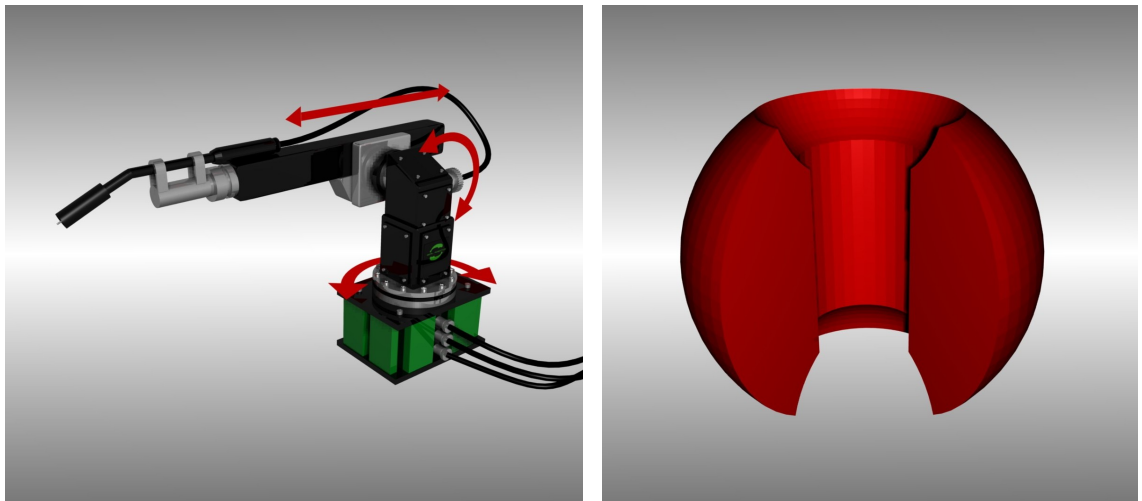


Figure 7: Polar coordinate robot and its work envelope

2.5 Robot Grippers

The main idea of the robots was to develop a multipurpose, adaptive machine that can be applied to different tasks in industry. This was hard, mostly because of the fact that robot grippers did not have the same versatile and flexible abilities as human hands. To make the industrial robots more effective in the tasks which were normally carried out by humans, a more effective handling equipment had to be planned. The grippers have always been designed for individual cases. Most of the grippers have only specific uses such as loading of manufacturing lines, packaging in storages and and object handling in laboratory test and inspection systems. [31] This is because the multipurpose hand with multiple fingers and joints would just be too complicated to be used with the limited sensing abilities [22]. Grippers have been mimicking different postures of human hand depending on the needed functions and today, there is a great amount of different kinds of grippers [31].

Almost all grippers operate in the contact with the object's surface and the solid grip is the most important subject to be considered while designing a robot gripper. This can be achieved by maximizing the contact area and by applying effective gripping force. By matching gripper and object profiles, the best retention stability can be achieved, but this is not always the most practical solution. The most common geometric shapes are commonly utilized in gripper designs to create simple and working grippers with the most effective contact points. Contact points are the areas where the gripping force is centered. There are several types of contacting methods depending on the object's material and shape characteristics. Impactive, ingressive, contiguous and astrictive are the classified gripping methods. [31]

Impactive contact is using the grasping force and object's friction to get the grip. The servo powered and pneumatic jaws are impactive, and one of the most used gripper types. These grippers have usually two gripper fingers to grab the workpiece, but there are also grippers with three fingers to get a tighter hold. Jaw grippers can be designed to grab multiple kinds of items by a creative planning of the fingers and contact points. They can also center the workpiece when prehensing. This can greatly reduce the time of work cycle, when there is no need to center the object mechanically. [31]

Ingressive grippers are operating by deformatting or even penetrating the object's surface. These grippers can be brush elements, hooks or needles. Ingressive grippers can be used in food processing plants or other facilities, where products can be handled more roughly. [31]

Contigutive contact is based on chemical adhesion, surface tension forces or thermal adhesion. Object is grabbed by attaching it to gripper surface and removed mechanically. These grippers are especially useful when objects are irregularly shaped and have different contact points. [31]

Astrictive grippers use the electrostatic adhesion, magnetic forces or vacuum suction to achieve the robust grip. Magnetic grippers can only be used with magnetic materials and their lifting force depends on the object's material, shape and surface topography. [31] Vacuum grippers are widely used in industry. The advantages are that rubber or plastic suction cups won't scratch the surface and vacuum grippers can lift products by using only one flat surface of the object. [23]

While developing a robot gripper, some issues must be considered with care. The most important properties to be noticed are the robot's maximum payload, the used gripping method, tolerance analysis, the space needed for the gripper while functioning, and maintenance situations. The most desirable properties are usually a simple structure, small size and weight, the reliability of the grip and the gripper's centering abilities. The maximum lift capability of the robot is often limited and payload is calculated without the used tool, therefore a gripper's own weight must be considered while planning the tool for the operation. The gripping method should also be chosen with care, vacuum grippers are often better than jaws, as vacuum gripper needs to grasp only one side of the workpiece. Safety aspects are also primary when planning a gripper system. In case of malfunction, the gripper should fail safely. The wrist mechanism torques should be actively monitored and the gripper should be secured with a mechanical safety pin in case of malfunction or break caused by overload. Vacuum grippers have sudden pressure leakage if one suction cup is detached. For this situation there are often safety devices. [22]

2.6 Robot Sensors

The early robots were simple manipulators without any sensing capabilities. In moving, autonomous robots, such as service and surveillance robots, sensory feedback is crucial, but more and more sensors are added to industrial robots too, because they greatly improve the adaptivity of the robot's operation. When the robot is able to adapt to one's surroundings it is easier to set up the production. Especially machine vision adds a great deal of flexibility. In a typical robot and machine vision application, picking pieces from a conveyor, work pieces can be inaccurately positioned and the robot can still pick them. The sensors are classified to internal and external sensors.

Internal sensors are either analogical or digital and their function is to gather data from the robot itself. Positioning and movement speeds are collected with internal sensors. [7]

External sensors are used to gather information from the environment. External sensors can be divided further into two classes: contact or non-contact sensors. The former respond to different types of physical contact, such as touch, slip or force and torque. The latter detect the variation in either acoustic or electromagnetic radiation. They are used to measure range, proximity and visual properties of an object. Different external sensor types are discussed in more detail in the following chapters. [7]

2.6.1 Range Sensors

Range sensors are used to measure distances from reference points. Usually these reference points are range sensors itself. There are two ways to measure distances. The first one is based on optical measurements - the distances are calculated from acquired images with a known unit scale. Usually machine vision cameras are inspecting planes and calculating width and height, but also depth can be measured using stereo vision. The other way to find out range is based on time-of-flight concept. [7]

2.6.1.1 Stereo Vision

Stereo vision is a similar technique to the human eye. A specified point of interest is viewed from two views and the distance estimation is based on the differences in the two images. There are a few preconditions in correct stereo vision measuring: the cameras must be similar, the distance between centers of the lenses, referred to as *baseline*, must be known, and the cameras must be aligned side to side. Then images can be compared to find corresponding points which to use in calculating the depth. Stereo vision is not used to measure very long distances, because the images become more identical when the object is moved further. [7] The camera setup of a stereo vision system is illustrated in Figure 8 and the difference between the images in Figure 9.

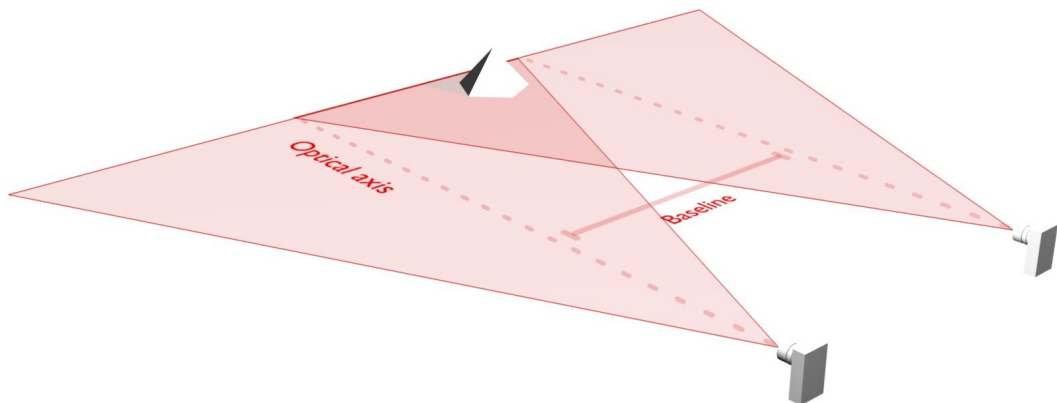


Figure 8: Camera setup of a stereo vision system

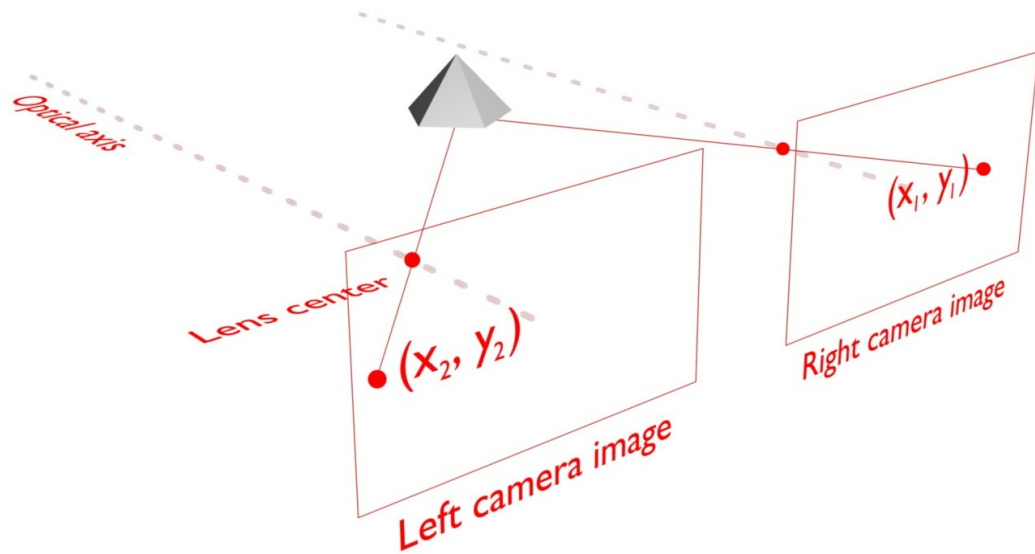


Figure 9: The difference between the stereo vision images

2.6.1.2 Triangulation

Triangulation is one of the simplest ways to measure range. The object being inspected is illuminated with a narrow light beam which is swept over the surface. When the light source is turned into a correct angle, light reflects to the detector, and the distance from the object to the baseline can be calculated geometrically when the angle and distance between the emitter and the sensor are known. The setup of a triangulation system is illustrated in Figure 10 produces a point measurement. It is also possible to obtain three-dimensional coordinates by moving the source and the detector together along a plane as seen in Figure 11. This produces a set of points with a known distance to the detector.

[7]

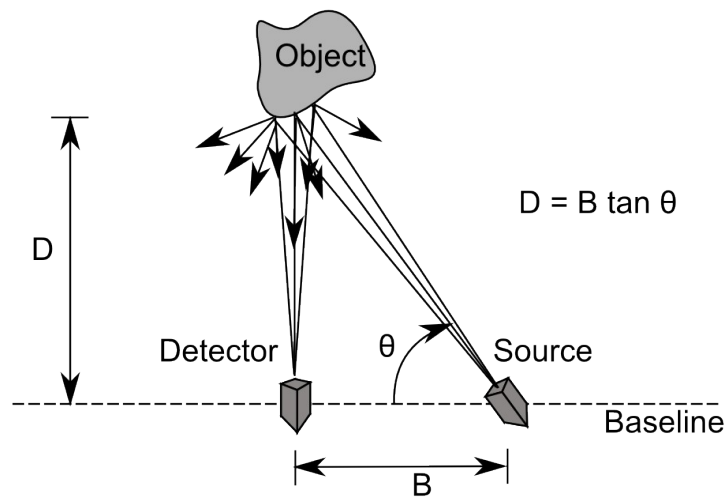


Figure 10: Triangulation

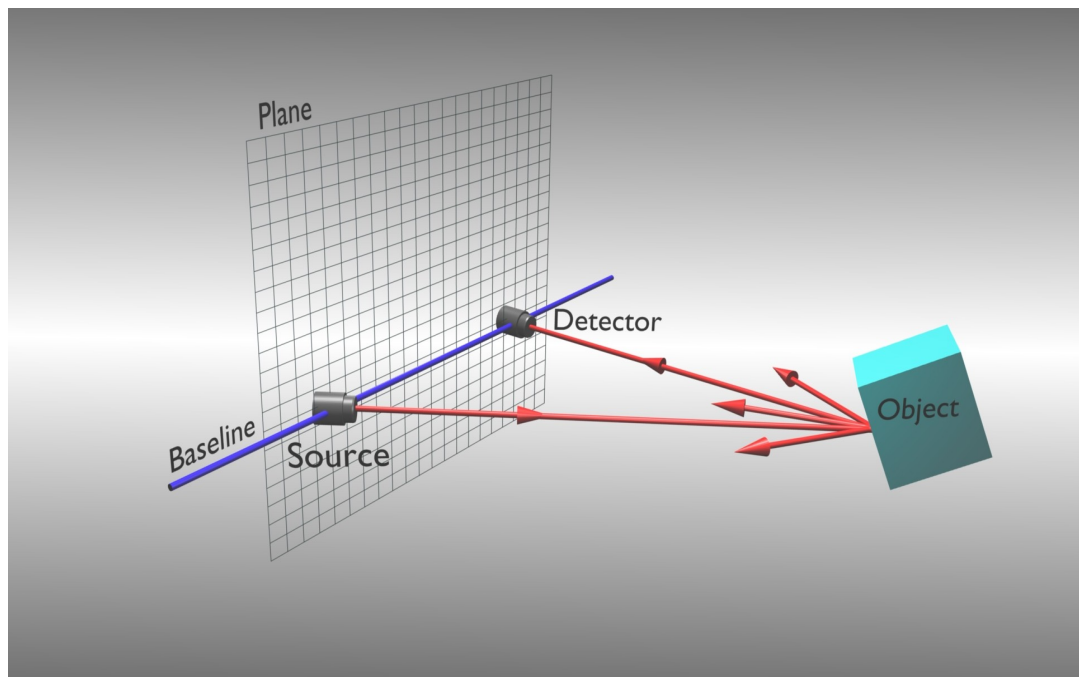


Figure 11: 3D triangulation

2.6.1.3 Time-of-Flight

Probably the most common way of measuring range in robotic applications is the time of flight concept. There are three methods, of which two utilize laser and one ultrasound. When either pulses of light rays or ultrasonic waves are emitted, obstacles reflect some of them back and when the speed of light or sound is known, the distance can be calculated.

In the first laser-utilizing method, the measured quantity is the time. A pulse of light is emitted and reflected back to the same direction by a reflective surface. The time between the outgoing and returning light is measured and as the speed of light is known, the distance can be calculated. This method sets high demands for the instruments used, because of the speed of light.

The another approach is to measure the phase shift between the outgoing and returning beams. Instead of a laser pulse used in the previous method, this method utilizes a continuous beam of light. In this scheme shown in Figure 12, a laser beam is split into two beams. The reference beam travels to the phase measuring device straight after splitting. The other beam travels to the reflecting surface and back. An increase in the distance D increases the phase shift. Because of the small wavelength of laser, the method is not practical for robotic applications. If the laser light is amplitude modulated with a sine signal with a much higher wavelength, the wavelength range is more practical to work with.

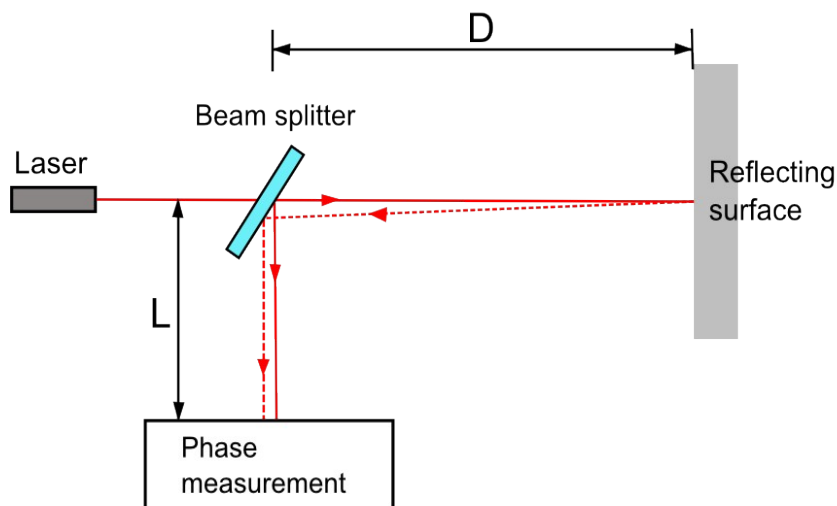


Figure 12: Range measurement by phase shift

The basic idea of ultrasonic range measuring is the same as in the pulsed laser method. A short chirp of ultrasonic sound is transmitted towards the range to be measured. The time between the outgoing pulse and its echo is measured and the distance calculated using the known speed of sound. [7]

2.6.2 Proximity Sensors

Proximity sensors are used to detect the presence of nearby objects without physical contact [32]. These sensors are widely used, when limit switches are repeatedly exposed to mechanical wearing. There are inductive, capacitive and optical proximity sensors. Inductive and capacitive proximity sensors have also an advantage compared to optical proximity sensors – they are unaffected by dust or opaque containers. [33] The different types of proximity sensors are discussed in the following chapters.

2.6.2.1 Inductive

Inductive sensors are one of the most common proximity sensors used in industry. Inductive proximity sensing is based on the change in inductance when a metallic object is present.

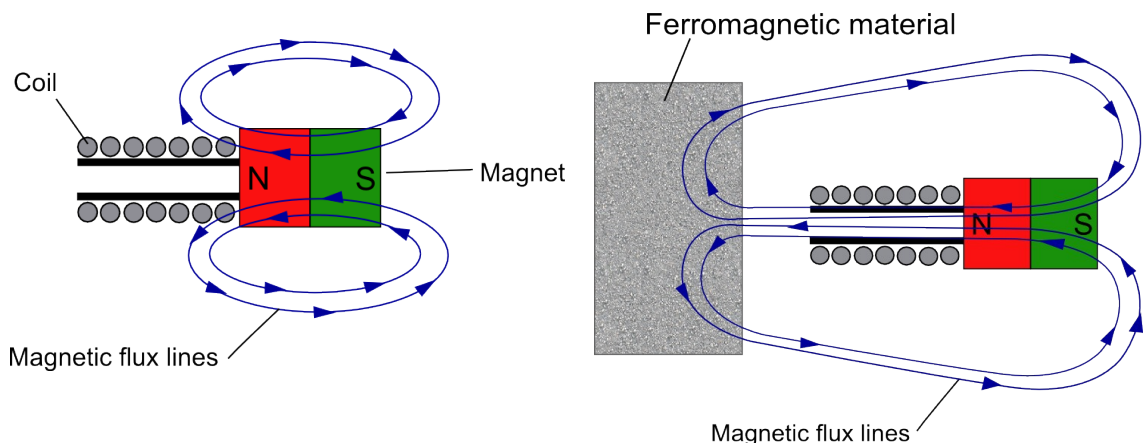


Figure 13: *The principle of an inductive proximity sensor*

An inductive sensor used for ferromagnetic metals consists of a coil and a permanent magnet as shown in Figure 13. When a ferromagnetic object is brought close to the sensor, the flux lines in the permanent magnet change. This induces a current into the coil. Under static conditions (i.e. the object stays still) there is no change in the flux lines and no current. If the object enters or leaves the the field of the magnet, the flux lines change and induce a current pulse. The amplitude and the shape of the current pulse are proportional to the change in the flux. As the Figure 14 shows, the amplitude of the voltage is proportional to the speed of the object and the polarity of the voltage depends on whether the object enters or leaves the field. [7]

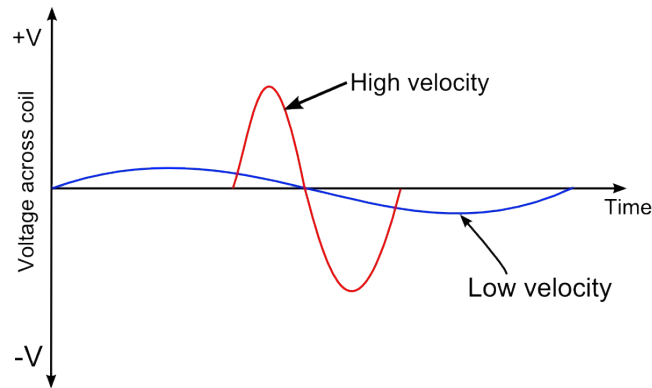


Figure 14: *The amplitude of the induced voltage depends of the object speed*

Inductive sensors can also be used to sense the proximity of non-ferromagnetic metals. The sensor utilizes a LC oscillator circuit which consists of an inductor and a capacitor. Current alternates between the inductor and the capacitor at the resonant frequency of the circuit. The coil produces a high-frequency electromagnetic field directed towards the measuring surface of the sensor. When a metallic object is brought to a close proximity from the sensor, eddy currents appear in the object. They absorb energy from the oscillator circuit and thus reduce the amplitude of the oscillation in the circuit. This loss is converted into a signal. [34],[35]

2.6.2.2 Capacitive

Capacitive proximity sensors react to the change in the capacitance that is induced by an object brought near it. In principle, capacitive sensors detect all solid and liquid materials. [36] A capacitive sensor consists of two electrodes separated by a dielectric material [7].

There are different types of circuitry used to detect proximity from the change in capacitance. In one approach, the capacitor is a part of an oscillator circuit. The oscillation starts when the capacitance exceeds a threshold value. The start of the oscillation indicates the presence of an object as binary value that depends on the threshold value. [7]

In another approach, the capacitor is a part of a circuit which is driven by a sinusoidal waveform. A change in the capacitance causes a phase shift which is proportional to the change in the capacitance. Also these sensors are typically used in a binary mode. [7]

2.6.2.3 The Hall-Effect

When a conductor carries a current in a magnetic field that is perpendicular with the conductor, a potential difference is generated between the opposite edges of the conductor. This is called the Hall-effect. On its own, a Hall-effect sensor can only sense magnetized objects, but in combination with a permanent magnet, they can detect all ferromagnetic objects. [34], [7]

Proximity sensors based on the Hall-effect consist of a Hall-effect sensor and a permanent magnet as shown in Figure 15. When no ferromagnetic object is present, the Hall-effect sensor senses a strong magnetic field. When an object made of ferromagnetic material is brought close to the sensor, the magnetic field lines bend through the material. This causes the magnetic field at the sensor to weaken. [7]

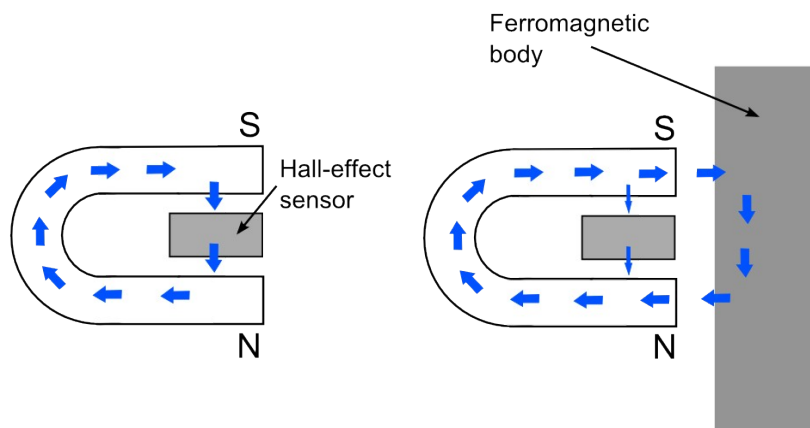


Figure 15: A Hall-effect sensor and a permanent magnet combination for sensing proximity of a ferromagnetic metal

2.6.2.4 Optical

A common type of optical proximity sensor is an infrared LED transmitter and a photodiode receiver combination [37]. The source and detector are focused on the same plane. The intersection of their light cones is the volume, in which the sensor detects an object. [7]

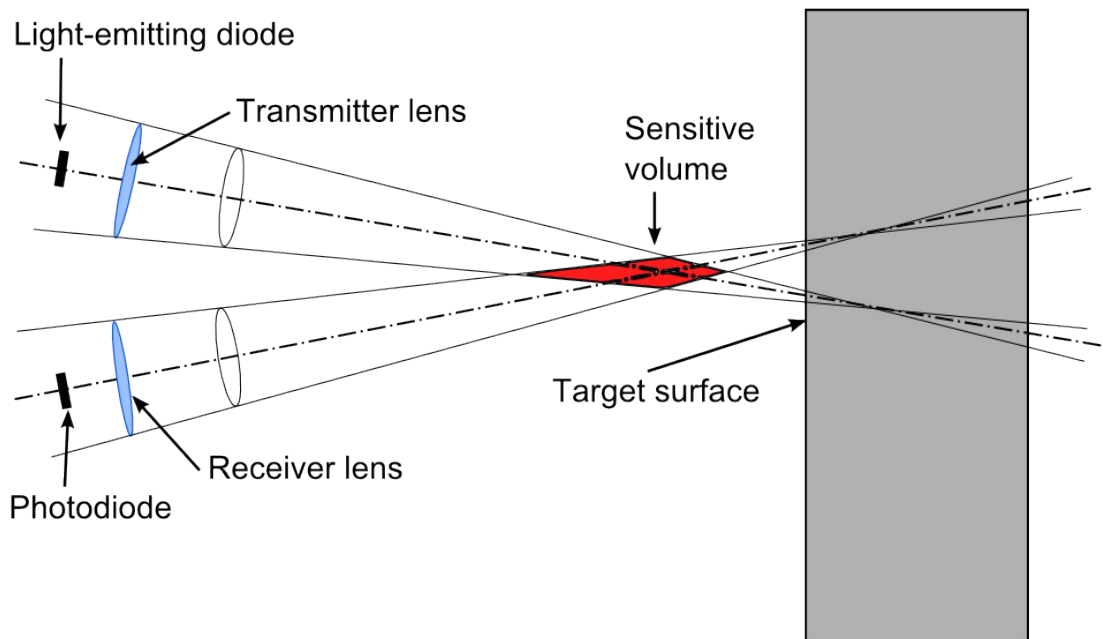


Figure 16: *The principle of an optical proximity sensor*

As seen in Figure 16, the shape and size of the sensing volume vary depending on the location and orientation of the transmitter and receiver. A reflective surface in anywhere inside the sensing volume produces a reading. The intensity of these readings depends on the distance, reflectivity and the orientation of the object. Usually optical sensors are used to generate a binary signal when a preset threshold value is reached. [7]

2.6.3 Touch

Touch sensors are used to obtain information about the contact of the manipulator and objects in the robot's workspace. Touch sensors can be divided into two groups: binary and analog. Binary sensors are switches that respond to the presence of an object. The output signal of an analog touch sensor is proportional to the force. [7]

2.6.3.1 Binary

As mentioned above, binary touch sensors are typically micro switches. As illustrated in Figure 17, such switches can be located in the inner surface of a robot hand's fingers to determine the presence of an object. Using an array of binary sensors on a surface provides more tactile information. [7]

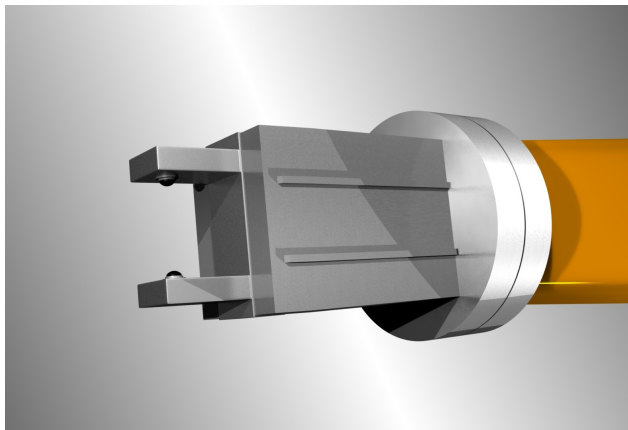


Figure 17: Binary touch sensors on the inner surfaces of a robot hand's fingers

2.6.3.2 Analog

The output of an analog touch sensor is proportional to the force being applied to the sensor. The simplest approach, shown in Figure 18, is built from a rotating shaft that is turned by the movement of a spring-loaded rod pushed by the object. The rotation of the shaft is measured using a potentiometer or a code wheel. The force can be calculated from the displacement when the spring constant is known. [7]

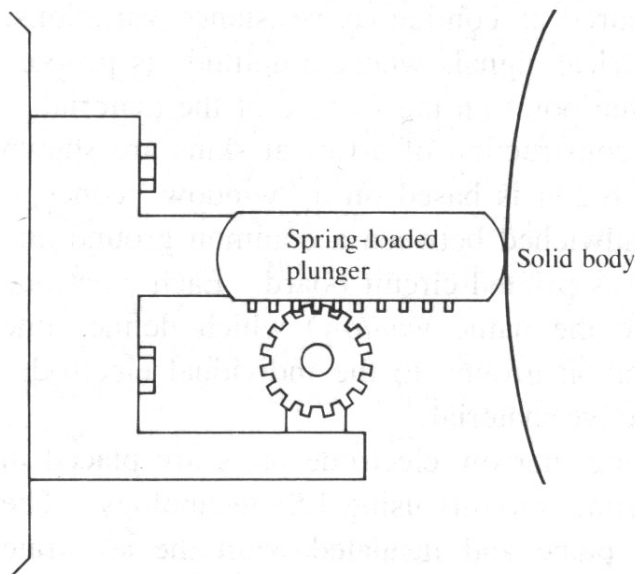


Figure 18: Analog touch sensor [7]

2.6.3.3 Artificial Skin [7]

Artificial skin can be classified as a type of an analog touch sensor, but because the principle behind the device is different from a tactile sensing array formed from multiple individual sensors, it is introduced in this separate chapter. The term “artificial skin” refers to a device, in which a compression causes a deformation in the surface that can be measured as a variation in resistance. There are many approaches to this, four of them shown in Figure 19.

In the first method, shown in Figure 19 (a), conductive material is sandwiched between common ground and an array of electrodes. One touch point is defined by a rectangular area including an electrode. The current flowing from the common ground to the electrodes is a function of compression of the conductive material.

In the second approach, shown in Figure 19 (b), pairs of narrow electrodes are placed on a substrate with electronic circuits. The plane is covered with a conductive material which is insulated from the substrate plain, but not the electrodes. Compression results a resistance change that is measured by the circuits between the electrodes.

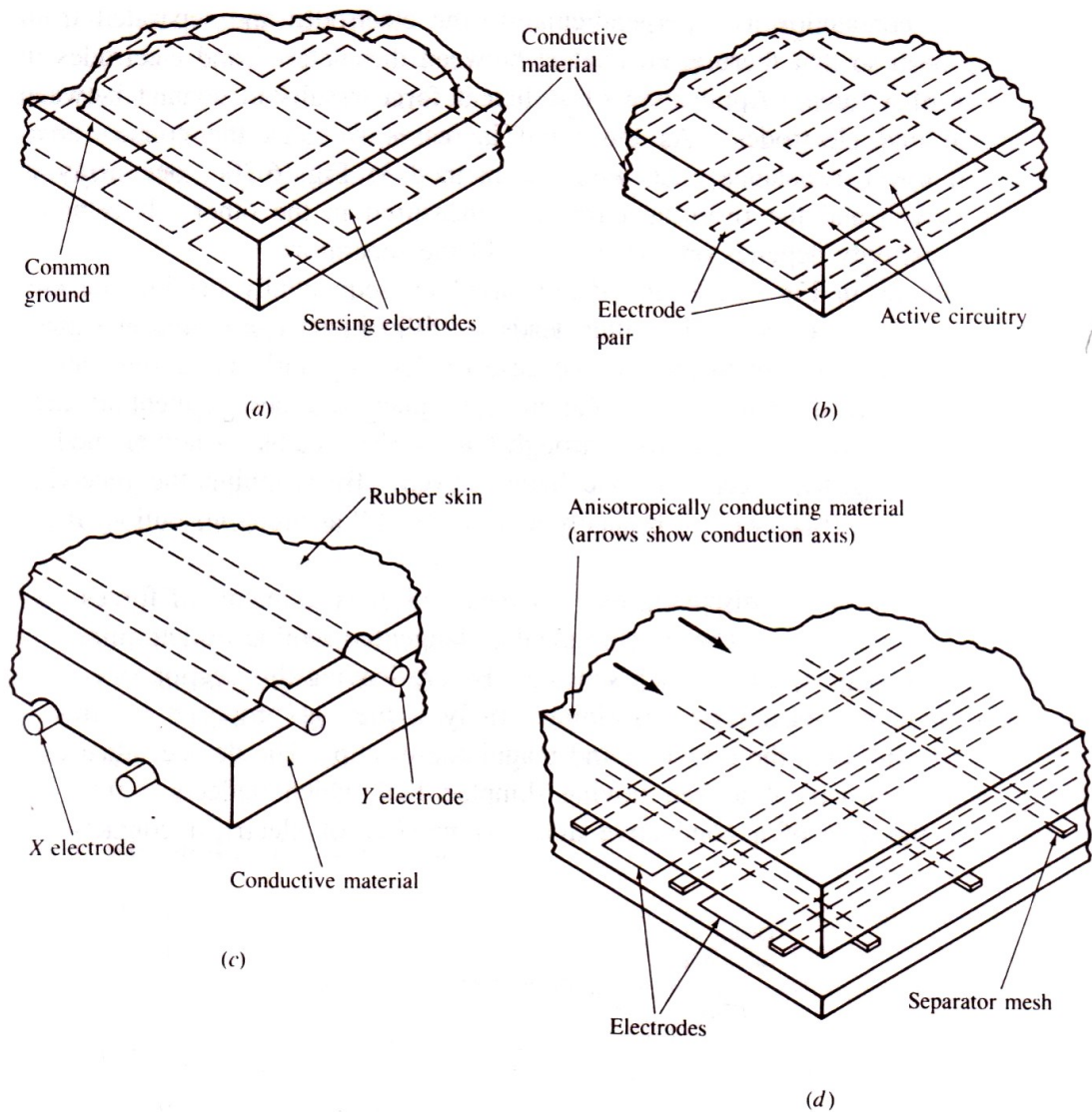


Figure 19: Types of artificial skin [7]

The third technique employs two perpendicular arrays of electrodes with a layer of conductive material between them (See 19 (c)). Each intersection forms a sensing point. The electrodes of one array are driven electronically one at a time and simultaneously measuring the current flowing in the other array. The magnitude of the current is proportional to the compression of the material between the electrodes.

The fourth method utilizes an anisotropic material. Anisotropic material is electronically conductive only in one direction. The sensor (Figure 19 (d)) consists of a linear array of thin, flat electrodes and a layer of conductive material on top of them with the conduction axis perpendicular to the electrodes. The material and the electrodes are separated from each other by a mesh. When a force is applied to the surface, the material and the electrodes come in contact. The size of the contact area increases with the force thus lowering the resistance. One array at a time is driven externally (as in the previous method) and the current is measured in the other.

2.6.3.4 Slipping

All the touch sensors described above measure forces that are perpendicular to the surface of the sensor. However, measuring tangential motion is an essential part of touch sensing. Slipping can be measured with a device illustrated in Figure 20. The device consists of a dimpled ball deflecting a rod mounted on the axis of a conducting disk. When the ball rotates, the disk comes in contact with the electrical contacts under the disk. The contacts are evenly spaced around the circumference of the disk, therefore the direction of the movement can be determined by which contact the disk touches. The frequency of the vibration of the disk is proportional to the ball's speed. [7]

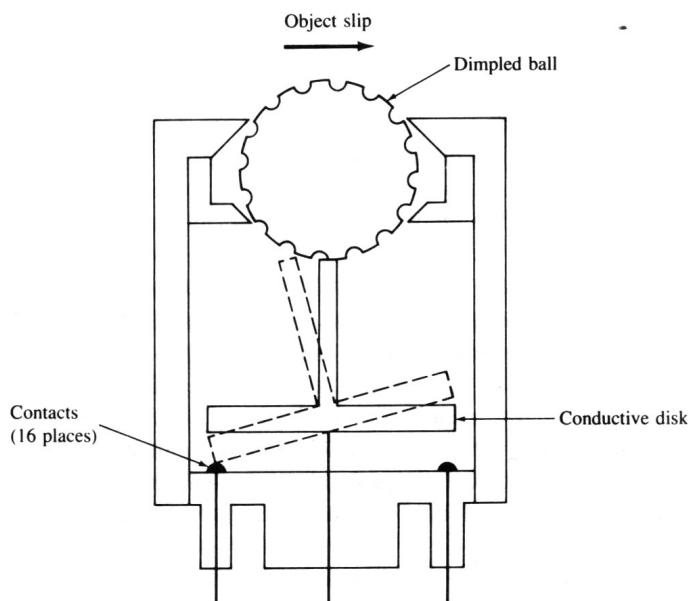


Figure 20: A sensor structure that is able to sense both the direction and the magnitude of slip [7]

2.6.4 Force and Torque

Force and torque sensing is used to measure the reacting forces between mechanical structures while moving the robot. The sensors are critical in applications such as assembly, product testing and material handling to get feedback from the robot's actions. [38] The two approaches to sense force and torque are joint and wrist sensors. These measurements are also necessary, when movements of the robot are actively monitored to detect collisions. [7]

2.6.4.1 Joint Sensors

It is a fault sensitive way to measure force from joints. When forces for each joints are measured and converted to torques, force and torque at hand level must be calculated [39]. Still, it is easy to measure the torque of DC motor driven robot joints, as sensing of torque can be done by measuring the armature current [7].

2.6.4.2 Wrist Sensors

Wrist sensors are mounted between the top of a robot arm and the end-effector. Wrist sensors use eight pairs of semiconductor strain gauges mounted differentially in the deflection bars as shown in Figure 21. Three components of force and moment can be determined by adding and subtracting the strain gauge voltages. [7]

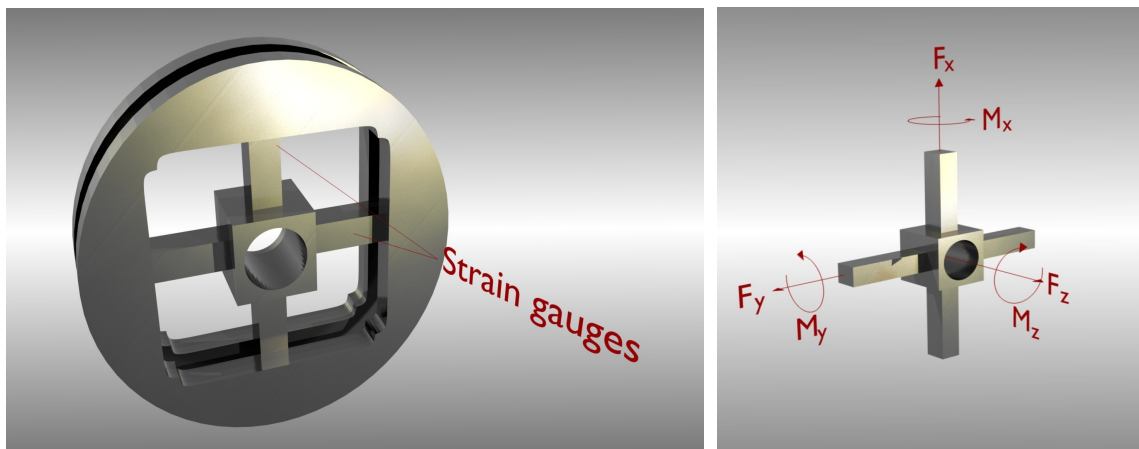


Figure 21: Wrist force sensor and the forces and moments of the strain gauge

2.6.5 Machine Vision

2.6.5.1 Vision, Computer Vision and Machine Vision

Visual perception means the ability to interpret information received from visible light. Vision is the most important sense for humans, and thus the natural choice for creating intelligence to machines, too. However, it has become clear, that the complexity of biological vision developed during the millions of years of evolution is something that cannot easily be imitated.

Computer vision is a subfield of artificial intelligence. It is a scientific discipline dealing with a variety of fields, for example pattern recognition, statistical analysis, image processing and projective geometry. Machine vision is an application of computer vision aiming at the practical applications for industry. Machine vision systems are used to control other equipment such as robotic arms using digital I/O devices and computer networks. [40]

Machine vision systems are used in many fields, such as medical imaging, security and surveillance, traffic monitoring and face, iris and fingerprint analysis and recognition, while the emphasis is strongly in the manufacturing industry. There machine vision is especially used for automated visual inspections and robot control in assembly solutions. [41]

Visual inspections can be either quality control or identification tasks. The former may include inspecting surface color, luminosity or texture, measuring the size or shape of the object or verifying the existence of parts (e.g. screws). Reading barcodes, 2D codes or text using OCR are examples of the latter. Vision based robot control is superior compared to traditional applications. A robot following preprogrammed paths without any sensory feedback isn't able to adapt to minor variations in its surroundings. A vision controlled robot is able to grab the work pieces despite the inaccurate location or shape. [41]

2.6.5.2 Components of a Machine Vision System

A simple machine vision system could consist of the following equipment:

1. an optical sensor
2. a camera
3. lighting
4. camera interface for computer, “framegrabber”
5. computer software
6. digital signal hardware of a network connection to report results

The optical sensor triggers the camera when the part to be inspected is present. Lighting is an important part of a machine vision system. Lighting is discussed in depth in its own chapter later.

The camera takes the picture which is transferred to the computer by the framegrabber. The framegrabber captures individual frames from an analog video signal or a digital video stream. Historically they only had memory to acquire and store a single frame but nowadays they typically are able to store and compress multiple frames. Machine vision system computer software is used in multiple tasks from preprocessing of the acquired image to the interpretation of the scene. The signal hardware or the network connection is used to communicate with other hardware to take the intended actions. In modern machine vision cameras all the listed equipment except the optical sensor are combined together. Machine vision cameras are discussed in more depth in the next chapter. [42]

2.6.5.2.1 Cameras

There are two different sensor technologies for capturing digital images, CCD (charge coupled device) and CMOS (complementary metal oxide semiconductor). They are both developed in the late 1960s, but CCD sensors have dominated the market with their superior image quality until the development of lithography techniques in the 1990s [43], [44]. The manufacturing of CMOS sensors sets much higher requirements for the silicon wafers and fabricating methods than that of CCD sensors'. This is due to the different operating principles of the sensors. [45]

In both sensors, there is an array of small, light-sensitive pixels that convert the photons that hit them into electrons. In a CCD image sensor, there are only a few, or often only one output node that converts the charges of the pixels into voltage. The voltages are buffered and sent forward as an analog signal. The analog signal is amplified and converted to a digital signal in the camera's supporting circuitry (See Figure 22). In a CMOS sensor, each pixel has its own charge-to-voltage conversion. Each column of pixels has an amplifier and a multiplexer (See Figure 23). When each pixel is making its own conversion, the uniformity is lower. The output of the chip is a digital signal, thus requiring less additional circuitry outside the chip. [45]

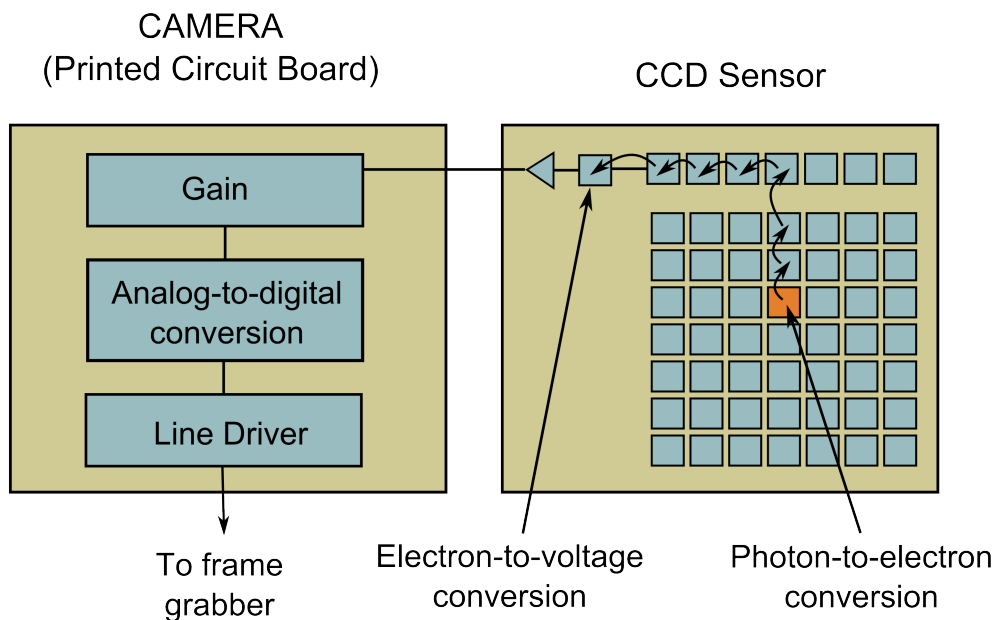


Figure 22: In a CCD image sensor, the charge-to-voltage conversion is done by one output node

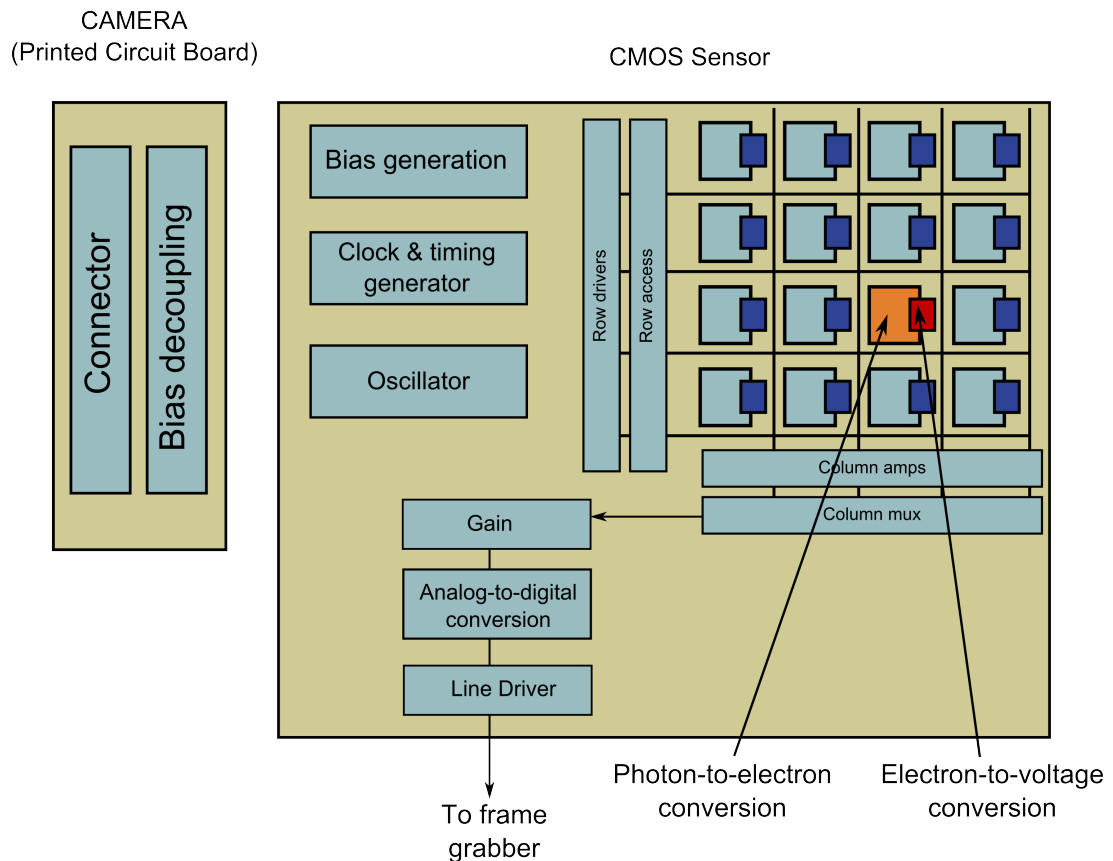


Figure 23: In a CMOS image sensor both the photon-to-electron and the charge-to-voltage conversions are done separately for each pixel

The most common resolutions of machine vision cameras range between VGA (640×480 pixels) and UXGA (1600×1200 pixels) [46], [47]. Usually the most significant factor in a succeeding machine vision system is not the high resolution of the image. The higher the resolution the more processing power is required. The same principle applies to the color. If the application does not truly require the use of a color camera, a monochrome camera should be used instead, because the larger amount of data produced by the color camera requires more processing power. The sensitivity of the camera is an important aspect. The higher the sensitivity, the less light is needed. This means shorter exposure times, which allows the target to move during the inspection. [48]

Today, cameras are much more than just their optics and sensor. They are full computers with their own processors, memory and networking equipment. Cameras store the machine vision software in their memory, process the image data according to it and send the inspection results to other devices. The most common camera interface is Ethernet. [48]

2.6.5.2.2 Computer Software [7]

As human vision, artificial vision is a multi-layered process. Machine vision system software is often divided into three layers: low-level, intermediate level and high-level vision. The low level vision deals with the image acquisition, filtering, thresholding and use of morphological operations. The image can be enhanced by reducing noise or a gray scale image can be converted to a black and white image.

Intermediate level vision detects features from the image using line, circle and ellipse detection and pattern matching. Depending on the application, the software can count, measure or identify objects in the image. Common techniques found in commercial and open source machine vision packages include also ready-made barcode and 2D code reading and OCR tools. [49] The machine vision software typically uses a number of different image processing techniques to identify the object.

High level vision interprets the data produced by the previous layers. For example, the software might compare the inspected object to the programmed criteria and either pass or fail the item. If the item fails, the software might signal the robot to remove the part from a conveyor or otherwise warn about the failure.

2.6.5.2.3 Lighting [7]

Lighting is possibly the most important factor in a successful use of machine vision. Or, in other words, it is the easiest way to ruin an otherwise working machine vision system. The effect is most notable in color identification applications, but affects all inspections. The lighting circumstances should remain as static as possible. The compensation of the changes is also possible, but demanding.

The 5 basic techniques for lighting the target are directional light, diffuse light, back lighting, strobe light and structured lighting.

In **directional lighting**, shown in Figure 24, bright light sources are aimed directly at the target surface. Directional light produces an intense, uneven light and strong shadows. In addition to normal images, it can be used to detect flaws on the surface by measuring the amount of scatter. A flat surface reflects the light into one direction, but if the surface has pits or scratches, the amount of light scattered to the camera increases.

Diffuse light is reflected onto the target surface by another surface or surfaces. This gives more even lighting and reduces shadows. Usually, this lighting scheme is used, when the surface characteristics are important. Diffuse light is illustrated in Figure 25.

Back light is aimed to the camera while the object being inspected is between the light source and the camera as shown in Figure 26. Back light is used, when the edges of the object should be as distinctive as possible e.g. for measuring or when the object can be reliably identified by its silhouette.

Strobe light is a very high intensity light that is applied only upon the exposure time. Using a strobe light enables short exposure times that are vital when capturing images of moving targets to avoid motion blur. Strobe light is illustrated in Figure 27.

Structured lighting means projecting stripes, points or other patterns onto the surface being inspected. In the simple solution, the disturbance in the pattern indicates the presence of the object. In more sophisticated applications the three-dimensional characteristics of the object are examined. Figures 28 and 29 show an approach to structured lighting. Figure 28 shows an example of the setting of a structured lighting system. In this scheme, two light planes are projected from different directions. When no object is present, the planes appear as a single stripe. If an object intercepts the planes, the stripe does not appear as continuous, but as divided into two stripes on the object's surface as shown in Figure 29.

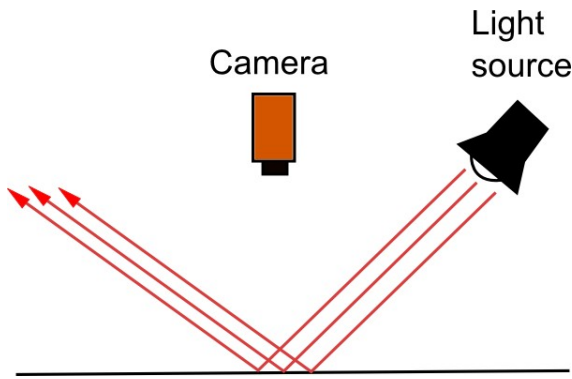


Figure 24: Directional lighting

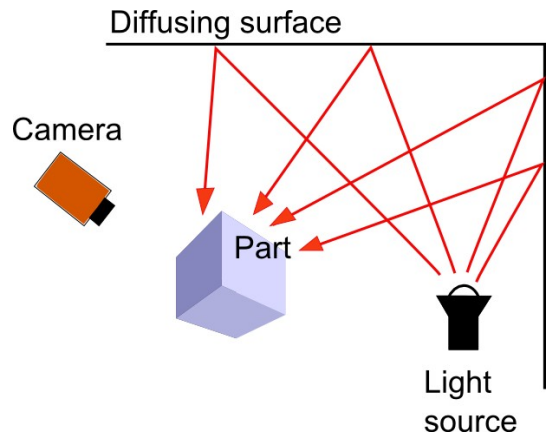


Figure 25: Diffuse lighting

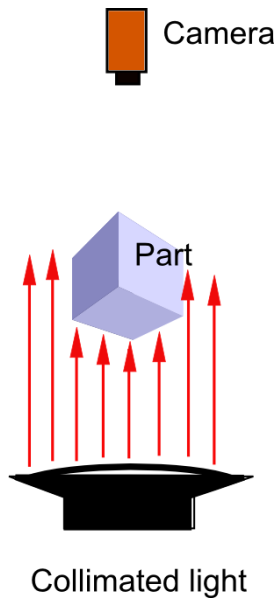


Figure 26: Back lighting

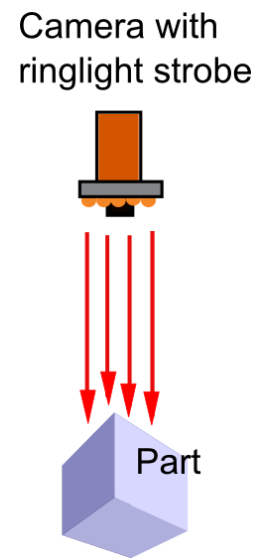


Figure 27: Strobe light

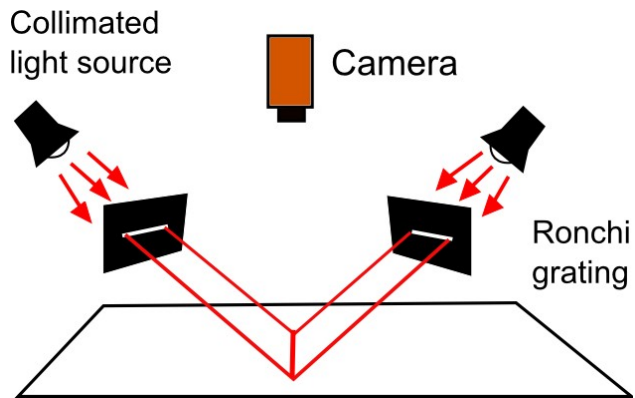


Figure 28: An example of a structured lighting system

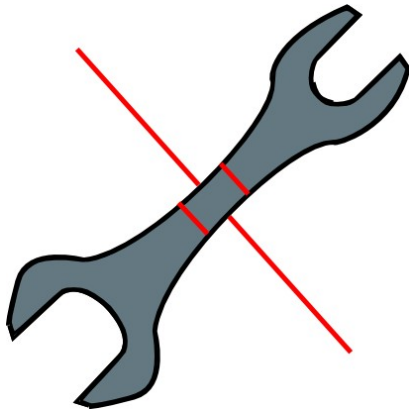


Figure 29: The light stripe split by the object

2.7 Robot Control

2.7.1 Robot Arm Kinematics

2.7.1.1 Introduction

Robot arm kinematics studies the geometry of motion of a robot arm with respect to a fixed reference coordinate system. It does not take into consideration the forces or moments causing the motion. Kinematics analytically describe the spatial displacement of the robot, especially the relations between the joint angles and the position and orientation of the end effector. The two fundamental problems in kinematics are the *direct* (also referred to as *forward*) *kinematics* problem and the *inverse kinematics* problem. [7]

The components of a robot arm form a kinematic chain. From the kinematics viewpoint, there are two types of components, joints and links. The links are the fixed construction between the joints in which the motion occurs. Joints can be divided into two types depending on the type of the motion. Revolute joints rotate about an axis of rotation. A prismatic joint allows telescopic motion. [50]

2.7.1.2 The Direct Kinematics Problem

The direct kinematics problem can be represented by the question:

“For a given manipulator, given the joint angle vector $q(t) = (q_1(t), q_2(t), \dots, q_n(t))^T$ and the geometric link parameters, where n is the number of the degrees of freedom, what is the position and orientation of the end-effector of the manipulator with respect to a reference coordinate system?” [7]

Systematic approaches for solving the problem include the use of trigonometry and matrix algebra. The reference coordinate system is the global coordinate frame (denoted with x, y and z). Another, local coordinate system (denoted with u, v and w) is established along the joint axis of each link. The links are rotated and/or translated with

a respect to the reference coordinate system as shown in Figure 30. The solution for the direct kinematics problem is the transformation matrix, which relates the two coordinate systems to each other. [7]

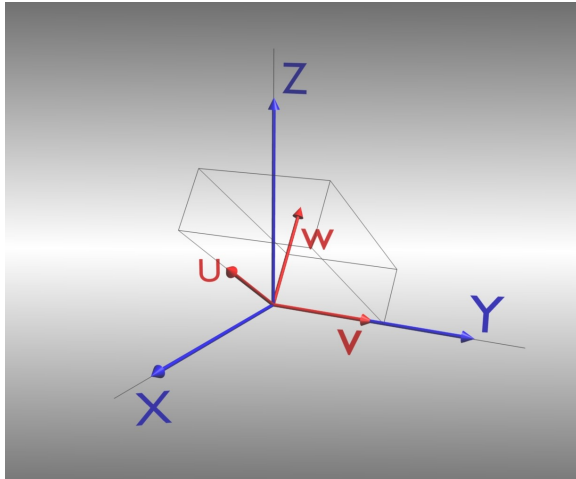


Figure 30: The coordinate systems (xyz, uvw)

2.7.1.3 The Inverse Kinematics Problem

The second fundamental problem in robot kinematics is the inverse kinematics problem. Inverse kinematics is in practice more important, because it provides the answer to the following question:

“Given the desired position and orientation of the end-effector of the manipulator and the geometric link parameters with respect to the reference coordinate system, can the manipulator reach the desired prescribed manipulator hand position and orientation? And if can, how many different manipulator configurations will satisfy the same condition?”[7]

Robots usually operate in a joint-variable space because they are powered by servo motors. The workpieces, on the other hand, are typically described using the world coordinate system instead. The inverse kinematics problem can be solved using various techniques. The methods include the use of inverse transformation or dual quaternions and iterative and geometric approaches. [7]

2.7.2 Robot Arm Dynamics

2.7.2.1 Introduction

Robot arm dynamics deals with the robot arm motion. The dynamic behavior of the manipulator is described as a set of dynamic equations. These equations are used in robot control, simulation and design. [7]

2.7.2.2 Equations of Motion [7]

The dynamic model of the robot arm can be formed using classical mechanics. The dynamic equations of motion are created for each joint when the geometric and inertial parameters of the links are known.

Different robot arm motion equations are derived from the Lagrange-Euler and Newton-Euler formulations. The dynamic motion equations of a six-joint robot arm are highly nonlinear and consist of various influencing forces, such as coupling reaction forces between joints that depend on the manipulator's structure, joint velocity and acceleration and the load being carried. The equations require a massive amount of arithmetic operations.

Development of efficient algorithms for computing the forces and torques has resulted in dynamic equations which are a set of forward and backward recursive equations that can be applied to the links sequentially. The velocities and accelerations of each link and joint are produced by the forward recursion and the forces and torques by the backward recursion. Such algorithms can be used for creating a real-time control of a robot.

2.7.2.3 Manipulator Trajectories [7]

Controlling the manipulator motion to follow a predefined path can be divided to two separate tasks, the trajectory planning and the motion control. The motion is typically defined by the initial point and the endpoint. Both points consist of a position and orientation and often there are more than one possible trajectories between them.

The trajectory is usually created by approximating the desired path as polynomial functions. This operation is referred to as interpolation. After interpolating the path, a sequence of time-based control points is generated. The manipulator's end effector moves via this points from the initial point to the endpoint. Two basic types of interpolation used in robot arm control are *linear interpolation* and *joint interpolation*. In linear interpolation, the path is a straight line between the two endpoints (Figure 31). In joint interpolation, the path is a smooth and polynomial line (Figure 32).

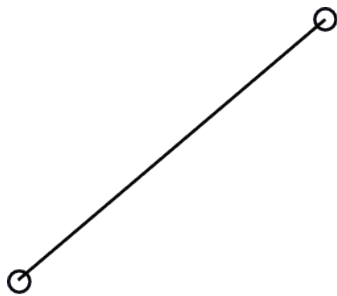


Figure 31: *Linear interpolation*

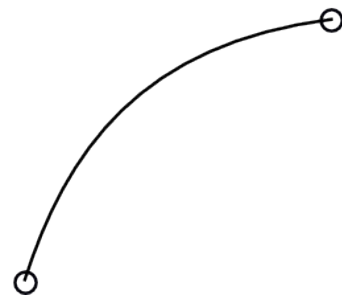


Figure 32: *Joint interpolation*

When a linear path is not specifically required, it is beneficial to use joint interpolation. In joint interpolation, the joints move only into one direction towards the desired position, which makes the movements faster. In linear interpolation, some joints have to change their movement direction during the operation.

2.7.2.4 Robot Trajectory Planning [7]

The kinematics of a robot arm were discussed in the previous chapters. Before actually moving the robot, it is necessary to take two things into consideration. The first is whether there are any obstacles in the robot's planned path. The second is whether the manipulator must traverse a specified path. The former is called *obstacle constraint* and the latter *path constraint*. Together these two constraints result in four different control modes shown in Table 1. The control problem can be divided into two subproblems, motion (trajectory) planning and motion control. Various trajectory planning schemes exist for obstacle-free motion.

Table 1: *The control modes [7]*

		Obstacle constraint	
		Yes	No
Path constraint	Yes	Off-line collision-free path planning plus on-line path tracking	Off-line path planning plus on-line path tracking
	No	Positional control plus on-line obstacle detection and avoidance	Positional control

Trajectory planning schemes approximate (interpolate) the desired path, the space curve between the manipulator initial and final locations, by a class of polynomial functions. Also a sequence of time based control set points is generated for the manipulator control between the initial location and destination. These path endpoints can be specified in joint coordinates or cartesian coordinates. Usually they are, however, specified in the latter, because the correct end-effector configurations are easier to visualize in cartesian coordinates. Joint coordinates are not suitable as a working coordinate system because the joint axes of the manipulators are not orthogonal and separate the position from orientation. If the joint coordinates are for some reason needed, the inverse kinematics can be used to make the conversion. Often there are many possible trajectories between the two endpoints. The path can, for example, be a straight line or a smooth, polynomial trajectory.

2.7.3 Robot Manipulator Control Techniques [7]

Robot control systems include real-time computers capable of reacting to sensory data and controlling the actuators of the robot thousands of times in a second [22]. Motion control can be divided into three categories: **joint motion control**, **resolved motion control** and **adaptive control**. The most simple robot arm control scheme is a design, in which each joint of the arm is treated as a separate, simple servomechanism. This is the joint motion control. A more efficient control method, the resolved motion control, utilizes dynamic models of the robot arm with nonlinear compensation of the interaction forces between the joints. Adaptive control is the scheme of choice when even the resolved motion control schemes turn out to be inadequate. In adaptive control a reference model is completed with an adaptation algorithm.

In early robots, such as the PUMA 560 robot arm, each of the joints was treated as a separate servomechanism. The **joint servomechanism control** is basically a PID controller (Proportional, Integral, Derivative). In the PUMA system, each of the joints of the robot arm has its own microprocessor as an integral part of the joint controller. Each of the microprocessors operates together with its own joint encoder, DA converter and a current amplifier. A supervisory computer handles user interaction and subtask planning for the microprocessors. In such a system, the feedback gains are constant and prespecified, which makes this control scheme incapable of updating the feedback gains under varying payloads. An industrial robot is a highly nonlinear system. The inertial loading, the coupling between joints and the gravity effects caused by the motion and configuration of the arm are either position- or position- and velocity-dependent terms.

This type of control system is not suitable for controlling a nonlinear system. An improvement is a digital control with a dynamic model of the arm providing the torques of the robot. This method is called the computed torque technique. In this control scheme, the desired time-base trajectory is tracked as closely as possible by calculating the forces/torques of all joints in real time using the Lagrange-Euler or Newton-Euler equations of the motion of the manipulator.

Correction torques are calculated from the position and derivative feedback signals and added to the torques calculated from the model. This compensates the modeling errors and parameter variations of the model.

The two previous control methods and the other joint motion control schemes are used to move the manipulator in the joint-variable space using a joint-interpolated trajectory. Because users are normally better oriented with the cartesian-coordinate space, the ability to control the manipulator hand to move in desired cartesian direction is often preferable. In **resolved motion control**, various joints of the manipulator are rotated simultaneously to move the manipulator hand along a desired world coordinate axis. The resolved motion control can be done using either motion rate, acceleration or force control.

The previous control methods are sometimes inadequate because of the inaccuracy of the dynamic model and the changes in the payload of the controlled manipulator during the work cycle. When the application requires high precision **adaptive control** is used. The simplest of the various control schemes is the model-referenced adaptive control. This computationally light method utilizes a suitable reference model and an adaptation algorithm. The errors between the reference model and the actual system are used to modify the feedback gains of the actuators of the system. The payload is taken into consideration by adding it to the weight of the final link.

2.7.4 Problems in Robot Control [22]

2.7.4.1 Ambiguous Problem

Upon a moving command the robot control system calculates the joint angle of each joint from the desired manipulator location and orientation using the inverse kinematics solution. Because sometimes the joints can rotate more than 360 degrees, the motors could be in multiple motor increments and the joints is multiple angles. This is called ambiguous problem.

Usually the angles closest to the initial angle are selected. However, depending on the previous robot position, the choice can turn out to be wrong and cause the external process cables wired to the end-effector to break. The easiest way to avoid this is to use reference positions with known joint angles during movement.

2.7.4.2 Singularity

Another common problem occurring with 6-DOF robot arms is the singularity. In some positions the robot loses one of its degrees of freedom, for example, when two of its joint axes become parallel. Such a situation is illustrated in Figure 33. In this case, the manipulator must be moved to the direction of the arrow with its orientation constant. To accomplish this, the joint J1 should be rotated counterclockwise. This would cause the manipulator angle to change, which cannot be compensated because of the singular configuration of the joints in the wrist. Thus, the robot cannot execute the moving command. Usually this stops the robot and triggers an error message.

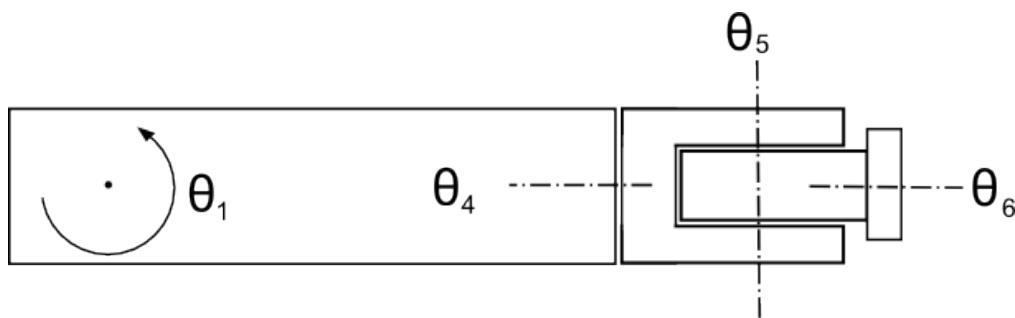


Figure 33: The singular point of the joints of the wrist [22]

Making smooth movements near the singular points is often challenging. The problem is not caused by the control algorithm but mechanical construction. The phenomenon cannot be completely avoided, but some robots operate better even near the singular points.

In the case of a typical 6-DOF robot arm, one way of dealing with the effects of singularity is adding a seventh, additional degree of freedom. This, however, requires more computing power because of special, optimizing algorithms for movement control. Also mechanical dilemmas arise, for instance a decrease in payload and a difficult transmission issue in the wrist. The advantages of the additional joint are the robot's ability to avoid objects around it and to reach positions unreachable for a 6-DOF robot.

2.7.5 The Work Environment of a Robot [22]

The work environment of a robot differs from that of a human worker. The inflexibility of an industrial robot sets high demands for the constancy of its surroundings. In most cases, robots operate as a part of a work cell. Depending on the task, the robot itself manipulates the products (e.g. welding or painting robot) or only moves them, possibly serving some other machine (e.g. CNC machining station). Typical accessories are discussed in the following chapter.

Simulation of the robot system plays a significant role in design and programming. Simulation is an efficient way for work cell design. Different layouts can easily be tested, the robot's work sequences defined and possible collisions detected safely. Different simulation methods are introduced in their own chapter later.

Besides being able to work in conditions unpleasant or even dangerous for humans, the robot itself can be a threat. Robots with powerful actuators and fast movements can cause accidents. Safety aspects are discussed briefly at the end of this chapter.

2.7.5.1 Accessories

A robot application consists of a robot, its software and accessories. The most common accessories include conveyors and turntables used for work piece handling. Conveyors can be controlled by the robot controller or a separate programmable logic controller. The turntables are typically operated using a servo mechanism that is integrated to the robot controller as an additional axis and that can be controlled similarly as the robot's axes.

As a part of a work cell, the robot can also serve other machines, such as presses, grinding and turning machines and assembly devices. A common feature for this equipment is that they typically have an operating system and that they communicate with the robot using I/O signals. These signals can be either analog or digital depending on the application. Traditional serial transfer (e.g. RS-232, RS-485, Ethernet) and field buses (e.g. Profibus-DP) are the most used interfaces. When needed, also standard analog signals (e.g. 0-10 V, 0-20 mA) are used.

2.7.5.2 Simulation

Simulation enables the verification of new robot programs before putting them into production. In simulation, the syntax of the program, the I/O traffic and correct and safe trajectories are confirmed. Simulation also makes it possible to try different layouts, specify the work sequences and detect collisions. Simulation shortens the product initialization time. In ideal cases, no test products are needed.

2.7.5.2.1 Reachability Simulation

In reachability simulation, the robot's geometrical and kinematic characteristics are modeled as realistically as possible. The geometry consists of a 3D CAD model of the robot. Reachability simulation is used for dimensioning the robot and its accessories, the gripper testing and validation of the whole work cell.

The kinematic characteristics are the description of the joints, of how they move and how much. The kinematics also includes the forward and inverse kinematic solutions of the robot arm. Another important feature in reachability simulation is the automatic geometry based collision detection. It makes it possible to discover the collisions occurring outside of view.

2.7.5.2.2 Process Simulation

As an addition to the geometrical and kinematical model used in reachability simulation, the process simulation model of the robot work cell consists of the model of the robot controller. In process simulation, normal robot programs performing production-like tasks are created. These programs are run with the simulated robot in a simulated environment. Thus, the controller model must be able to execute the same code the robot uses and control the simulated robot according to it.

The operation can be analysed both visually and numerically. In the most advanced simulators the whole production process can be simulated, meaning that in, for example, machining the correct material removal occurs in the work piece. Process simulation is used in off-line programming that is discussed later.

2.7.5.2.3 Dynamic Simulation

In dynamic simulation, the forces relating to the robot and the process are taken into account. The robot and the products move under the influence of the same forces as in reality, which sets especially high standards for the model of the robot controller. Also the masses of the equipment and the dynamic characteristics of the joints must be modelled. Dynamic simulation is used when there are high requirements for the accuracy of timing and placement.

Dynamic simulation requires a lot from the simulator. The simulation of the movements caused by varying loads must be extremely precise. The kinematic characteristics are not enough, also the control algorithms must correspond to the real robot controller. Dynamic simulation is used for example in testing the movement synchronization of a multi-robot work cell and in off-line programming of high speed work cells.

2.7.5.3 Off-line Programming

With small product batches simulation and off-line programming minimize the time the robot is out of the production during the change of the product. This has a direct effect on the profitability of the robot system investment. Off-line programming is done without the actual robot using only a 3D model of the robot and its accessories. Off-line programming is used when the robot cannot be programmed in a potentially hazardous production environment such as foundries, ammunition factories and nuclear plants. It is especially suitable for applications, where a large quantity of positions must be taught to the robot, as in welding, cutting, grinding or painting.

2.7.5.4 Safety

The measures for removing the dangers of the system can be divided into three levels. The first and most effective way is the design. This includes choosing a safe technology and processes, following safety principles when designing the control systems and eliminating manual tasks.

The second level is the use of technology, such as protecting barriers and safety devices. They should be used to protect from dangers that cannot be avoided using design methods. Barriers are used to prevent humans from reaching the dangerous area. Safety devices, such as door switches, do not prohibit movement but stop the machines, if someone enters the restricted area.

The remaining dangers, that cannot be eliminated even with the use of safety devices, must be reported to the receiver of the device. Warnings about the dangers can be attached to the device itself as well as in the instruction manual.

2.8 The Robot Work Cell

The robot work cell (Figure 34) is equipped with a robot arm, a conveyor belt with controlling PLC, two machine vision cameras, two operation panels and an additional axis, which operates a small turntable. There are also two web cameras and a light beacon with red, yellow and green lights. The equipment of the work cell is introduced in depth in the following chapters.

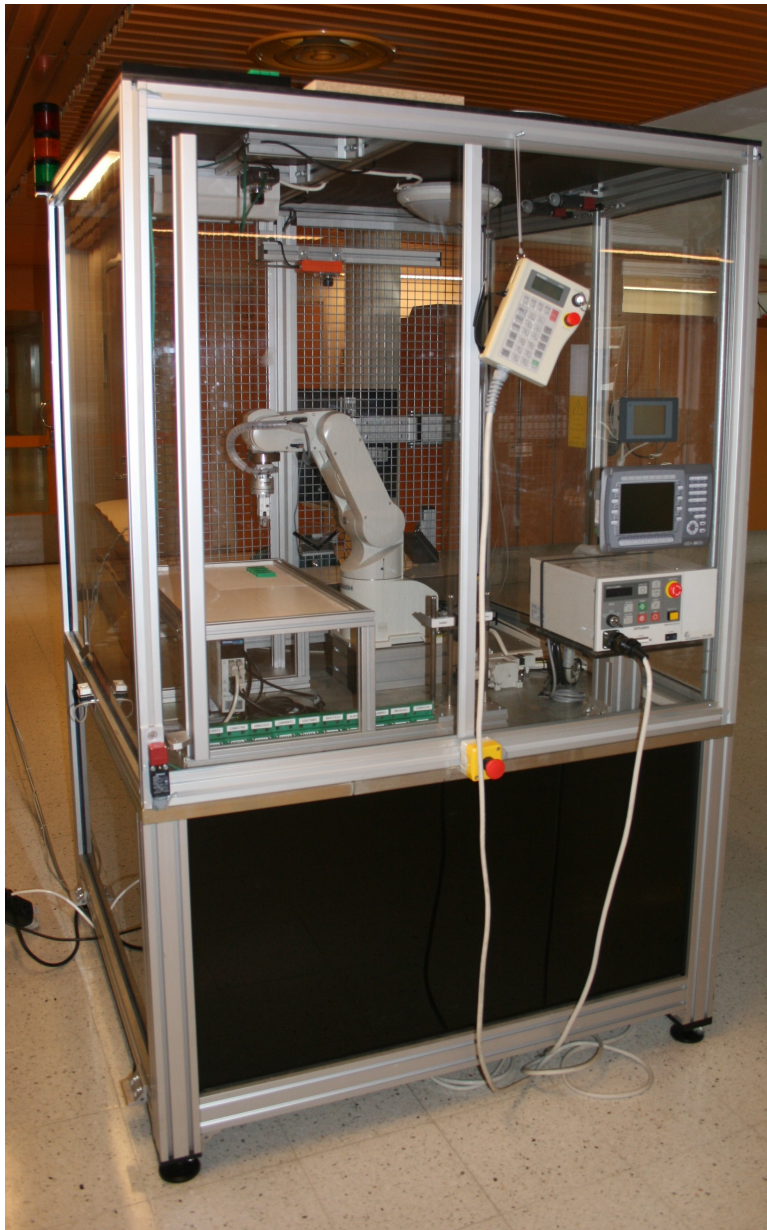


Figure 34: The robot work cell

2.8.1 The Equipment

RV-2AJ is a 5DOF (Degree of freedom) robot arm designed for light assembly work, and it is used in research labs, medical facilities and CD pressing plants. [51] The maximum reach distance is 410 mm. It is one of the smallest robots manufactured by Mitsubishi. It weighs only 17 kg and its' maximum payload is 2 kg and the recommended maximum payload for continuous operation is 1,5 kg. The robot can be supplied with four different pneumatic tools and a motorized hand. The base section has four outputs. The maximum resultant speed of its movements is 2100 mm/s and the repeatability 0,02 mm. RJ-2AJ is a vertical type robot that can be installed on the floor or suspended from the ceiling. The RV-2AJ robot is shown in Figure 35. [52]



Figure 35: Mitsubishi RV-2AJ robot arm

CR1 controller has a 64-bit RISC and DSP processor. Memory is reserved for up to 88 programs, of which each can have 5000 lines of code and 2500 position points. The controller uses two programming languages, Melfa-BASIC and MoveMaster. Communicating is possible with RS-232C port, RS422 port and RJ45 Ethernet cable. The controller has three expansion slots and there is already an additional axis and Ethernet interface installed to the controller. [53]

The most common use of the teaching pendant (TB) is the teaching of positions, but it also enables much more versatile operations, such as creating and editing programs, moving the robot and altering the robot's parameters. The R28TB teaching pendant complies with the IEC IP65 protection rating while the protection rating of the controller is only IP20, meaning that the controller is not protected against water. [53]

There are two machine vision cameras in the robot work cell, both manufactured by DVT. The color camera, DVT 542C, is shown in Figure 36. It has a 640×480 pixels CCD image sensor, Hitachi SH 4 processor, 64 MB of RAM and 16 MB of Flash memory [54]. The DVT 530 is a grayscale CCD camera with the same VGA resolution as the color camera. It has a Motorola Power PC processor, 32 MB of RAM and 16 MB of Flash memory [55]. Both cameras operate on 24 VDC and have their own LED ring lights to ensure the proper lighting of the target [54], [55]. In the vision system the cameras operate as servers sending processed data to the robot controller.



Figure 36: *DVT 542C machine vision sensor*

The conveyor in the work cell is operated by a 24 VDC motor. The width of the conveyor belt is 255 mm, the length of the working surface is 700 mm. The PCL controlling the conveyor belt is Mitsubishi FX1N-14 MR-DS. Its operating voltage is 12-24 VDC and it has 14 I/O ports. The PLC gets a signal from the IR sensor, when an object cuts its beam.

There are two Beijer Electronics' operation panels in the work cell. The smaller, E410 (Figure 37) is a black and white touch screen with the resolution of 320×240 pixels. It is connected to MELSEC FX1N PLC with RS-422 cable and together they are used to operate the conveyor belt. E410 has also RS-232 port and Ethernet connection. With various features, the operation panel can log and show data graphically and send it using FTP or operate as a HTML server as well. E410 has support for multiple languages. E1070 (Figure 38) is a 6,5" TFT display capable to show 65536 colors with the resolution of 640×480 pixels. It is operated using the buttons positioned along the edges of the screen. E1070 has 416 MHz RISC CPU and 64 MB of RAM. It has RS-422 and RS-232 ports, Ethernet and USB connections and also a slot for a CF card. Both of the panels use 24 VDC to operate. The panels are programmed with E-Designer 7 software. [56]

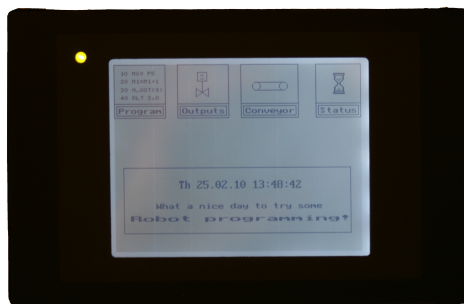


Figure 37: The E410 operation panel with touch screen



Figure 38: The E1070 operation panel

There are 2 Microsoft LifeCam VX-6000 web cameras in the work cell. These cameras have CMOS image sensors capable of capturing still images and video with a resolution of 1280×1024 pixels. There are also built-in microphones for capturing audio. Cameras are connected to PC with USB connectors.

2.8.2 The Software

Melfa-BASIC is a programming language used for programming Mitsubishi robots. Melfa-BASIC includes all usual programming structures such as flow control and repetition statements, different variable types, subroutines etc. and also a wide range of special functions and data types for robot applications. There is, for example, a distinct variable type for robot position. Another application specific data type that is for is the pallet. Pallets are arrays of positions declared by the four corner points and ideally suitable for handling large amounts of identical workpieces.

COSIMIR Industrial is a industrial robot programming and simulation software. Program has support for a wide range of robots from different manufacturers, different manipulators and common work cell equipment. It is also possible to create completely new models for simulation purposes. Programs can be simulated, debugged and transferred to the robot controller before execution. Also the position lists can be edited using COSIMIR. Other characteristics worth mentioning are RCI Explorer, which makes it possible to examine and record almost any data of the robot, including error reports, positions, speeds and servo voltages and currents. It is also easy to change the robot's parameters using RCI explorer.

The machine vision systems for the DVT vision sensors are made using **Intellect 1.5** software. Intellect has an camera emulator which consists of all the characteristics of a real DVT vision sensor. The machine vision system is built from small modules, *Tools*, each performing one specific task. There are tools for preprocessing the image, identifying shapes, measuring distances etc. The tools can be chained to be applied in specific order, for example, preprocessing filters before other tools. The final inspection result is the combination of the results of the different tools.

E-designer 7 is a Human Interface Design software by Mitsubishi Electric Automation, Inc. It is used for creating programs to monitor and control the other work cell devices with the operation panels. E-designer has a support for dozens of manufacturers, and the controllers for both robot and the DVT cameras could be found easily with comprehensive instructions. Also great samples for the all operation panel models could be found. The operation panel software is built as a hierarchy of blocks linked to each other. There are some ready-made blocks for built-in operations such as the System monitor block. User-made blocks can contain text, digital symbols, buttons and touch keys (if the panel has a touch screen) and bitmap images.

3 The Modification and the Exercises

3.1 Battery Change and Calibration

The robot position is detected by absolute encoders. While the toolpower is turned off, the position must be saved with back up batteries. The batteries need to be replaced regularly. There are in total six batteries to be replaced, five of them in the robot arm and one in the additional axis unit. The batteries are 3,6 V lithium batteries. Normally, the controller power should be turned on to avoid losing the encoder data when the batteries are removed, but in this case, since we were going to reset the origin, the controller was turned off.

The battery holder in the robot is located inside the robot arm's lowest part. The robot must be turned to a specific position to make the removal of the cover possible. New batteries were soldered directly to the cables. The new batteries were smaller than the original ones. Thus, some filler was needed for the batteries to stay in the battery holders.

There are three possible methods for setting the origin of the robot, the *origin data input method*, the *mechanical stopper method* and the *user origin method*. The calibration of the robot coordinates was carried out using the mechanical stopper method. In this method, the servos are turned off and the breaks of each axis released one at a time. The arm is then moved by hand to come in contact with the mechanical stopper. This origin is recorded for the axis. After a successful calibration, the calculated origin point was written inside of the plastic shoulder cover where the list of calibration points was located.

3.2 Robot Work Cell Exercises

3.2.1 The Pedagogical Goals of the Exercises

The aim of the created exercises is to make it possible for the student to learn the use of all the equipment in the robot work cell. The emphasis is in Melfa-BASIC programming and machine vision. Also, there are instructions and some exercises for setting up the operation panels and the conveyor logic. The practices are applying the previously learned issues and the difficulty level is increasing smoothly.

After completing the exercises, the student should

- 1) be able to record position lists using the teaching pendant
- 2) know the syntax and the most common keywords in Melfa-BASIC programming language
- 3) know the use of pallets in Melfa-BASIC
- 4) know, how to use robot outputs (M_OUT(X))
- 5) know the basic use of COSIMIR and to be able to transfer programs and position lists between the robot and PC
- 6) be able to create and transfer machine vision systems using Intellect 1.5
- 7) know, how to send data from the machine vision sensor using DataLink
- 8) be able to input and output data using robot program and COM lines
- 9) have the necessary knowhow to create E410 operation panel software including digital symbols, touch keys and graphics and being able to handle I/O
- 10) have general knowledge of the common problems occurring in the robot programming and how to avoid and solve them

The tasks were divided to three difficulty levels: *Beginner*, *Intermediate* and *Advanced*. The Beginner level tasks are practically tutorials of the use of the device or software in question. The Intermediate level exercises give the student an opportunity to use the new skills in different tasks. Advanced level exercises are more demanding, successful completion of them requires independent learning and problem solving skills.

In addition to the exercises, also a set of exam questions was designed. Because these exercises cover a very wide area of different devices in the work cell, the exam was planned to include only the needed questions. There is a series of complete questions with sample answers to give the instructor the possibility to choose the emphasis according to the learned issues. The questions include explanation of terms introduced in the exercise handout, essay type questions and error correcting of Melfa-BASIC code.

The grading system of the course is figured in a spread sheet that calculates the point limits for each grade. By default, 50 % of maximum points give the grade 1 and 95 % give the grade 5. Also the number of points per exercise can be set, the defaults are 3, 6 and 10 points depending on the task difficulty. The exam tasks are designed to give 6 points, but also this can be changed.

3.2.2 The Exercises

3.2.2.1 Programming Exercises

Before starting the first exercise, there is a summarized speed guide to get started with Melfa-BASIC. It presents how to write programs with the teaching pendant and COSIMIR, goes through the common variables, condition statements and movement jogs and teaches to set position points and to debug the programs in steps. There is also an example of each topic. This guide was written, because it was thought to be frustrating for the student to leaf through the 437 page controller instruction manual to get started.

The first task is a simple moving and placing tutorial, which gives explicit guidelines to get started with programming. A short program moves a mosaic tile from one position to another. This task gives a base to the most common commands to move the robot, to delay, turn outputs on and off and to end the program. The student is provided with a complete code listing, illustrated list of needed positions and a guide on how to download programs to the robot controller and to set the operating speed of the robot.

The second task deals with palletizing functions and it is still a very easy one, but this time there is no ready solution to the task, only one brief example of the palletizing. In the real world applications, palletizing is one of the most common tasks in simple placing and packing work done by robots. In the task, two different kinds of pallets must be created, a rectangular and an arc. The plastic pieces, placed in the rectangular pallet, must be moved to the arc pallet by adding the pallet index. Next, the pieces are moved back to the arc pallet in a reverse order. The program is repeated three times. The purpose of this task is to guide the student to use pallets, variables and if- and for-statements. Instructions of the palletizing commands and figures of the possible assignment directions are described before the actual task.

In the third task the mosaic tiles are used to form alphanumeric characters from mosaic tiles. The letters or numbers are created by using the Dot Matrix technique, in which each character is formed by dots in a 2-dimensional array of dots. In this exercise the student will learn more about programming. The palletizing functions in Melfa-BASIC introduced in the previous task are taken into use. Because of the small size of the mosaic tiles, the positions of the pallets must be recorded precisely or the robot will not be able to pick them with the vacuum gripper. Conditional statements, integer variables and loops are needed to create the work cycle. The biggest challenge will probably be to figure out, how the robot handles the information about the two different colored mosaics. In the example solution, each letter array is coded as a series of ones and zeros denoting the foreground and background colors. The string is examined with the substring function.

Tasks 4, 5 and 6 are additions to task 3. In task 4, the aim is to make the robot return the mosaic tiles after assembling the character. This requires the robot to “remember” the tile colors thus testing the system used for coding the colors. The operation can be implemented in various ways. In task 5 the two previous programs are combined to write full words. The word to be written must be examined letter by letter using the substring function in Melfa-BASIC. Task 6 utilizes the Ethernet interface. The aim is to establish a connection between the laptop PC and the robot controller and to send the word to the robot from the PC. Also the robot sends data to the PC. This task teaches the use of input and output functions in Melfa-BASIC.

Tasks 7 and 8 are intermediate exercises. Both tasks test the student's knowledge of the previously learned programming techniques. As an aim to create a functioning digital clock program, student has to know how to use pallets, variables, loops and subroutines. The usage of substrings and the command to return the system time has to be found out - independent learning from the Instructions Manual eases the completion of the exercise. In these exercises the robot moves either mosaic pieces or green plastic blocks to pallets, in order to show the current time. Time is constantly inspected, and when needed, updated. To make the task 8 more challenging than task 7, removed segments must be moved to the places which need a new segment while an update occurs.

3.2.2.2 Advanced Programming Exercises

Task 9 deals with subroutines. In the task, two programs are written. First, the main program moves the robot to the initial position and then calls the subroutine to calculate the next point where to move the robot. The next position is calculated in relation to the current position point. The subroutine is called four times, and the robot moves through a square shaped path.

In the task 10, subroutines are used again to change the tool according to the inspected object. The camera is sending a string consisting of the object type and position of the object's pickpoint, then the program decides, if it needs to change the tool to move the object to the right container. Relating to this task, the student learns the use of the tool coordinates, corresponding parameters and commands to set and checking of the tool number.

In the first multitasking exercise, task 11, there are two pallets in two separate programs. Both of the programs move a nut in the program's own pallet to the next pallet index. This happens in turns and an external variable is used as a signal between the programs, when the nut is moved. When the nut is in the last pallet index, it is moved back to the start in a reverse order. In this task, the student learns to load programs to the multitasking slots, to start and end multitasking programs, to use external variables as flags, to clear the multitasking slots and to pass the mechanism control from one program to another. In industry, these kinds of multitasking applications can be utilized when there are multiple processes using only one robot.

In the real industrial systems, multitasking can greatly optimize the processing time, if next inspection can be done at the same time as the previous product is being processed. In task 12 the robot moves the nuts located on the conveyor to the pallet in the worktop. This is done by synchronizing the program moving the robot and the program inspecting the camera. Two external variables work as flags to inform if the camera is inspecting or ready or if the robot is moving or idle. The student learns to pass data with external variables and to handle multiple input data flows from the camera.

The task 13 is a basic connection exercise which teaches the student to move data between the robot controller and the other devices such as smart cameras, terminals or computers and to use interrupts to start subprograms if a signal is set on or off. In the real systems, data is commonly transferred to be monitored in control rooms or to be archived in databases. This task gives a very practical base to get started with writing the communication programs with Melfa-BASIC and C#. First before the actual task, there are complete code listings for a simple PC server application. The robot communication parameters are also explained and set to ensure correct functioning as a client. There is a list of the Melfa-BASIC communication commands with useful examples. In the task, communication settings are defined, the connection is opened and data sent between the PC and the robot.

3.2.2.3 E-designer Exercises

The first E-designer exercise, task 14, introduces the most common controls used in the operation panel software. In the task, a rocker button on the touch screen is bound to the digital I/O of the PLC controlling the conveyor. The conveyor can be started and stopped using the operation panel. Also the testing and transfer of the program is explained in this task. The aim of this exercise is to learn the basic use of E-designer software.

In the next task, the previous software is developed further by adding a new block with imported graphics and a touch key. A built-in function is set as the action of the touch key and the properties of the touch key are modified thus introducing them to the student.

In task 16, the student will learn to set security levels to the blocks of the operation panel software. Also this task is done by developing the existing software. In the exercise, three different security levels are set to control the access to the blocks of the panel software. The panel's ready-made automatic login option is used as an example of E-designer's features that ease the programmer's work significantly.

In the task 17 the ready panel software is translated to some other language than English. The panel software can be localized easily with a built-in tool. Translating of the texts, testing the localization within the E-designer and the use of language register are explained.

3.2.2.4 Machine Vision Exercises

In the first machine vision exercise, task 18, the student will learn the usage of DVT Intellect software. This includes opening of the workspace, emulation of the camera, opening the picture sequence, adding the products, tool layers and tools and saving the system. The actual task is to use machine vision to locate and measure nuts. For this, the student has to use multiple tools, such as *Area Positioning*, *Line Fit*, *Circle Fit* and *Script* tools.

The next exercise widens the student's knowledge of the available tools. The creation of object positioning and other things learned in the previous exercise are rehearsed with a different object. This time the image is preprocessed before applying other tools. The use of DataLink and the String Expression Editor are learned. In the exercise the object's pick point is sent as a DataLink string. Also the Melfa-BASIC code needed for receiving the position data into the robot's program is given.

The last exercise, Task 20, is the most demanding task in the handout. It can be used as a final task in the robotics course because it makes use of all the previous techniques in the handout. In the task, special products are inspected with machine vision. The product has many inspection items thus making the machine vision system complicated. The machine vision system created in this exercise is meant to be used to actually control the robot to move the pieces off from the conveyor.

3.3 Accessories

3.3.1 RobotCamera

There have been a few attempts to send live video from the robot work cell in remote control purposes [2]. Many free and commercial software have been tested, but with poor results. This problem was also examined in this study. After testing some camera software, writing of a new program was seen as the only possible option. The intention in previous theses had been to develop monitoring and remote control features, and this is why robot cell was equipped with two additional web cameras, both are Microsoft LifeCam VX-6000 series.

3.3.1.1 Background

There have been difficulties in sending the web camera images from the server PC to other computers with tolerable framerate. This issue has been a major obstruction in remote controlling the robot; in case of a faulty program, the robot may collide before the operator notices the situation and has time to stop the program. In this project, a couple of promising programs were tested and compared to find out if there was an applicable one to be used in remote control purposes.

The first tested program was a commercial software, Active Webcam, which was already installed in the PC. Active Webcam is a video capture and surveillance program, which can stream ActiveX video from multiple cameras to a web server or optionally upload images to the HTTP or FTP server. Program has a *Watchdog* feature, which notices if program crashes and restarts it again. Active Webcam program was configured to send pictures to the web server from both of the cameras in five second intervals and also the web page was refreshed in every five seconds. ActiveX video could be sent, but receiving was only possible with Internet Explorer web browser, which was slow and unreliable with its many security breaches. As it was tried to speed up the sending with FTP, the program became very unstable and crashed the operating system. After careful examinations, the program turned out to be way too heavy to be used in the old server PC and it was decided to seek for an alternative program.

The second tested program was Fwink, which is an open source web camera application. The program was light and could send images faster than Active Webcam, upload was done with the interval of two seconds. Unfortunately, Fwink was only able to upload images through FTP and capable to capture image from one camera. Therefore, other solutions had to be considered.

As the web camera programs were studied and the most common video formats and data transfer protocols were familiarized with, the result was, that there are no applications having all of the required features. Also, when there was already a basic idea of the TicTacToe game and the game's internal way to receive video had to be developed, this is why it was clear that there were no ready solutions and it was decided to code a new application of our own.

Previously in the practical training period, it was considered to apply more of the machine vision features to the robot by using the server PC. This was planned to be done by using OpenCV computer vision library. OpenCV was originally commercially developed by Intel, but in the year 2000 it was released as an open source application programming interface. As the other PC based programs written in this study, Robot Camera was written in C# using Visual Studio 2008 Express Edition. OpenCV is created using C++, and EmguCV wrapper was needed to use the library with C# language.

3.3.1.2 Description of Operation

Video is sent using Robot Camera server software which uses TCP/IP protocol to transfer the video. TCP/IP was chosen to be used instead of UDP, because there turned out to be difficulties in sending the UDP packets through LAN. The communication method is a direct data stream between the server and the clients. The RobotCamera can accept up to 10 clients, but the number of clients affects the framerate, though. This is caused by loops inspecting the client socket status and selected stream number.

It was decided to be pointless to send clients both video feeds at a time and Robot Camera sends only one in order to save bandwidth. Clients still have the possibility to choose which camera they want to monitor. The server captures images from both cameras and converts them to byte streams even if there are no clients connected. It is also possible to send a still picture when there is no connection to neither of the cameras.

To not to forget the FTP functions of Active Webcam software, also a simple FTP client was implemented in this program. Images can be transferred to FTP server after a delay of one second, but while it is useless to update images that often, a 5-second-delay is set as a default. Other features consist of attaching the video colored text with date and time as well as Savonia logo. The resolution of the video can be changed in Robot Camera. The maximum resolution depends on the camera's capabilities. The functions of the Robot Camera program can be coarsely divided in *connecting*, *receiving*, *sending* and *image capturing* points. The structures of these parts are briefly explained in the next listings.

Listing 1: Connecting the clients to the server

```

for (int i = 0; i < socketCount; i++)
{
    if (usedSockets[i] == false)//if socket is not used already,
connect
    {
        usedSockets[i] = true;
        clientSockets[i] = tempSocket;

        clientSockets[i].BeginReceive(rcv, 0, 1024,
SocketFlags.None, new AsyncCallback(Receiving), clientSockets[i]);
        clientIPs[i] =
clientSockets[i].RemoteEndPoint.ToString();
        connect_socket.BeginAccept(new AsyncCallback(Connecting),
connect_socket);
        AddToconsole(Color.LightGreen, "Client connected to
socket " + i + "\nClient IP (" + clientIPs[i] + ") index #" + i);
        break;
    }

    if (i == socketCount)//all sockets are full, disconnect
    {
        AddToconsole(errorColor, "No free sockets for client...
disconnecting");
        clientSockets[i].Shutdown(SocketShutdown.Both);
        clientSockets[i].Disconnect(false);
        connect_socket.BeginAccept(new AsyncCallback(Connecting),
connect_socket);
    }
}

```

In the listing 1 it can be seen that inside the connection phase, each array index is checked. If *usedSockets* boolean is true, socket is already connected. When free socket is found, connection is moved from *connect_socket* to the free *clientSockets* index. Connection socket *connect_socket* starts to accept the clients and a new client socket starts to receive images from the server. If there are no free sockets, the client socket is disconnected from the server.

Listing 2: *Asynchronous receiving of client messages*

```

for (int i = 0; i < socketCount; i++)
{
    if (sendingIP == clientIPs[i])//find the sender of the message
    {
        //if message is null, user has disconnected
        if (receivedMessage.Length == 0)
        {
            AddToconsole(Color.White, "Client " + i + "
disconnected!");
            clientSockets[i].Shutdown(SocketShutdown.Both);
            clientSockets[i].Disconnect(false);
            clientNames[i] = string.Empty;
            clientIPs[i] = null;
            usedSockets[i] = false;
        }
        else
        {
            if (receivedMessage.Contains("1") == true)
            {
                //user changes camera source to #1
                selectedCam[i] = false;
                sendingClient.BeginReceive(rcv, 0, rcv.Length,
SocketFlags.None, new AsyncCallback(Receiving), sendingClient);
            }
            if (receivedMessage.Contains("2") == true)
            {
                //user changes camera source to #2
                selectedCam[i] = true;
                sendingClient.BeginReceive(rcv, 0, rcv.Length,
SocketFlags.None, new AsyncCallback(Receiving), sendingClient);
            }

            if (receivedMessage.Substring(0, 4) == "EXIT")//user disconnects
            {
                AddToconsole(Color.White, "Client " + i + "
disconnected!");
                clientSockets[i].Shutdown(SocketShutdown.Both);
                clientSockets[i].Disconnect(true);
                usedSockets[i] = false;
            }
        }
    }
}

```

Listing 2 shows that when a client message is received, it is inspected at first. When the matching IP address is found from the IP array, corresponding action is done for the client. The client has the possibility to change between the two camera sources and to end the session. In the client side, these actions are done with buttons.

Listing 3: Capturing the camera images

```
picture1 = captureImage1.QueryFrame().ToBitmap();
picture2 = captureImage2.QueryFrame().ToBitmap();
```

Image is captured from a web camera with one simple line (see listing 3). This is done in the *SendTimer_Tick* event. Capture is done every 40 ms by both web cameras. Pictures are converted to bitmaps.

Listing 4: Attaching text and logo to the images

```
string textFormat = textTxt.Text;
//text to be added
//parameters are parsed of the string
if (textFormat.Contains("%D") == true)
{
    textFormat = textFormat.Replace("%D",
DateTime.Now.ToShortDateString());
}
if (textFormat.Contains("%T") == true)
{
    textFormat = textFormat.Replace("%T",
DateTime.Now.ToString("HH:mm:ss"));
}
if (textFormat.Contains("%CR") == true)
{
    textFormat = textFormat.Replace("%CR", "\n");
}

//image is converted to bitmap,
//and text is attached with graphic tools
picture1 = captureImage1.QueryFrame().ToBitmap();
picture2 = new Bitmap(errorImageTxt.Text);
drawText = Graphics.FromImage(picture1);
drawText.DrawString(textFormat, videoFont, brush, new PointF(0, 0));

//resolution is shown in the camera tab
resolution1.Text = "(" + picture1.Width + " x " + picture1.Height +
)";
resolution2.Text = "(" + picture2.Width + " x " + picture2.Height +
)";

//Savonia logo is added to image
if (logoBox.Checked == true)
{
    ImageAttributes attr = new ImageAttributes();
    attr.SetColorKey(Color.Fuchsia, Color.Fuchsia);
    Rectangle rectangle = new Rectangle(300, 5, logo.Width,
logo.Height);
    drawText.DrawImage(logo, rectangle, 0, 0, logo.Width, logo.Height,
GraphicsUnit.Pixel, attr);
}
//image is shown in the camera preview window
frameBox1.BackgroundImage = picture1;
```

The listing 4 is also part of the *SendTimer_Tick* event. If the server adds also text to the image, it is parsed here (as seen in Figure 39). If the text has tags for time or date, they are replaced with values. There is also a tag for a new line. Text is attached to the image by using *System.Graphics* statement. Also the logo is attached to the image with the same tools.

Listing 5: Converting the images to bytes

```
//create memory stream
MemoryStream mStream1 = new MemoryStream();
//flush stream just in case
mStream1.Flush();
//and save image to stream
picture1.Save(mStream1, ImageFormat.Jpeg);
//convert stream to byte array
snd1 = mStream1.GetBuffer();
//flush the stream
mStream1.Flush();
```

Listing 6: Sending the byte arrays

```
for (int allClients = 0; allClients < socketCount; allClients++)
{
    //send to all used sockets
    if (usedSockets[allClients] == true)
    {
        //if camera boolean is false, send #1
        if (selectedCam[allClients] == false)
        {
            clientSockets[allClients].BeginSend(snd1, 0, snd1.Length,
SocketFlags.None, new AsyncCallback(Sending),
clientSockets[allClients]);
        }
        //if camera boolean is true, send #2
        else if (selectedCam[allClients] == true)
        {
            clientSockets[allClients].BeginSend(snd2, 0, snd2.Length,
SocketFlags.None, new AsyncCallback(Sending),
clientSockets[allClients]);
        }
    }
}
```

Images are first saved in memory streams and then converted to byte arrays (listing 5). While sending the images, the state of socket is checked at first. If the socket is connected, the requested data stream is checked. The feed from camera 1 or camera 2 is sent correspondingly as seen in the listing 6.

Listing 7: Sending the images using FTP

```

//paths for both images
Uri ftpTarget1 = new Uri(serverTxt.Text + "/cam1.jpg");
Uri ftpTarget2 = new Uri(serverTxt.Text + "/cam2.jpg");
//image is saved to .jpg
frameBox1.BackgroundImage.Save(System.Environment.CurrentDirectory +
"\cam1.jpg", ImageFormat.Jpeg);
frameBox2.BackgroundImage.Save(System.Environment.CurrentDirectory +
"\cam2.jpg", ImageFormat.Jpeg);
//creating webclient
WebClient wc = new WebClient();
wc.Proxy = null;
//set needed credentials
wc.Credentials = new NetworkCredential(userTxt.Text,
passwordTxt.Text);
//upload the images
wc.UploadFile(ftpTarget1, System.Environment.CurrentDirectory +
"\cam1.jpg");
wc.UploadFile(ftpTarget2, System.Environment.CurrentDirectory +
"\cam2.jpg");
//destroy the webclient
wc.Dispose();

```

In the listing 7 FTP client defines paths for both images. Then current images of the video preview boxes are saved to JPEG format to the program folder. A new webclient is created and credentials are passed for the webclient from the text boxes consisting of user name and password. Files are uploaded to the server and a web client is disposed.

3.3.1.3 Viewing software

Client computer must be provided with Camera View application and server IP address and port have to be inserted to program in order to connect to the server. The client has an option to choose either of the two datastreams to receive. Currently images can be captured in a minimum of 40 ms and maximum of 60 second intervals. Camera View program scales video automatically depending on the resolution.

Listing 8: Connecting Camera View to Robot Camera

```

public void connectToServer(IPAddress ip, int port)
{
    //if known IP is selected, proceed with it
    if (connect_select.SelectedIndex == 0)
    {
        iep = new IPEndPoint(ip, port);
    }
    //if IP is found out by DNS, connect with it
    else if (connect_select.SelectedIndex == 1)
    {
        string dnsString = ipBox.Text;
        IPAddress[] addresslist = Dns.GetHostAddresses(dnsString);
        MessageBox.Show(addresslist[0].ToString());
        iep = new IPEndPoint(addresslist[0], port);
    }
    //connect with selected settings
    connect_socket.BeginConnect(iep, new AsyncCallback(onConnected),
connect_socket);
}

```

As seen in listing 8 , the connection of the Camera View program can be done with a known IP address or by requesting it from a DNS server. Connection type is selected by a combobox in the left corner of the window. Depending on the connection type, there is a textbox which contains either the IP address or the domain name.

In the receiving function, the frames per second counter is incremented by one each time an image is received. The counter is shown and reseted every second. Image bytes are written in a memory stream and then the stream is converted to an image. After this, the image is shown in the picture box of the application.

The messages sent by the client are limited to disconnect from the server and to change the video stream. To change the camera view, the client program simply sends number 1 or number 2 to the server. In practice, this happens by pressing the camera buttons. To end the session, EXIT word is sent to the server with disconnect or exit buttons.

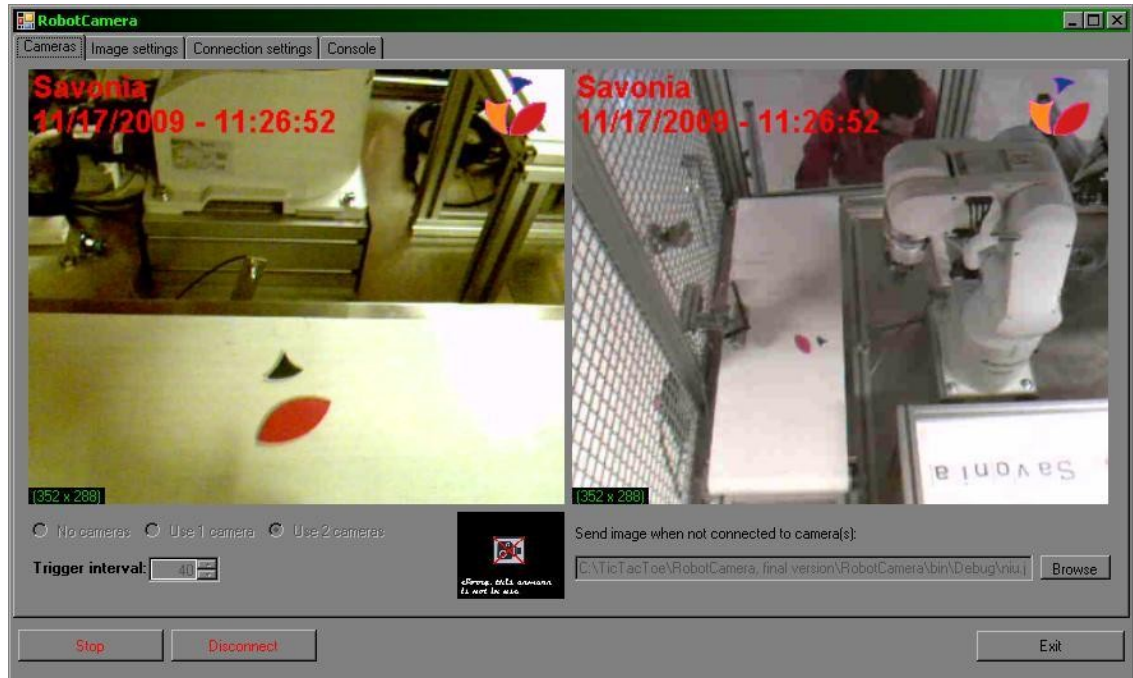


Figure 39: Robot Camera in use

3.3.2 Tool Change Program

Changing the robot's tool can be a repeating task in the continuous use of the robot. This is why it is not a good idea to program it over and over again in each application. Creating a subroutine that can either be included in the source code or called from other programs, is a preferable solution. In the Melfa-BASIC language, there is a command to run another program as a subroutine. With this command, the tool change program can be called inside another program. The manual version of this program can be accessed by using the Server Console.

In the first version of the program the user was allowed to remove or add both of the tools or to change from one tool to another. In this version, the tool coordinates were not changed and the robot had its tooltip in the center of the end-effector regardless of the tool. Because of the length difference of the tools, the position points had to be separate for both tools to not to crash to the table. This slowed down the programming and there was also a risk to accidentally collide with the other tool in the tool rack if a wrong input value was given to the program.

The next version had tool coordinates set to the robot parameters and this made it safer and easier to change tools. With the parameters, it was possible to check which tool robot had currently in use and allowed to create a program which only changes the tool to another. Also the risk of colliding with a wrong tool was gone, because the tool's length was compensated in the position points.

3.3.3 Server Console

Server Console and Client Console are HMI applications developed to interact with the robot cell either locally or remotely. Server Console is a server software which can connect multiple clients. At the beginning of this project it was used for straight communication to input and output data from the laptop, while the robot controller was running as a client. In the previous studies, possibilities to remote control the robot had been planned, and this is why this program was a good base to continue the development.

These programs were created with C# as well as the other projects. Also the almost same network source code was used as in the Robot Camera application. These programs were designed to resemble console applications, because of the simplicity. There was no need for a graphical user interface and this kind of application was fast to write.

Server Console installed in the laptop is meant to be used only in local communication with the robot cell. It connects only to one client at a time. The server PC below the robot cell has a program with remote access functions. Up to ten clients can be connected to Server Console and if everyone of the PC clients disconnect, the server turns the robot program off. In order to use remote control features, also the robot has a program which receives commands from the server and can start or stop programs.

In Server Console, the clients have a socket array, a boolean array for used sockets, and an array for endpoints. The only differences compared to the Robot Camera application's network code were made to ensure that the robot could not receive the same messages it had sent. The messages of the clients are directed to the other clients and to the robot.

Listing 9: Connecting the robot and clients to Server Console

```

//if the endpoint has robot's IP
if (tempSocket.LocalEndPoint.ToString().Contains("192.168.0.1") ==
true)
{
    //connect to robot socket
    robot_socket = tempSocket;
    robot_socket.BeginReceive(robot_rcv, 0, 1024, SocketFlags.None,
new AsyncCallback(Receiving), robot_socket);

    //send message to all connected clients
    for (int allClients = 0; allClients < socketCount; allClients++)
    {
        if (usedSockets[allClients] == true)
        {
            if
(clientSockets[allClients].LocalEndPoint.ToString().Contains("192.168
.0.1")==false)
            {
                snd = Encoding.ASCII.GetBytes("Robot connected (" +
tempSocket.LocalEndPoint.ToString() + ")");
                clientSockets[allClients].BeginSend(snd, 0,
snd.Length, SocketFlags.None, new AsyncCallback(Sending),
clientSockets[allClients]);
                robotConnected = true;
            }
        }
    }
}
else
{
    //if any other IP connects, find first free socket
    for (int i = 0; i < socketCount; i++)
    {
        if (usedSockets[i] == false)
        {
            usedSockets[i] = true;
            clientSockets[i] = tempSocket;

            clientSockets[i].BeginReceive(rcv, 0, 1024,
SocketFlags.None, new AsyncCallback(Receiving), clientSockets[i]);
            clientIPs[i] =
clientSockets[i].RemoteEndPoint.ToString();
            connect_socket.BeginAccept(new AsyncCallback(Connecting),
connect_socket);
            addToDisplay(Color.LightGreen, "Client connected to
socket " + i + "\nClient IP (" + clientIPs[i] + ") index #" + i);

            break;
        }
    }
}
}

```

Connection method is introduced in the listing 9. In order to connect the robot to the correct socket, each connecting client's endpoint must be checked. If the IP matches the robot's IP, the temporary connection is moved to the *robot_socket* and each of the connected clients is informed of the connection to the robot by a message. If the client with any other IP is connected, the first free socket is filled.

Listing 10: Sending commands to the robot

```
//if F1 is pressed
if (e.KeyCode == Keys.F1)
{
    Array.Clear(snd, 0, snd.Length);
    //check current state
    if (isStopped == false)
    {
        isStopped = true;
        addToDisplay(Color.Yellow, "Stopping program...");
        //send stopping string for robot and other clients
        snd = Encoding.ASCII.GetBytes("STP");

        try
        {
            for (int allClients = 0; allClients < socketCount;
allClients++)
            {
                if (usedSockets[allClients] == true)//to each used
socket
                {
                    clientSockets[allClients].BeginSend(snd, 0,
snd.Length, SocketFlags.None, new AsyncCallback(Sending),
clientSockets[allClients]);
                }
            }
        }
        catch
        {
        }
    }
    else
    {
        Array.Clear(snd, 0, snd.Length);
        isStopped = false;
        //send run command for robot and other clients
        addToDisplay(Color.LightGreen, "Starting program...");
        snd = Encoding.ASCII.GetBytes("RUN");
        try
        {
            for (int allClients = 0; allClients < socketCount;
allClients++)
            {
                if (usedSockets[allClients] == true)//to each used
socket
                {
                    clientSockets[allClients].BeginSend(snd, 0,
```

```
snd.Length, SocketFlags.None, new AsyncCallback(Sending),
clientSockets[allClients]);
        }
    }
    catch
    {
    }
}
}
```

In listing 10 one example of Server Console's various key shortcuts can be seen . The status of the robot is checked and depending on it, the robot is either stopped or the program is continued. The commands sent to the robot and responds coming from the robot are also sent to the other clients to inform about the robot status.

The robot stops the current program if the connection to the server is lost. Client Console is an application, which can be connected to the server through the local area network and used to choose an executed program, pause and continue the program, emergency stop the robot, send any other input data to the controller, or to get important output data such as the current position, the program number, the code line, the speed or the latest machine vision camera string.

3.4 TicTacToe

The robot's capability to receive input via Ethernet inspired us to make an application using this feature. There has been previous research with the aim to remote control the robot, but no actual applications [2]. Therefore, a slightly different approach was chosen.

The idea was to create an online version of the traditional pencil-and-paper game Tic Tac Toe. According to the original purpose of the game as a promoting trick for Savonia to attract new students, the game was planned to be strongly bound to be under Savonia's name and logo. For example, the game could be a nice point of interest in the fairs, or TicTacToe client application could even be shared in the Savonia's homepages. The two web cameras would provide real time images of the ongoing game that could be viewed with the client software.

Making the robot to actually use a pencil to draw the moves on paper would have required so extensive arrangements that an alternative setup had to be created. The solution was to make the Xs and Os from pieces of plastic and move them onto the game board using the vacuum gripper. In the Figure 40, the robot is moving one of the pieces.

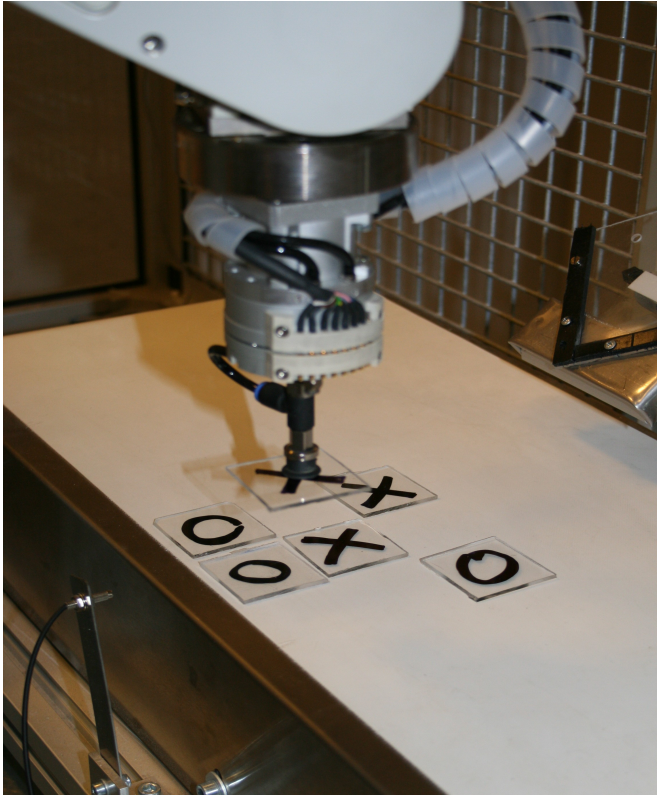


Figure 40: The robot playing TicTacToe

3.4.1 Description of the System

The basic idea of TicTacToe is to play Tic Tac Toe game against the robot over the Internet. The game is based on the client-server principle, where the game logic runs on a server PC and the human players operate as clients (See Figure 41). The clients can communicate with each other during the game and while queueing using the chat integrated to the game. The robot receives commands from the server, and while it has a special status co-operating with the server, technically, it is classified as a client, too. The networking functions are implemented using Sockets. The networking is executed asynchronously.

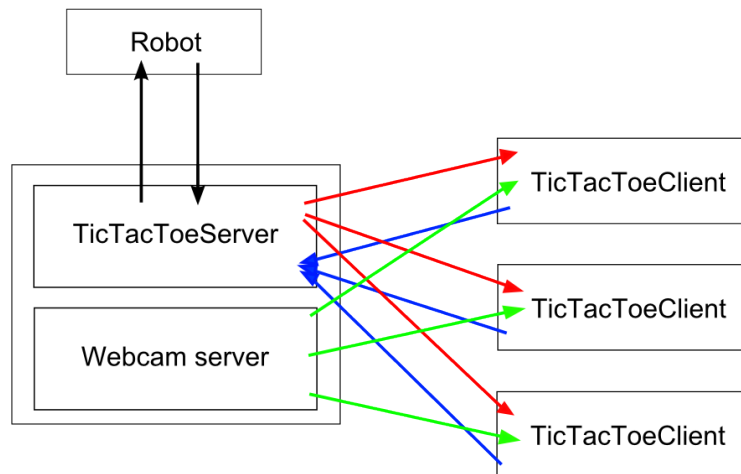


Figure 41: The connections between the game server, the web camera server, the robot and the clients.

Security was a very important aspect to take into consideration while planning the TicTacToe Data Transfer Protocol. The whole system is designed to be as fault immune as possible. All input received from user is verified. The moving of the robot is performed by sending the cell index of the game pallet to the robot, and the robot only moves between predefined positions. The player never gains free control of the robot's movements.

3.4.2 TicTacToe Data Transfer Protocol

A specific protocol was designed for the data transfer between the server and clients and the server and the robot. The protocol was designed to be as lightweight as possible yet user-friendly. All messages sent between the server and the client begin with a keyword followed by the actual data. The keywords are 4-digit-long real words such as “NICK” for a nick request/response. The data is coded as numbers or letters as in the message “GAME71147”, which means the robot moves to the cell no. 7, the robot (no. 1) wins, the winning line is through cells 1, 3 and 7. The full TicTacToe Data Transfer Protocol is described in Appendix 1. The most important thing in this application is obviously the security of the robot. There must not be a change for the player to move the robot directly.

3.4.3 TicTacToe Server

The TicTacToe Server software runs on the server PC in the robot work cell. It handles the client connections, the game AI and commands the robot to make the moves. TicTacToeServer uses asynchronous networking for handling the multiple simultaneous client connections.

3.4.3.1 Client Class

The server handles the clients as *Client* objects. A *Client* has the following public variables:

Socket clientsSocket, to which the established connection is moved from *connectSocket*

String nick, which stores the nick name the user wishes to use

Int index, which is the client's index in the *clientArray* array

IPAddress IP, which stores the client's IP address

String desiredChar, which stores the character the player wants to use

The *Client* class has two overloaded constructors. The first takes 3 parameters, a *string requestedNick*, an *int socketIndex* and an *IPAddress clientsIP*. The second takes just one parameter, the *int socketIndex*. The former is used when creating *Clients* normally for use, the latter when the *clientArray* is full and the *Client* is created only for sending the NICKFS message. In this case there is no need to give a nick name to the *Client*.

3.4.3.2 Creating Clients and Receiving Data

The server listens incoming connections with one socket, the *connectSocket*. When a new connection is established to the *connectSocket* with the NICK REQUEST message, the server creates a new *Client*, moves the connection from the *connectSocket* to the *Client's* socket and releases the *connectSocket* to listen new connections.

On startup, the server gets its own IP address using the *GetHostName()* function of the *Dns* class. Then the server creates a *connectSocket* and binds it with the local endpoint created using the server's IP. The *connectSocket* listens incoming connections in port 10003 using the socket's method *BeginAccept()*.

The method takes two parameters, an *AsyncCallback*, which basically tells what to do next and an *object* that is passed to the next method called by the *AsyncCallback*. The code for this is shown in Listing 11.

Listing 11: Accepting a client in TicTacToe server

```
connectSocket.BeginAccept(new AsyncCallback(OnClientConnect),
connectSocket);
```

The callback refers to the method *OnClientConnect*, which handles new connections as shown in Listing 12 below.

Listing 12: Moving the connection from connection socket

```
public void OnClientConnect(IAsyncResult iar)
{
    receiveSocket = (Socket)connectSocket.EndAccept(iar);
    receiveSocket.BeginReceive(receive, 0, receiveSize, Socket-
Flags.None, new AsyncCallback(ReceiveData), receiveSocket);
    connectSocket.BeginAccept(new AsyncCallback(OnClientConnect), con-
nectSocket);
}
```

As the first line in listing 12 shows, the method takes one parameter, the *IAsyncResult*. The result is the *object* passed in the *IAsyncCallback*, the *connectSocket*. The connection is moved to the *receiveSocket*, which starts to receive and then the *connectSocket* is released to receive the next incoming connection.

In the *receiveSocket*'s method *BeginReceive*, a new *AsyncCallback* is introduced, the *ReceiveData()* method. This method performs the actual receiving of the data sent by the new client-to-be. First, the method gets the number of characters sent and if it is more than zero, handles the assumed NICK REQUEST. The ignoring of the empty messages is done because the server for some reason receives also totally empty messages.

If the array is not full, the first four characters of the message are examined. If they are not NICK, as they should be according to the protocol, the connection is closed. Also other inspections are carried out. If they are all passed, a new *Client* object is created into the *clientArray* array as Listing 13 shows.

Listing 13: Creating a new client object

```

clientArray[index] = new Client(
desiredNick, index, IsolateIP(sendingClientSocket.RemoteEndPoint.ToString()));
clientArray[index].clientsSocket = (Socket)sendingClientSocket;

```

The *sendingClientSocket* was passed to this method and now it is finally moved to the *Client's* own socket. From this point on, the socket is handled only by referring to the index number of the *Client* in the *clientArray*.

The *IsolateIP()* function takes an *IPEndPoint* as a parameter and returns the IP Address. The function was written because it is not possible to get the IP address of the socket's connection otherwise. The function simplifies the action of separating the IP address and the port number.

After the creation of the new *Client*, the NICKS message is sent to the *Client*, a chat message informing about the new player is sent to all clients and the turn array is updated and the turn update sent to all clients before the *clientSocket* of the new *Client* is allowed to start receiving as shown in the next listing 14.

Listing 14: Starting the receiving from a client

```

UpdateTurnArray(true, index);
SendTurnUpdate();
clientArray[index].clientsSocket.BeginReceive(
receive, 0, receiveSize, SocketFlags.None, new AsyncCall-
back(ReceiveDataClient), clientArray[index]);

```

In the *BeginReceive* method there is a *AsyncCallback* for method *ReceiveDataClient*. The method handles the receiving of data from the *Clients* in the *clientArray*. To sum up:

ReceiveData is used to receive the data of the incoming new connections, once per client ("client" referring to any connection, not the *Client* object)

ReceiveDataClient is used to receive all data coming from the actual *Clients* created in the *clientArray* (ie. CHAT, GAME and EXIT messages)

The *ReceiveDataClient()* method receives the data similar to the *ReceiveData()* method, but in addition, it determines which message it is and takes the corresponding actions. If the incoming data is a chat message, the sender's nick is added to the message and the new message is sent to all *Clients*. If the message is a request for exit, the *Client's* socket is disconnected, the *Client* object deleted and the *turn* array updated and other clients informed about the change in the queue.

If the message is a game move, the first thing to examine is, whether the player wants to reset the game. If the next character (*resetString*) after the keyword GAME is “R”(for “Reset”), the player wishes to reset the game grid, which is allowed only, if there are no queuing players. This is checked using the *bool* type *CheckQueue()* function, which takes the current player's index number in the *clientArray* (*sendingClient.index*) as parameter and returns *true*, if there are others in the queue. If there are queuers and the game is over, the player is moved to the end of the queue, otherwise the robot is ordered to reset the board and to allow the next round for the current player.

If the player wishes to continue the game (*resetString* is “C” for “Continue”), the next character is parsed into integer. If the parsing is successful and the number is on the valid range, the player's symbol is inserted into the corresponding cell in the integer array *grid* which stores the actual game information and the robot is asked to perform the move on the board.

After the robot finishes performing the player's move, the *RobotsMove()* function is called. This function is the place “where the magic happens” in TicTacToeServer. The game's logic itself is the simplest possible, just a long set of *if..else if* statements examining the state of the *grid* array. The strategy is as follows:

1. *If there are two own marks and an empty cell in some row, fill the empty cell to win the game.*
2. *Otherwise, if there are two opponent's marks and an empty cell in some row, fill the empty cell to prevent the opponent from winning.*
3. *Otherwise, create a possibility to win in two ways (fork).*
4. *If the opponent is going to fork, block.*
5. *Play the center.*
6. *If there is the opponent's mark in the corner, play the opposite corner.*
7. *Play an empty corner.*
8. *Play an empty side.*

After the decision is made, the *grid* is updated and the move sent to the robot. When the robot sends the “READY” message, the robot's move is sent to the client.

3.4.3.3 Sending Data

The sending of the data is done similarly to the receiving. The actual function used in the code is a *void* type *SendToClient()*, which takes two parameters, an *integer* type *client* and the *string* type *message*. The function converts the *string* to *byte* array and calls the *Client's socket.BeginSend()* function as shown in the following example (Listing 15):

Listing 15: *Sending data to a client*

```
byte[] messagebyte = new byte[1024];
messagebyte = Encoding.Unicode.GetBytes(message);

clientArray[client].clientsSocket.BeginSend(messagebyte, 0, messagebyte.Length,
SocketFlags.None, new AsyncCallback(Sending), clientArray[client]);
```

The *AsyncCallback* calls the *Sending()* function which gets the *Client* in question and calls the *BeginReceive()* method.

3.4.3.4 Turn Management

The server allows the *Clients* to play in the order they have connected to the server. At first, the *Clients* are arranged in the *clientArray* in the same order that they are allowed to play, but if the players decide to leave before playing, the empty cell of the *clientArray* is given to a new connection. Therefore, the array cannot be used to determine, who is in turn to play.

The information of the order of the players is stored in the *int* type array *turn[]*. It contains the indexed of the *clientArray* in correct order. The array is rearranged using the *UpdateTurnArray ()* method. The *void* type method takes two arguments, the *bool* type *joining* and the *int* type *socketNumber*. The first one tells whether the method is called on the creation of a new *Client* or when a *Client* has left.

If the *joining* is *true*, the function searches the first empty cell from the *turn* array and inserts the *socketNumber* into that cell. If the *joining* is *false*, the function searches the *socketNumber* from the array and replaces it with a zero. Then the array is rearranged to remove the zeros between used cells. This is done by searching for zeros and if they are found, moving the next cell's contents into that cell.

Each time the *turn* array is updated, the information of the queue length is sent to all *Clients*. This is done with the *void* type *SendTurnUpdate()* method. The method is a straightforward *for* loop, which calculates the queue length for each *Client* using the *QueueLength()* function and sends either TURNT or TURNF message to the *Client*.

3.4.3.5 Chat

As mentioned, there is a chat functionality in the game. Most of the operations are carried out in the client software, such as changing the text color. The server only forwards the chat messages the players send to all clients. Before sending the message, the sender's nick is added to the message as shown in Listing 16:

Listing 16: *The sender's nick is added to the chat message*

```
if ( receivedString.Substring(0, 4).ToUpper() == "CHAT" )
{
    ConsoleWrite(informColor, receivedString);
    SendToAll("CHAT" + sendingClient.nick + " says: " +
receivedString.Substring(4, receivedString.Length - 4));
    ConsoleWrite(informColor, "CHAT message send");
}
```

When a new player enters the server, two chat messages are sent: The notification about the new player to all players and a welcome message to the new client only (Listing 17):

Listing 17: *Sending the welcome message to the new player and notifying other is done after the "NICKS" message*

```
SendToClient(index, "NICKS");
SendToClient(index, welcomeText);
SendToAll(desiredNick + " joined the game");
UpdateTurnArray(true, index);
```

3.4.4 TicTacToe Client

TicTacToe Client is the software the player uses to connect to the TicTacToeServer and to play the game. While planning the network game, we listed some perks to be applied in our client program, because it is most important to make the game experience as trouble-free and seamless as possible. As a social feature, we decided to add a chat to the game, because waiting for one's own turn in line may bore some users. The client program was intended to be light, easy to use without unnecessary controls and graphically neat and personable. At first it was planned to write the application with Java to make the client program lighter and possible to play with the web browser, but time limitations forced to use C# also in this project, because the language was known quite well already.

3.4.4.1 Description of the Operation

Client program is split into three forms; login form, game form and video form. In the login form, the user inputs the required information – the user is prompted a nick name and character (whether X or O), then the client program attempts to connect to the server. The program checks if the nick name textbox is empty or nick has invalid characters and returns a failure message according to cause before trying to connect.

On the server side, the cause to disconnect from the server is sent in NICKF message. Connection is closed if the player's name is already in use, the server is full or the player is banned from the server. If the player is granted a pass to the server, the login form closes and the game window opens up. The game form contains only chat when there are other players on the line before, and when there is nobody else, game controls appear on the right side of the window.

The game is played with a simple game board that has nine squares used for input and to show game status, and a message box that informs whether it is the robot's or player's turn or who has won the game. The client itself doesn't contain any game logics, it receives the robot moves and the game status from the server machine. If the

player is the only one in the server, it is possible to reset the game and try again until somebody else connects to the server. The chat function sends a message to the server and the server sends a copy to all players, thus players can interact with each other. The game form contains also “View Robot” button, which opens a new form that shows the robot through the two web cameras of the robot work cell. There are also a few different themes to change the appearance of the game and make it personal as shown in Figure 42. The themes can be accessed by right clicking the form to open the context menu.

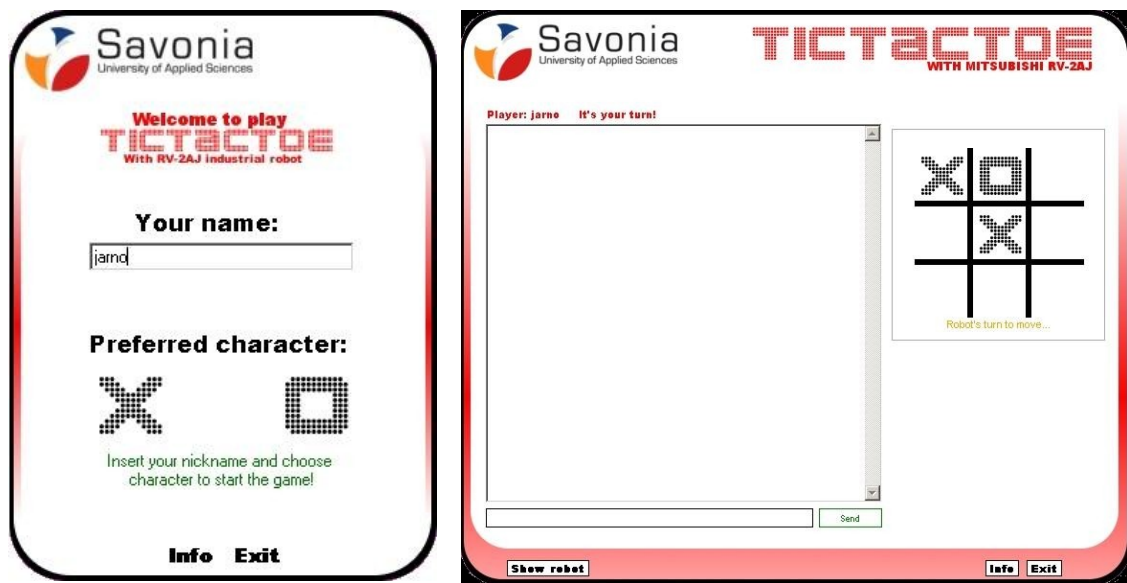


Figure 42: Client application's login and game forms

3.4.4.2 Networking

Data transfer over network is needed in four tasks, sending the player's name and character to the server, using chat, sending game moves and transferring video feed to players. To achieve smooth and reliable data transfer, it was decided to use TCP/IP protocol in the applications. With Visual Studio 2008 and .NET framework 3.5, it is easy to create connections by using sockets with a little familiarizing with the *System.Net* and *System.Net.Sockets* statements. In order to rapidly send messages to the server, the socket should be streaming the data. At first, it was studied how to use sockets with the synchronous method in the console application, but as it was tried to communicate with a window based application, it was found out that the synchronous method cannot handle acceptance of the sockets and receiving of data at the same time.

Also multiple sockets are not supported. The asynchronous method offered a worthwhile approach to get used to socket programming and with this know-how, almost any socket based network application can be programmed.

There are two server applications that the client needs to connect, the first is the game server and the second one is the web camera server application. It was decided that these should be separate programs, as the camera software was thought to be using UDP protocol. UDP is a fast way in moving large data packets such as video stream, but it is not as reliable as TCP/IP, as UDP is a connectionless protocol. Also, there were similar connecting problems using UDP protocol, as there was with Robot Camera application.

First, the client program connects to the server when the player has typed a nick and presses the desired character. At the beginning of the project, the client program connected to the server straight at startup, it was noticed that this was not a practical way, because the connection socket of the server machine was reserved for one client at a time and other players had to wait for the release of the connection socket. Also the possibility of creating multiple connection sockets was examined, but it was not very reasonable in a server consisting of sockets for only ten players.

The connection to the server is continuous and the clients only disconnect if the exit button is pressed, the server is shutdown or the player is kicked from the server. Data is transferred based on our own “protocol” consisting of keywords and parameters tied in them. Simple command words make the data flow easy to understand and allow users even plan their own client program interface which to play with. The data from the client was originally meant to be sent numerically to minimize the amount of network traffic, but to make it simple to understand the code, keywords were changed to four character keycodes.

Client program sends only NICK, CHAT, GAME and EXIT words in order to communicate with the server. With NICK marker, the player sends his character and name to the server. CHAT works as a marker for a chat message, and the only data moved with it is the message from the player. GAME means the player’s game move and sends also information if the player is trying to reset the game. EXIT is sent to the

server to inform that the player is shutting down the client program. The commands between the server and the client are encoded in Unicode character mapping to allow the sending of Scandinavian letters.

The video server is connected, when the player chooses to open the stream by clicking the View Robot button. This is good because the bandwidth is not used unnecessarily, if the user doesn't want or need to look at the robot video all the time. The video window connects to the video server and begins to receive images in an interval that is set to the server PC. There are multiple issues affecting the frame rate of the video stream, but the resolution of the feed is the major cause of lag.

A client can receive video from two cameras, if the server is set to capture both simultaneously. This may also cause some delay between the images. The camera is selected with *View Camera #1* and *View Camera #2* buttons of the video form (Figure 43). These buttons send the server whether "1" or "2" to request another feed. It was decided to send only one stream at a time to spare bandwidth. A single error picture is streamed to the client, if the server can't find the cameras, or it crashes to an exception.



Figure 43: Client video window

3.4.4.3 The Structure of the Client Program

The game form and the login form are created in the startup. Initially, the game form is hidden and the login form is shown. This had to be done with *ShowDialog()* and *Hide()* commands, because forms could not be hidden with *Show()*. The load event of the login forms uses *System.IO* statement's *StreamReader* to read the last connection settings (IP and port) and the player's name from the settings file. This was added later to make it easier to test the program in other computers, the earlier version had to be opened in the Visual Studio if there was a need to change the settings. The login form is used also for inserting the user name and character; it doesn't connect to the server application, it just calls parent's *getLoginInformation()* function to pass the data to the game form with public variables. If the name has special characters (i.e. @, /, \ or *), they are removed in the login form's code to prevent possible malfunction of the server. If the name is correct, information is submitted with "X" or "O" button.

When *GetLoginInformation()* is called, the game form starts to connect in the *ConnectToServer()* function. First, *ConnectToServer()* creates remote endpoint *iep* and declares socket *connect_socket*. In TCP/IP transfer, *SocketType.Stream* is used. *Connect_socket*'s options are also changed to not to linger after disconnect, to reuse the address after disconnect, and to set lower time to live for IP. These are done to make reconnection easy if the previous connection to the server has lost. Then *connect_socket* is beginning an asynchronous connection to the server with *BeginConnect()*, which refers to connect to *iep* and to move on to the *OnConnected()* function when the server is found.

Then the client sends a NICK message provided with user data consisted of whether "X" or "O" and player nickname. Message is sent as soon as the connection is established. If the server is full or the player's name is already in use, the server sends NICKF and the client closes the connection to the server. As the server responds with NICKS, the application's login form is closed and the game form is shown. The connection socket is now receiving data all the time and every action coming from the server is parsed in the *OnReceive()* asynchronous function. The client bypasses everything else than the words beginning with "TURN", "INFO", "GAME" and "CHAT".

To allow the *Receiving()* function to change the values of the form components without causing cross thread exceptions, delegates had to be added and functions with invokes had to be created to modify each property of elements. There are several functions to handle game functions. For changing the *Text* and *ForeColor* properties of the login form's status box there is *ChangeStatus()*. *AddNewMessage()* is for adding the new chat messages and for changing the chat colour in order to make the conversation easier to understand. For adding a new player message and changing its colour there is *AddPlayerMessage()*, and *HideGameboard()* is for hiding and showing game controls.

The state message of the login form can be changed using the game form, and this function is used to inform the player of denied access and cause of it when the game form receives a negative answer from the server. When the server grants an access to the connecting client, the game form is shown and the login form is closed. When the player's turns are updated, the game's queue status can be seen in the information box right up ahead of the chat box. Because of the occurring cross thread exception in *Receiving()* function, the game form has also delegates for some objects.

3.5 Modification of the Robot Work Cell

During the study, the usability of the robot work cell was improved by building a larger worktop to replace the turning aluminum table shown in Figure 44. The existing additional axis was not very usable because it swung freely several millimeters to both directions of its set position. It was due to the servo motor's gear mechanism which was loose. Another disadvantage was that the size of the plate was not as large as the robot's movements would allow and the useful area was only one third of the plate. When the utilization benefits of the machine vision camera and the solid worktop were realized, it was decided to build a completely new worktop.

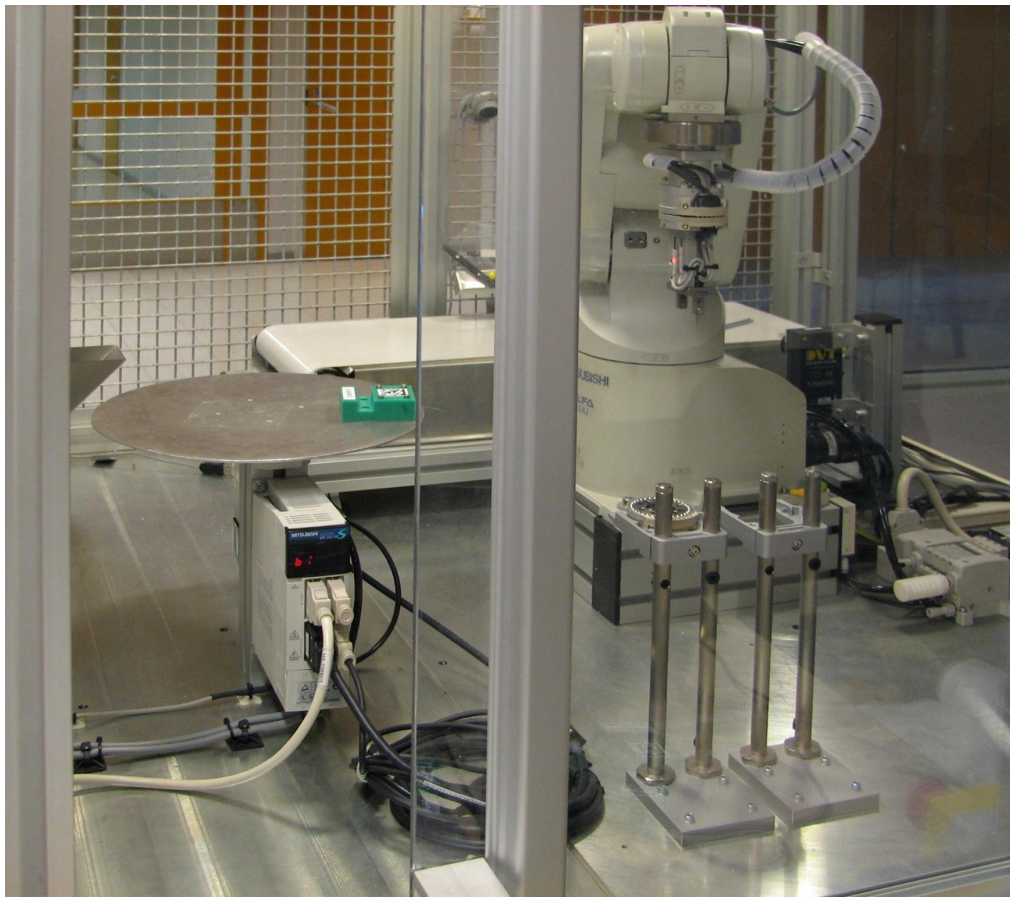


Figure 44: The side view of the work cell before modification.

The grayscale machine vision camera was moved above the new worktop. The camera's original location (See figure 45) on the side of the conveyor didn't allow many practical applications. When located in the ceiling above the worktop, the camera was intended to be used in co-operation with the color camera in versatile assembly tasks and other

solutions. It was mounted on the ceiling with a similar rack as the color camera. The aluminium parts of the original structure were used in the new rack and new wirings for data transfer and power were drawn unobtrusively through the workcell's wall and ceiling profiles. The rack can be moved into any location above the table, only the reach of the robot arm is relevant.

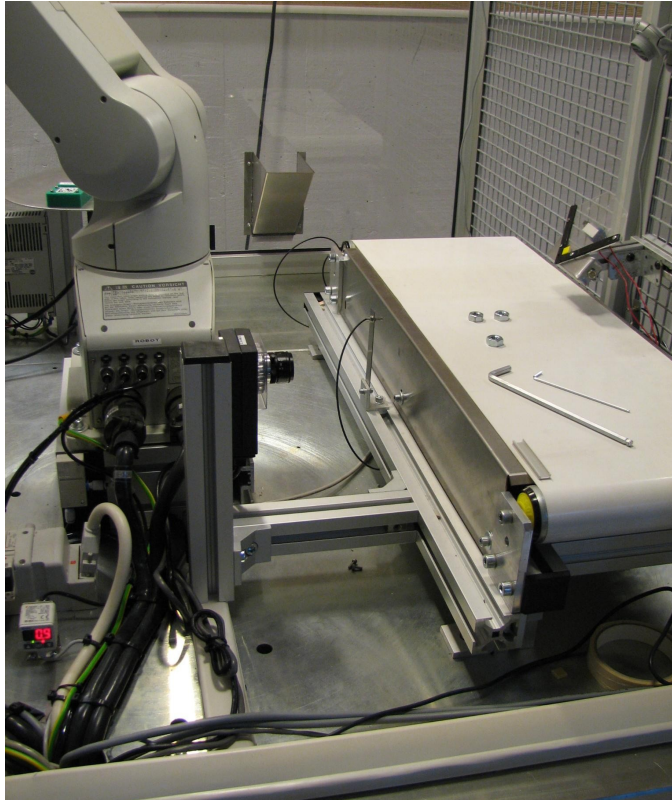


Figure 45: The original position of the grey scale camera.

The chassis of the table was assembled from 30×30 mm aluminium profile. The rack was anchored to the two threaded holes in the steel bottom of the workcell. Also the tool racks were brought closer to the robot. The racks were attached to the legs of the new worktop using the same 30 mm profile. No existing structures were disassembled except the aluminium plate, which can be put back to its place easily with 4 screws. The robot's additional axis settings were not altered to make it possible to take the original equipment into use quickly. The top of the table was made of plywood, which was painted white with semigloss paint. The plywood was inlaid into the aluminium profile's recess that was milled to the correct width. The new worktop is shown in Figure 46.



Figure 46: The new worktop installed.

3.6 Demo of the Robot Work Cell

3.6.1 Planning and Problems

While we were working on the robot in the lobby, many groups of visitors passed by and each time the absence of even the simplest demonstration was overwhelming. The “Robot Work Cell Demo” was designed to illustrate the operations of the robot work cell.

At first the demo was ment to be more complex, using all the equipment in the work cell. It soon became obvious that the machine vision system was far too unstable to be used in a system that should create a positive impression of Savonia UAS. Also the available working area on the conveyor limited the possible applications tremendously. A decision was made that the demo should most definitely be something that is less stunning technically but that absolutely works every time rather than something that has gorgeous details and that suffers from chonical malfunction.

In the original demo, a Savonia UAS logo was ment to be assembled from pieces brought in random order and angles by the conveyor. The demo should have used all the equipment in the work cell. Each piece would have been identified using machine vision and placed in a correct position onto the side table. Along with the robot operation, a slide show should have been presented to give the viewer a better idea of the interoperation of the devices which is invisible to the audience. Another problem with this type of actions would have been the returning of the pieces after the assembling.

The pieces could have been lifted back to the conveyor either by the robot or the viewer. If the robot lifted the pieces, the idea of randomness of their position would water down. Explaining, how the positions are actually random and different every time would not quite help. On the other hand, if the pieces were lifted into the waste chute for the viewer to lift back to the conveyor, the randomness would be a problem. The pieces could end up upside down or off the conveyor, if not totally lost.

Therefore, the operation had to be simplified. The use of machine vision had to be forgotten. In addition to the changing lighting conditions, the cameras themselves formed a problem. Their operation was so slow that the total length of the demo would have been closer to half an hour which was considered to be too much. The major issue was, however, the unreliability of the camera connection. At times, the camera lost its firmware during operation.

Leaving the machine vision out meant that the conveyor could not be used either. The small working area on the conveyor limited the possibilities tremendously. The only thing remaining from the original concept was the tool change. The actual work performed by the robot was planned to be simple moving of plastic pieces between two pallets, one on the conveyor and the other on the table.

3.6.2 Explanation of the Operation

When the demo program is started, the robot checks its tool and changes it if necessary. Then it waits for the slide show to send the start signal. At the beginning of the demo the robot introduces its hand gripper and the difference between linear and joint interpolated movement. Then it moves the plastic pieces from the table onto the conveyor.

The pieces are not lifted directly to the pallet but first into a special location outside the pallet. From there, the first piece is lifted into the pallet's first cell. The second piece is lifted to the additional position, the previous piece to the second pallet cell and then the new piece into the pallet's first cell. This action was designed to make the operation last longer to match the slide show. After lifting all the pieces to the conveyor, the robot changes its tool to a vacuum gripper. With it, the robot lifts the pieces back to the table via the rack used for correcting the mosaic tile angle.

3.6.3 Slide Show

During the actions described above, the operations are explained with a slide show displayed by a Windows application which communicates with the robot program. In the beginning, and after a complete cycle, the software is stopped to a state, where a "Welcome" slide is shown. The user can start the demonstration by selecting the desired language.

The slide show displaying information about the operation was written in C# as a Windows Forms application. The requirement of interaction between the robot program and the slide show ruled out the possibility to use a presentation software (e.g. PowerPoint). The slide show signals the robot to begin the demonstration when needed and the robot asks the slides to be displayed in correct stages of its program.

The slide show is displayed in fullscreen mode for two reasons. Firstly, it of course looks better without any window borders or the Windows' taskbar. Secondly, the user has a mouse to signal the software to start the demonstration. If the taskbar was shown,

an evil-intentioned viewer would be able to close the slide show and use the computer freely. The fullscreen mode can be exited only with a keypress. The keyboard is located in the cabinet out of viewer's sight.

Creating fullscreen applications is not quite straight-forward. Maximizing the window obviously is not enough, it leaves the titlebar and the taskbar visible. Actually, not even the use of Win32 API service to find and hide the taskbar (which in fact is just a window) makes the application truly fullscreen, because the window will not occupy the area of the hidden taskbar. [57] The correct way of making a fullscreen application is to use the whole primary monitor area. This is done by setting the window size explicitly and thus automatically covering the taskbar as Listing 18 below shows. [58]

Listing 18: *Setting the window size according to the primary monitor resolution*

```
[DllImport("user32.dll", EntryPoint = "GetSystemMetrics")]
public static extern int GetSystemMetrics(int which);

[DllImport("user32.dll")]
public static extern void
    SetWindowPos(IntPtr hwnd, IntPtr hwndInsertAfter,
        int X, int Y, int width, int height, uint
flags);

private const int SM_CXSCREEN = 0;
private const int SM_CYSCREEN = 1;

private static IntPtr HWND_TOP = IntPtr.Zero;
private const int SWP_SHOWWINDOW = 64; // 0x0040

public static int ScreenX = GetSystemMetrics(SM_CXSCREEN);
public static int ScreenY = GetSystemMetrics(SM_CYSCREEN);

public static void SetWinFullScreen(IntPtr hwnd)
{
    SetWindowPos(hwnd, HWND_TOP, 0, 0, ScreenX, ScreenY,
SWP_SHOWWINDOW);
}
```

The *User32.dll* DLL is used to fetch the monitor resolution with the *GetSystemMetrics()* method. The x and y resolutions are referred to as 0 and 1 respectively. These values are stored to the private constants *SM_CXSCREEN* and *SM_CYSCREEN* in the beginning, because the parameter for the *GetSystemMetrics()* must be a constant.

The window is made fullscreen with the *SetWindowPos()* method. *IntPtr* parameter is a “platform specific type used to represent a pointer or a handle” [54]. The *IntPtr* is an integer and its size depends on the hardware and operating system on which it is used (i.e. 32 bits on a 32-bit platform and 64 bits on a 64-bit platform). [59]

The slide show window is maximized and restored with the *Maximize()* and *Restore()* functions. The functions take the Form *thisForm* as a parameter. See the *Maximize()* function in Listing 19. The content of the *Restore()* function is similar.

Listing 19: *The function for maximizing the window*

```
public void Maximize(Form thisForm)
{
    if ( !IsMaximized )
    {
        IsMaximized = true;
        thisForm.WindowState = FormWindowState.Maximized;
        thisForm.FormBorderStyle = FormBorderStyle.None;
        thisForm.TopMost = true;
        WinApi.SetWinFullScreen(thisForm.Handle);
    }
}
```

The restoring is done when the key combination CTRL+R is pressed. The maximizing is done when loading the form and when the key combination CTRL+F is pressed. The examination whether two keys are being pressed simultaneously is done with the *KeyDown* event handler (Listing 20):

Listing 20: *The event handler for keyboard input*

```
private void Demo_KeyDown(object sender, KeyEventArgs e)
{
    if (
        ( e.KeyCode == Keys.F ) &&
        ( e.Modifiers == Keys.Control ) )
    {
        Maximize(this);
    }
    if (
        ( e.KeyCode == Keys.R ) &&
        ( e.Modifiers == Keys.Control ) )
    {
        Restore(this);
    }
}
```

3.6.4 Graphics

The slides are simply images that are displayed as the background image of the form. The resolution of the images is optimized for the display used (i.e. 1280×1024). The choice of using images was done because of the reliability and simplicity of updating the slide content. Other possibilities were to use typical Windows Forms controls (such as *Labels* and *Buttons*) or the *WebBrowser* control. The use of Windows Forms controls would have meant that the slide show content could only be altered by changing the source code of the application. The use of the *WebBrowser* control was thought to be a solution, because of the use of standard HTML documents that anyone could create and substitute easily since they are not inside the application itself. However, when the *WebBrowser* was displayed in fullscreen, the application did not accept any keyboard input.

The background image of the slide consists of a group of 3-dimensional Savonia UAS logos as shown in Figure 47. The logo material resembles colored acrylic which is semitransparent and glossy. The logos were modelled with Blender3D open source 3D-modelling software.



Figure 47: The first slide of the robot work cell demonstration slide show.

3.6.5 Networking

The networking between the robot program and the slide show is implemented in the same manner as in other applications created for the robot work cell (e.g. TicTacToe). The PC software operates as a server and the robot as a client. In this case, asynchronous networking is not used because of the need to handle multiple client connections, but only to be able to re-establish the connection to the robot if it is lost during the operation.

3.6.6 Robot Program

First, the program for the robot checks which tool is initially attached in the beginning and changes it to hand if needed. This is done using the *M_TOOL* parameter, it returns the currently used tool from the controller's memory. The value of the parameter is changed to match the tool number after each tool change procedure. In the beginning, the robot poses for the audience and shows the difference between joint interpolation and linear interpolation.

Then the program gets each of the green plastic objects and moves them to the conveyor. As an addition to increase the delay between slides, the robot arranges pieces on the conveyor. Then the tool change subroutine is executed and the robot moves near the tool racks through the help position located above the work top. The tool is changed to the vacuum gripper. After this, the robot picks objects and lifts them to the mechanical structure that is used to center the pieces. After this, objects are moved back to the work top and the tool changed back to the hand gripper.

The program has INIT, SET, GET and CH subprograms. INIT subprogram initializes the variables, defines the pallet coordinates and what is the most important, creates connection between the robot controller and the server PC running the slide show. If connection is lost, program is interrupted and halted. SET subprograms are used to move the plastic objects to the conveyor using hand, CH subprogram calls the tool change procedure and GET moves the pieces back to the worktop with vacuum gripper.

3.6.7 Hardware Configuration

A TFT display was mounted outside of the robot work cell for the demonstration slide show. A special rack was designed and manufactured for the display. The rack was built from 40×80 mm aluminium strut profile, two swivel bearings and a supporting arm joint. A small table is mounted below the display for the mouse. The length of the rack is 500 mm. The display can be turned around the corner of the work cell to make possible to view the slide show from different sides of the work cell as shown in Figure 48. The display can also be used when working with the work cell PC.



Figure 48: The display rack mounted to the corner pillar of the work cell

4 Results

This study had three distinct goals: replacement of the servo position data encoder batteries and recalibration of the robot's coordinates, creation of student exercises about the robot work cell and the development of the operations of the robot work cell. As an additional task, there was a wish, that the robot work cell's web cameras would be put into operation. All these goals were achieved.

During the work, also three additional objectives were set: a TicTacToe online game played against the robot, structural modification of the work cell and the demonstration of the operations of the work cell. Also these goals were achieved.

4.1 Results of the Main Objectives

4.1.1 Battery Change and Calibration

The battery change and the calibration were carried out successfully. The robot origin data was recorded using the mechanical stopper method. The origin data was written to the table located inside the cover of the robot's base as it is meant to according to the Mitsubishi's instructions.

4.1.2 Exercises

A student exercise handout was written from the total of 20 student exercises developed (Appendix 2). These exercises start from the same basics as the existing exercises, but they go far further with their advanced programming and complex machine vision exercises. Along with a large amount of documentation about the robot work cell, the exercise handout can be used as the main study material about the robot work cell.

The most essential instructions were included into the exercise handout. Some instructions that are not needed for completing the exercises were collected as additional instructions to the end of the exercise handout. These include instructions for advanced features of the programming software.

Example solutions of the exercises were developed. These include DVT Intellect systems, Melfa-BASIC programs and E-designer projects. The exam created for the course measures the student's knowhow about the handout's content (Appendix 3).

4.1.3 The Web Cameras

No software was found capable to capture and send the video stream out of the work cell. The tested software were either too unstable or too resource consuming. The problem was solved by creating an image capturing software that is also able to send images using FTP.

The web cameras were taken into use successfully by writing two separate applications for sending and receiving video. Robot Camera software sends real time video to the Camera Viewer program and still pictures to the web server. Camera server sends both camera feeds properly and it has been tested and stated to be fully functional in the local area network, but the option to request IP address from DNS server has not been tested yet.

4.2 Results of the Additional Goals

4.2.1 TicTacToe

The first additional goal was the TicTacToe game, a network version of the traditional pencil-and-paper game played against the robot. In the project, a game server and client software were created using C# and .NET 3.5 framework. The game was tested in the local network and verified to be operational. While playing, the player can also view the robot make the moves in the work cell using a built-in web cam viewer.

A lot of planned features had to be removed due to problems that were left unsolved because of the shortage of time. These include the server's logging, IP address based banning and settings functionality.

Logging would have been a helpful feature in possible troubleshooting. The log files were designed to be written using the *StreamWriter* class. Five different log files (system, game, chat, connections and error logs) and corresponding functions for writing them were designed. All the log files were planned to be opened at startup and kept open while the server is running. This is less resource consuming than opening them every time something needs to be written. The logging was designed to be done by writing the actions into a file as plain text and converted into a viewable format (e.g. HTML) using an external software. A single event written into file as a line of text would have consisted a timestamp and other information varying depending on the log file.

The IP address based banning was ment to be a way to minimize the effects of possible vandalism. In the original concept, the IP addresses were stored into a text file using *Ban()* and *Unban()* functions that add and remove the IP addresses into or from the file and keep the file organised. The file containing the banned IP addresses could have been viewed and modified in the settings window of the server.

A wide range of settings were designed for customizing the appearance and operation of the server. Controls for things such as the server's name, chat's welcome text, log file

locations and the colors used in the GUI were collected into the settings form. The usage of these properties would require only the creation of a settings file and appropriate read and save functions for its use.

The log file writing, IP banning functions and the settings form were left to the source code of the game. Thus, these features would be relatively easy to take into use in an updated version of the server. These are all additional features that would have improved the usability of the server. The actual game-related features are functional.

The intended use via Internet would require a way to reach the robot work cell's PC from outside the school's network. Before deploying the application it would require more testing and further development.

4.2.2 Modification of the Robot Work Cell

The robot work cell itself was modified to allow more complex applications to be developed. In the modification a new work top was built into the work cell and the grey scale machine vision sensor mounted to the ceiling above it. The toolracks were moved to make space for the work top.

The turning round plate that was operated by the robot's additional axis interface was replaced by a larger, solid worktop. The additional axis unit was left onto its original place under the new table. It was also left connected to the robot controller. This way it can easily be taken into use if needed in the future. The new solid worktop has already shown its superiority over the old labile turntable with its larger working area.

The gray scale machine vision camera was moved from its original position at the side of the conveyor and mounted into the ceiling above the new worktop. When the camera was tested with the Intellect software, it was noticed that some dark smudges can be detected in the images. These smudges were disrupting the calibrating operation, and it was impossible to set the coordinates and the real world scale for the camera.

The original plan was to build the whole table from aluminium. However, the surface of the table had to be made of plywood. When starting to make applications that utilize the new work top, it was discovered that the plywood plate was crooked. Therefore, the table had to be left to the current state. The height difference between the edges and the center of the table is close to 1 cm and it makes the teaching of positions difficult. The problem is currently handled with an additional piece of particle board on top of the plywood surface but it is obvious that the plywood should be replaced.

4.2.3 Demo of the Robot Work Cell

Robot Work Cell Demo has already been shown to two groups of visiting upper secondary school students. The demo program has been tested extensively and there has not been major imperfections in neither the robot's program or the PC application. The instructions for starting the robot work cell and the demo program were written to allow anyone to start the demo program without actual know-how of the work cell's devices.

4.3 Development ideas

During the study, many possibilities for improvement were noticed in the robot work cell. However, the scope of this project did not allow all the changes to be carried out. This chapter points out the things that should be considered if the robot work cell is developed further in the future.

It is suggested that the conveyor in the work cell is moved closer to the robot to maximize the area the robot is able to reach. Currently, the robot can only use about one third of the total width of the conveyor belt.

When working with the machine vision exercises, the acceleration of the conveyor formed a problem because objects rolled on the conveyor. A way for controlling the speed of the conveyor would remove this problem and allow versatile objects to be used.

Setting the forbidden areas for the robot would improve the security of the robot. Total of 8 forbidden planes can be defined. Together with the tool parameters set in this work, the planes would prevent collisions between the robot and the other equipment in the work cell.

The surface of the new work top could be replaced with a plate made of semi-transparent material. With a light source installed below the work top, a back-lit machine vision system could be created using the grey scale machine vision camera.

To make it possible to use the machine vision system more reliably in the challenging lighting conditions of the corridor, curtains around the work cell would reduce the amount of ambient light significantly. Also two powerful light sources on both sides of the conveyor would reduce the amount of disturbing shadows. Sharper images would improve the results.

Currently the doors of the work cell have switches that stop the robot and turn off the controller if the doors are opened during operation. This of course makes the work cell safe for human users but can cause problems in robot operation. For example if the doors are opened during tool change, the new length of the tool may not have been changed to the tool parameters. This may cause the robot to collide with the environment. Also, if the robot is stopped to a precise position and the program must be restarted, the controller tries to move the robot to the initial position regardless of possible obstacles around the current position. This could be prevented by changing the whole point of view in the safety setup. Instead of stopping the robot when a human moves to its reach area, the whole approach could be made impossible. In other words, the robot would be protected from humans rather than humans from the robot. This could be done by changing the door's safety switches into solenoid locks. With them, the robot could keep the doors locked during its operation.

OpenCV was decided to be used only in capturing the web camera images without analyzing or processing them, but this interface still offers a good base for further development of the program. OpenCV offers various interesting attributes to be used in the robot work cell. For instance, there are some human-computer interface elements (HCI), object identification, optical character recognition, stereo vision and motion tracking functions. They would allow augmented reality features to be added to the outgoing video.

Also the TicTacToe game could be enhanced with augmented reality elements with minor changes in the work space. The piece the robot moves from the stock pallet and its destination area could be located and marked in the video. Also the winning line could be highlighted.

A network version of the robot work cell demonstration could be made with the possibility for the viewer to affect the actions performed by the robot. There was also a plan to gather data of the work cell to be shown in the web pages. Information could be downloaded from the integrated FTP server of the E1070 operation terminal. Sending it to the web page could be integrated in the Robot Camera program.

5 Conclusions

The project was carried out successfully. All the main objectives were achieved. The exercises designed in this project are versatile and their degree of difficulty ranges from very easy to demanding, thus making them suitable for students with varying skills and motivation.

The additional goals were achieved. The programs were tested to be mostly functional. Still, this project leaves opportunities for the additional development of the software, because some of the features were left disabled due to inadequate testing.

The search of information was eased by gathering basic instructions into the exercises handout. Some additional documentation was also collected into a separate document. Together they should make the use of the work cell a lot easier.

We hope that the robot work cell will now be a more interesting project for the courses in the future and the exercises and the related instructions will offer an easy approach to industrial robotics. We also hope that in the future, our ideas for further development will be noted and the work cell will be made even more efficient.

This work has been extremely interesting and educational. It has taught us a lot about robotics, both theory and practice. Working with the devices of the work cell and the related software has given an insight to the planning and programming of the working robot system. The study has been independent, which has given both freedom for experimentation and responsibility for the results.

6 References

- [1] Korhonen, Mika, *Robotisoidun ja konenäöllä varustetun tuotantosolun harjoitusten kehittäminen*, Thesis, Savonia University of Applied Sciences, 2008, 44 pages.
- [2] Kauppinen, Anssi, *Robotilla ja konenäöllä varustetun tuotantosolun harjoitusten ja Internet-käyttöliittymän kehittäminen*, Thesis, Savonia University of Applied Sciences, 2008, 39 pages.
- [3] Gera, D. L., *Ancient Greek ideas on speech, language, and civilization* (2003). [www-document]. Available: <http://books.google.com/books?id=h5tKJvApybsC&pg=PA114&lpg=PA114&dq=hephaestus+handmaidens&source=web&ots=AmE4CYagER&sig=qoE-R-FGa3CRe9fKPjBKCdk24C4#v=onepage&q=hephaestus%20handmaidens&f=false> (Searched 10.11.2009)
- [4] Leonardo3 Official Website 2010, *Leonardo Da Vinci's Robots*. [www-document]. Available: <http://www.leonardo3.net/leonardo/books%20I%20robot%20di%20Leonardo%20-%20Taddei%20Mario%20-%20english%20Leonardo%20robots%201.html> (Searched 10.11.2009)
- [5] Wired Magazine, *The Real Da Vinci Code* (2004). [www-document]. Available: <http://www.wired.com/wired/archive/12.11/davinci.html> (Searched 10.11.2009)
- [6] Suomen automaatioseura ry, *Robotit*. [www-document]. Available: <http://www.automatioseura.fi/index/tiedostot/Robotit.pdf> (Searched 10.11.2009)
- [7] Fu, K.S., Gonzalez, R. C., Lee, C. S. G., *Robotics*. Singapore: McGraw-Hill Co., 1987, 580 pages
- [8] University of Texas at Austin, Robotic Research Group, *History*. [www-document]. Available: http://www.robotics.utexas.edu/rrg/learn_more/history/ (Searched 03.11.2009)
- [9] Wikimedia Commons, *Unimate*. [Image]. Available: <http://upload.wikimedia.org/wikipedia/commons/7/7f/Unimate.jpg> (Searched 22.2.2010)
- [10] Britannica Encyclopedia, *Robot* (2010). [www-document]. Available: <http://www.britannica.com/EBchecked/topic/505818/robot/248208/Industrial-robots#ref=ref849962> (Searched 03.11.2009)
- [11] Webopedia, *Magnetic drum* (2001). [www-document]. Available: http://www.webopedia.com/TERM/m/magnetic_drum.html (Searched 03.11.2009)
- [12] Carnegie Mellon University, *The Robot Hall of Fame*. [www-document]. Available: <http://www.robothalloffame.org/unimate.html> (Searched 07.11.2009)
- [13] Matthias Rauterberg, *Unimate (1961)*. [www-document]. Available: <http://www.idemployee.id.tue.nl/g.w.m.rauterberg/presentations/hci-history/sld090.htm> (Searched 07.11.2009)
- [14] Computer History Museum, *Timeline of Computer History* (2006). [www-document]. Available: <http://www.computerhistory.org/timeline/?category=rai> (Searched 08.11.2009)

- [15] Kurfess, T. R., *Robotics and automation handbook* (2005). [www-document]. Available: http://books.google.fi/books?id=stlWUpWvI94C&pg=PT21&lpg=PT21&dq=stanford+arm&source=bl&ots=sjsd7aBopW&sig=Uo5rp2u9TKNatKkp--B1jHWovVg&hl=fi&ei=gOr7SrjQE5DS-QalvOWUAg&sa=X&oi=book_result&ct=result&resnum=10&ved=0CC8Q6AEwCTgU#v=onepage&q=stanford%20arm&f=false (Searched 12.11.2009)
- [16] Stanford University, *Robot*. [www-document]. Available: <http://infolab.stanford.edu/pub/voy/museum/pictures/display/1-Robot.htm> (Searched 12.11.2009)
- [17] Robot-Welding, *Robot Types* (2001). [www-document]. Available: <http://www.robot-welding.com/robots.htm> (Searched 9.11.2009)
- [18] National Instruments, *Motor Fundamentals* (2010). [www-document]. Available: <http://zone.ni.com/devzone/cda/tut/p/id/3656> (Searched 13.01.2010)
- [19] Omega Engineering Inc, *Stepper Motors*. [www-document]. Available: http://www.omega.com/prodinfo/stepper_motors.html (Searched 13.01.2010)
- [20] Putatunda, R., *Types of Robots* (2008). [www-document]. Available: <http://www.buzzle.com/articles/types-of-robots.html> (Searched 02.12.2009)
- [21] RobotWorx, *Robot Timeline – Robotic History*. [www-document]. Available: <http://www.used-robots.com/robot-education.php?page=robot+timeline> (Searched 02.12.2009)
- [22] Kuivanen, R., *Robotiikka*. Vantaa: Talentum Oyj, 1999, 188 pages
- [23] Lahden ammattikorkeakoulu, *Robotiikka* (2008). [www-document]. Available: http://tl-automaatio.lpt.fi/automaatio/opetus/luennot/pdf_tiedostot/Robotiikka_yleinen.pdf (Searched 15.11.2009)
- [24] Argon Lab Systems, *Laboratory Robots and Automation – Benefits & Advantages* (2007). [www-document]. Available: <http://www.argonlabsystems.com/content/view/10/18/> (Searched 22.01.2010)
- [25] VDMA Robotics, *Executive Summary of World Robotics 2009 Industrial Robots* (2009). [www-document]. Available: http://www.worldrobotics.org/downloads/2009_executive_summary.pdf (Searched 30.10.2009)
- [26] RobotWorx, *Industrial Robot Basics*. [www-document]. Available: <http://www.robots.com/faq.php?question=robot+basics> (Searched 8.11.2009)
- [27] Olympus Technologies, *Linear Robots*. [www-document]. Available: <http://www.olympustechnologies.co.uk/about/types-linear.php> (Searched 8.11.2009)
- [28] Bonev, I., *Delta Parallel Robot – the Story of Success* (2001). [www-document]. Available: <http://www.parallelic.org/Reviews/Review002.html> (Searched 24.02.2010)

- [29] Wikipedia, *Parallel Manipulator* (2009). [www-document]. Available: http://en.wikipedia.org/wiki/Parallel_manipulator (Searched 24.02.2010)
- [30] Yamaha Motor, *What's the Scara Robot*. [www-document]. Available: <http://www.yamaha-motor.co.jp/global/industrial/robot/ykx/what/index.html> (Searched 9.11.2009)
- [31] Monkman, G. J., Hesse, S., Steinmann, R., Schunk, H., *Robot Grippers* (2007). [www-document]. Available: <http://books.google.fi/books?>
(id=prZ1MiKjdhgC&dq=robot+grippers&printsec=frontcover&source=bl&ots=YYplEMbCdG&sig=gzyOgwOXIeb-W0UIhuTyQf9YxB4&hl=fi&ei=KVdUS9yZJ8nb-Qa22tzMCA&sa=X&oi=book_result&ct=result&resnum=10&ved=0CC0Q6AEwCQ#v=onepage&q=&f=false) (Searched 16.11.2009)
- [32] Fargo Controls Inc, *Proximity Sensors: Inductive, Capacitive, Photoelectric & Magnetic* (2005). [www-document]. Available: <http://sensorsproximity.com/> (Searched 10.12.2009)
- [33] Penton Media Inc & Machine Design Magazine, *Proximity Sensor Information* (2010). [www-document]. Available: http://www.sensors-transducers.machinedesign.com/guidedits/content/bdeee4/bdeee4_7.aspx (Searched 10.12.2009)
- [34] Ohanian, H. C., *Physics*. Canada: Penguin Books Canada Ltd, 1985, 1012 pages
- [35] SICK Oy, *Induktiiviset lähestymisanturit*. [www-document]. Available: <http://www.sick.fi/fi/products/tuoteryhmat/teollisuusanturit/induktiivisetlahestymisanturit/fi.html> (Searched 16.02.2010)
- [36] Fargo Controls Inc, *Capacitive Operating Principles* (2005). [www-document]. Available: http://sensorsproximity.com/op/capacitive_op.html (Searched 22.02.2010)
- [37] Fargo Controls Inc, *Photoelectric Operating Principles* (2005). [www-document]. Available: http://sensorsproximity.com/op/photo_op.html (Searched 22.02.2010)
- [38] Dwayne, P., *Force/Torque Sensor Systems: Optimizing Robot Performance* (2010). [www-document]. Available: http://findarticles.com/p/articles/mi_qa3957/is_200308/ai_n9247065/ (Searched 20.02.2010)
- [39] University of Ottawa, *Sensor-Based Robot Control*. [www-document]. Available: <http://www.site.uottawa.ca/~petriu/CEG4392-IntroRobotics-Sensors.pdf> (Searched 30.01.2010)
- [40] Wikipedia, *Machine Vision*. [www-document]. Available: http://en.wikipedia.org/wiki/Machine_vision (Searched 07.12.2009)
- [41] Oulun yliopisto - sähkötekniikan osasto, *Konenäkö*. [www-document]. Available: <http://www.ee.oulu.fi/mvg/about/konenako.pdf> (Searched 07.12.2009)
- [42] Wikia, *Machine Vision*. [www-document]. Available: http://computervision.wikia.com/wiki/Machine_vision (07.12.2009)
- [43] Wikipedia, *CMOS*. [www-document]. Available: <http://en.wikipedia.org/wiki/Cmos> (Searched 02.02.2010)

- [44] Wikipedia, *Charge-coupled device*. [www-document]. Available: http://en.wikipedia.org/wiki/Charge-coupled_device (Searched 02.02.2010)
- [45] DALSA, *CCD vs. CMOS*. [www-document]. Available: http://www.dalsa.com/corp/markets/CCD_vs_CMOS.aspx (Searched 02.02.2010)
- [46] National Instruments, *NI Smart Cameras*. [www-document]. Available: <http://sine.ni.com/nips/cds/view/p/lang/en/nid/204077>. (Searched 02.02.2010)
- [47] Toshiba Teli, *Color Cameras*. [www-document]. Available: http://www.toshiba-teli.com/index.cfm?com=tplGen_products&cid=2&mid=1 (Searched 2.2.2010)
- [48] Prosilica, *Choosing a Machine Vision Camera*. [www-document]. Available: http://www.prosilica.com/support/choose_camera.html (Searched 02.02.2010)
- [49] *OpenCV 2.0 C++ Reference* [www-document]. Available: <http://opencv.willowgarage.com/documentation/cpp/index.html> (Searched 02.02.2010)
- [50] Lewis, F. L., Abdallah, C. T., Dawson, D. M., *Control of Robot Manipulators*. [www-document]. Available: http://robotics.cucei.udg.mx/Index_files/notes/Extra3HT.pdf (Searched 15.01.2010)
- [51] RobotWorx, *Mitsubishi RV-2AJ Five-Axis Robots* (2006). [www-document]. Available: <http://www.used-robots.com/articles.php?tag=1695> (Searched 28.10.2009)
- [52] Mitsubishi, *Industrial Robots Specifications Manual, RV-1A/RV-2AJ Series*. [Instruction manual]
- [53] Mitsubishi, *Standard Specifications Manual, CRI-571 Controller*. [Instruction manual]
- [54] DVT Sensors, *DVT 542C*. [www-document]. Available: <http://www.dvtsensors.com/products/LegendManager.php?action=Spec&Type=542C> (Searched 13.10.2009)
- [55] McNaughton-McKay Electric Company. *Cognex Vision Sensors, Low-Cost, High-Powered*. [www-document]. Available: http://www.mc-mc.com/portals/1/Product%20Content/articles/0407_Cognex.htm (Searched 13.10.2009)
- [56] Mitsubishi, *HMI Visualisation Tools*. [www-document]. Available: <http://download.mitsubishi-automation.com/resources/technical/207075.pdf> (Searched 15.10.2009)
- [57] Vesic, D. D., *How to make Windows Form app truly Full Screen (and to hide Taskbar) in C#?* (2006). [www-document]. Available: <http://www.vesic.org/english/blog/winforms/full-screen-maximize/> (Searched 25.11.2009)
- [58] Microsoft, *How to cover the Task bar with a window*. [www-document]. Available: <http://support.microsoft.com/kb/q179363/> (Searched 25.11.2009)
- [59] MSDN, *IntPtr Structure*. [www-document]. Available: [http://msdn.microsoft.com/en-us/library/system.intptr\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/system.intptr(VS.71).aspx) (Searched 25.11.2009)

7 Appendixes

1. TicTacToe Data Transfer Protocol
2. Robot Work Cell Exercises handout
3. Exam

TicTacToeServer data transfer protocol

The Messages the Client Sends

Nick request

Syntax

NICK : Nick : Symbol (String [4] : String [16] : String [1])

Symbol is “X” or “O”

Example

NICKJarnoX

Game move

Syntax

GAME: Socket number : Reset : Move (String [4] : Int [3] : String [1] : Int [1])

Socket number is 001...100

Reset is “R” for Reset or “C” for Continue

Move is 1...9

Example

GAME023C7

Chat message

Syntax

CHAT : Socket number: Message (String [4] : Int [3] : String [256])

Example

CHAT023OMG stfu n00b! lol

Exit request

Syntax

EXIT

The Messages the Server sends

Nick response

Syntax

NICK: Success: Socket number (String [1] : Int[3])

or

NICK: Failure: Reason (String [1] : String [1])

Failure/success:

Failure = "F"

Success = "S"

Reason:

Server full = "S"

Nick used = "N"

IP Address banned = "B"

Socket number:

001...100

Example

NICKFN (Refuse)

NICKS023 (Accept)

Game update

Syntax

GAME : Move : Who won (Int [1] : Int [1])

GAME: Move : Who won : Cell 1 : Cell 2 : Cell 3
(Int [1] : Int [1] : Int [1] : Int [1] : Int [1]))

Move is 1...9 or 0, if someone has won

Who won:

0 = nobody, game continues

1 = player

2 = robot

3 = nobody, game over

Example

GAME80 (move to 8, game continues)

GAME71147 (move to 7, player wins, line through cells 1, 3 and 7)

Chat update

Syntax

CHAT : Nick : Message (String [max 20] : String [256])

Example

CHATDefaultNickOMG stfu n00b! lol

Turn update

Syntax

TURN : True/false : Queue length (String [1] : String [2])

True/false:

“T” = Client’s turn to play

“F” = Not yet client’s turn to play

Queue length = 0...99

Example

TURNF08

Info message

Syntax

INFO : Total connections : Players before own turn : Message (String [4] : String [4] : String [256])

Example

INFO00230002Study in Savonia University of Applied Sciences! It’s jätte fun!

Robot commands

Game Moves

Syntax

GAME : Turn: Move : Game status (String [4] : String [1] : Int [1] : Int [1])

RESET

Turn = "X" or "O"

Move: 1 - 9

Game status:

0 = nobody, game continues

1 = player won

2 = robot won

3 = nobody, game over

Example

GAME033

RESET

Robot sends

BUSY

READY + last move

ERROR



Robot Work Cell Exercises

2010
Auvinen, Jarno
Jalkanen, Miia

Table of contents

General About the Robot Work Cell.....	4
Exercises.....	6
Mitsubishi RV-2AJ.....	6
Programming with the Teaching Box.....	6
Coding with Melfa-BASIC.....	7
Variables.....	7
Conditional Statements and Loops.....	9
Moving the Robot Arm.....	9
Connecting the Robot Controller and PC Using COSIMIR.....	10
Teaching the Positions.....	12
Changing the Tools and Checking the Tool Parameters.....	12
Task 1: Pick and Place a Mosaic Tile.....	15
Using Palletizing Functions.....	17
Task 2: Defining Pallets.....	19
Task 3: Composing a Dot Matrix Character	21
Task 4: Returning the Mosaic Tiles After Use.....	21
Task 5: Writing Complete Words.....	21
Task 6: Receiving the Text From The User via Ethernet.....	22
Task 7: Digital Clock Using Dot Matrix.....	22
Task 8: Seven-segment Digital Clock.....	23
Using Subroutines and Functions.....	24
Task 9: Multiple Items.....	25
Task 10: Changing the Tool During Operation.....	25
Creating Multitasking Programs.....	26
Task 11: The Moving Blocks.....	27
Task 12: Camera Inspection with Multitasking.....	29
Ethernet Interface.....	29
Enabling the Ethernet Connection from Controller.....	29
Creating a Server and Client Software.....	30
Network Functions of a Simple .NET Network Program (C#).....	31
Transferring Data Between the Robot and PC Software.....	34
Task 13: Connection Test.....	35
E-designer 7.....	36
Driver Update.....	36
E-designer Workspace.....	36
E-designer Project and Blocks.....	37
Connecting the Operation Panel and PC.....	38
Conveyor Operation.....	39
Task 14: Conveyor Operation Using Built-in Blocks.....	39
Task 15: The Savonia UAS Logo.....	41
Task 16: Security Levels.....	42
Task 17: Localization.....	44
The Language Register.....	44
Translating Texts.....	46
Testing the Localization.....	47
Intellect 1.5.....	48
Task 18: Going Nuts (with DVT).....	48

Measuring the Hole Diameter.....	49
The Nut Location.....	49
Measuring.....	51
Hole Diameter Measure Script.....	53
Task 19: Bolt Action.....	54
Bolt Location.....	55
Preprocessing.....	55
Measuring the Bolt Length.....	56
Sending the Pick Point to the Robot.....	56
Task 20: Quality Control.....	58
Additional Instructions.....	61
The equipment needed for the exercises.....	61
Mitsubishi RV-2AJ.....	62
Melfa-Basic	62
Changing the tool.....	64
Changing the batteries.....	64
About COSIMIR Industrial.....	65
RCI Explorer.....	65
GX IEC Developer.....	66
Conveyor operation.....	66
E-designer 7.....	70
Conveyor operation.....	70
Graphics.....	71
Intellect 1.5.....	74
About the workspace.....	74
Tool Referencing Diagram.....	74
About recording the sequence.....	76
The calibration of the coordinate system	76

General About the Robot Work Cell

The robot work cell (Figure 1) is equipped with a robot arm, a conveyor belt with a controlling PLC, two machine vision cameras, two operation panels and an additional axis, with operates a small table. There are also two web cameras and a light beacon with red, yellow and green lights.

RV-2AJ is a 5DOF (Degree of freedom) robot arm designed for light assembly work. The maximum reach distance is 410 mm. It is one of the smallest robots manufactured by Mitsubishi. It weighs only 17 kg and its maximum payload is 2 kg and the recommended maximum payload for continuous operation is 1.5 kg. The maximum speed of its movements is 2100 mm/s and the repeatability 0.02 mm.

There is a turning aluminium table in the workcell. It is installed as an additional axis. The HC-KFS13 servo produces a continuous torque of 0.32 Nm and a maximum torque of 0.95 Nm. The maximum speed is 4500 rpm.

The robot controller is Mitsubishi CR1 with a 64-bit RISC+DSP processor. It provides functions for linear interpolation, circular interpolation, 3D-modeled circular interpolation, pallet functions, interrupts and multitasking. It is programmed using the Melfa-BASIC IV or MOVEMASTER COMMAND programming languages. It has 16 inputs and 16 outputs (the amount can be extended up to 240) and up to 88 programs can be saved into controller. The controller has emergency stop and door switch functions. The controller has a RS-232C port for connecting to PC and a RS-244 port for the teaching pendant.

The teaching pendant of the cell is Mitsubishi R28TB (Teaching Box). Among many other uses, the TB is used for teaching the positions to the robot. It is also possible to write whole programs using the TB.

There are two Beijer Electronics' operation panels in the work cell. The smaller, E410 with black and white touch screen, has the resolution of 320 x 240 pixels. It is connected to MELSEC FX1N PLC and together they are used to operate the conveyor

belt. The panels are programmed with E-Designer 7 software. The another, E1070 is a 6,5" color TFT display and it is operated using the buttons positioned along the edges of the screen.

The machine vision system of the work cell includes two cameras by Cognex. DVT 542C color camera is installed to the ceiling of the work cell above the conveyor and DVT 530 grayscale camera is located at the side of the conveyor. DVT 542C has a 640 x 480 pixels CCD image sensor, Hitachi SH 4 processor, 64 MB of RAM and 16 MB of Flash memory. DVT 530 is a grayscale CCD camera with the same VGA resolution as the color camera. It has a Motorola Power PC processor, 32 MB of RAM and 16 MB of Flash memory.

The operation voltage for both cameras is 24 VDC and they have their own LED ring lights to ensure the proper lighting of the target. They are connected to the DVT Isolated Bob which provides digital I/O and power to the cameras. In the vision system the cameras operate as servers by sending the processed data to the robot controller. The vision system is configured with Intellect 1.5 software.

The PLC that controls the conveyor is Mitsubishi FX1N-14 MR-DS. It is a compact and affordable PLC suitable for many applications. Its operating voltage is 12-24 VDC and it has 14 I/O ports. It is programmed with GX IEC Developer 7 software.



Figure 1: Robot work cell

Exercises

This section contains the exercises for the different devices of the robot work cell. The basic exercises are tutorials to guide student through the task, step by step. All of the tasks are graded and example solutions are available for each of them.

Mitsubishi RV-2AJ

CAUTION!
DO NOT MODIFY, REPLACE OR DELETE THE PROGRAMS OR POSITION LISTS IN SLOTS 84 – 88!

Programming with the Teaching Box

It is possible to write a whole new program without PC by using the robot's teaching pendant. Unfortunately, teaching pendant is very slow to use, but its still a great aid in doing minor changes in programs.

Make sure that the controller is in *TEACH* position. Turn the TB's mode switch to *ENABLED*. From the main menu, select *TEACH*. There you can select a new program slot or edit existing programs by typing the slot number. *PR* shows the program slot number, *ST* shows the command order and *LN* tells the current line number. Move to code line by pressing *RPL* button twice. Now you can write by holding *POS/CHAR* key and typing characters. Remember to add a line number to the beginning of each line or the program will crash in syntax error.

Coding with Melfa-BASIC

Melfa-BASIC is a programming language used for the programming of Mitsubishi robots. It includes all of the usual programming structures such as flow control and repetition statements, different variable types, subroutines etc. The language also contains a wide range of special functions for robot applications. In these exercises, only the very basic commands are introduced. Huge amounts of interesting coding stuff can be found in the “Detailed explanations of functions and operations” Instruction Manual.

In Melfa-BASIC, each line in a program must start with the line number. In Mitsubishi’s own documentation and examples the numbering starts from 10, growing by 10 in every line. There is no obvious reason for this manner, because the programs also work, if the numbers increase, for example, by 1. However, the modifying of the code is much easier, if you can add a line or two here and there, so keep to that custom. Code can also be written without the line numbers at all, if the numbers are added in COSIMIR with *Renumber (CTRL + R)* command.

Variables

Melfa-BASIC has two kind of variables, internal and external. Internal variables are used inside the programs and each time a program starts, the variables are created and set to the initial value. Internal variables cannot be affected by the other programs they are created in. Internal variable types are sorted in Table 1. Each variable's first letter identifies the variable type, as seen in the example. Also, the last letters are required in some cases.

Table 1: Internal variables and examples

Variable	Letter	Example
Integer	M	M1=31008
Single precision real number	M (!)	M1!=1.2E+5
Double precision real number	M (#)	M1#=3.14
Character string	C (\$)	CSMPL\$="SAMPLE"
Position	P	P1=(12,3,3,0,0,0)(0,0)

External variables can be used between programs and the values remain in the controller's memory even if a program is stopped and power is turned off. These variables are especially useful in the multitasking programs. External variables are preconfigured in the system memory. Table 2 shows the possible variables to be used.

Table 2: External variables

Type	Range
Double precision real number	M_00 - M_18
Position data	P_00 - P_19
Character string	C_00 - C_19

Internal variables can be defined before use, but this is not required. Still, defining the variables is a good programming manner and it also allows the naming of variables without the identifying letters. Defining is introduced in the Table 3.

Table 3: Defining the variables

Variable	Example
Integer (INTE)	DEF INTE INT1
Single precision real number (FLOAT)	DEF FLOAT LILNUMBER (DOUBLE)
Double precision real number	DEF DOUBLE BIGNUMBER
Character string (CHAR)	DEF CHAR SAMPLESTRING
Position (POS)	DEF POS INITPOS

Melfa BASIC also supports arrays up to three dimensions. The arrays are explained in the Table 4, below.

Table 4: Arrays

Example	Explanation
DIM MNUMB!(2,2,2)	Three dimensional single precision real number array with $2 \times 2 \times 2$ elements
DIM PLIST(4,2)	Two dimensional position array with 4×2 elements
DIM CSTR(8)	String array with 8 elements

Conditional Statements and Loops

If-then-else statement is familiar from almost any programming language, and it doesn't differ in Melfa-BASIC. The comparison operations in Melfa-BASIC are equal (=), not equal (<>), smaller than (<), larger than (>), smaller or equal (<=) and larger or equal (>=). The logical operations *OR*, *AND*, *NOT* and *XOR* can be used between the comparisons. *GOTO* jump point can refer to a label or a linenumber (referring to the linenumber is not recommended). Label is marked with asterisk (*). Subroutine call *GOSUB* jumps program to execute the referred subroutine and then returns to normal execution of the main program. The subroutine start points are also marked with asterisk and the last line of subroutine is *RETURN*. Example is shown below in Table 5.

Table 5: Example of conditional branching

Example	Explanation
10 IF M1<M2 THEN	If M1 is smaller than M2
20 GOSUB *COUNT	Jumps to “COUNT” subprogram
30 BREAK	Escapes from IF statement to line 70
40 ELSE	If equal or larger than M2
50 GOTO *EP	Goto label *EP
60 ENDIF	
70 GOTO 10	Jump to line 10
80 *COUNT	Start of subprogram
90 M1=M1+1	Add one to M1
100 RETURN	End subprogram
110 *EP	Label EP
120 END	End program

Moving the Robot Arm

In Melfa-BASIC, there are several operation command types for moving the robot. The most used is the joint interpolation movement (*MOV*), which moves from current position to the another position with smooth curves. The second movement command used in these exercises is linear interpolation movement (*MVS*), which moves the robot linearly from the current position to another. There is also circular movement interpolation (*MVR*), but it is not used in these exercises. In addition to the actual movement commands, there are a couple of commands affecting the movement. Continuous movement (*CNT*) command can be set to prevent the robot to stop or to slow down the movement between the position points. Speed override command (*OVRD*) can slow down the movements related in the set maximum speed. Delay command (*DLY*) is good to be used after movement to confirm that the robot has already moved to the next position. Table 6 shows short examples of these commands.

Table 6: Moving the robot

Example	Explanation
10 OVRD 20 20 MOV P1	Force the speed to 20% of the set speed and move from the current position to P1 position using the joint interpolation
10 CNT 1 20 MVS P1 30 CNT 0 40 DLY 1 50 MVS P2	Moves continuously through P1, then stops for a second and moves to position P2. Linear interpolation is used.

Connecting the Robot Controller and PC Using COSIMIR

COSIMIR Industrial is a programming and simulation software for the industrial robots. The robot program can be tested in a 3D-environment without the risk of the faulty program causing the robot to damage itself or its surroundings. The basic workspace is a very typical Windows MDI application (Figure 2).

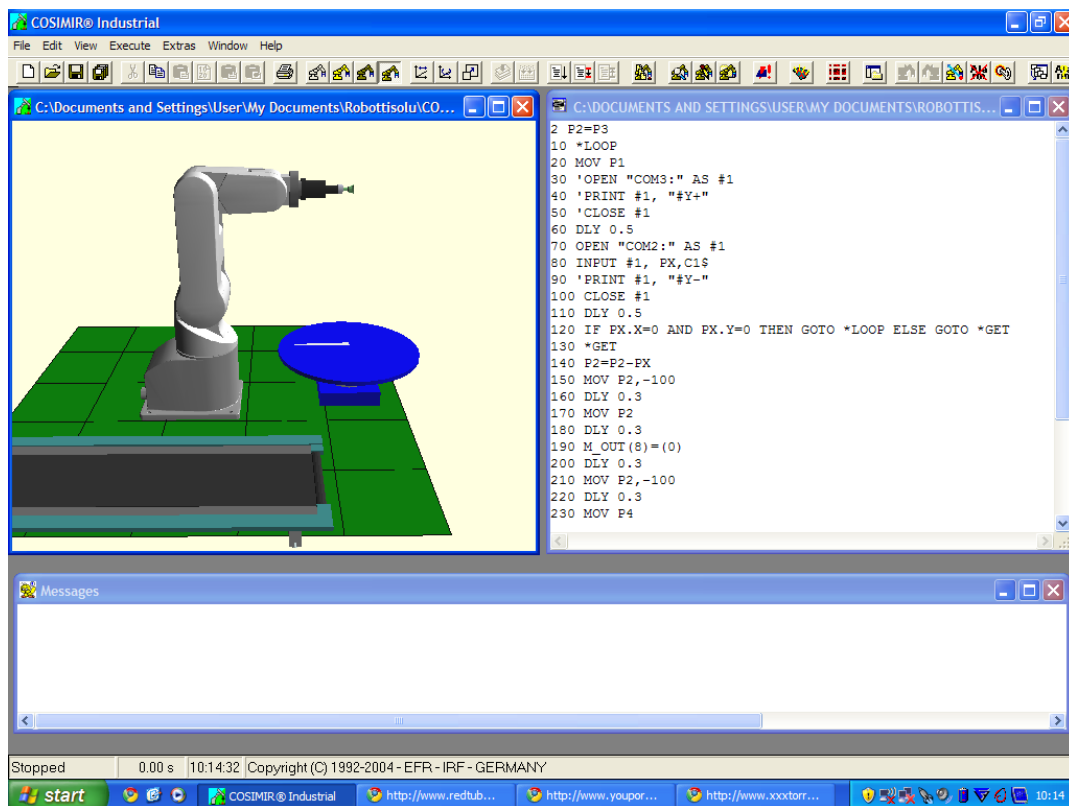


Figure 2: Standard view with Robotti.MOD workcell open

To use COSIMIR, connect the COSIMIR licence connector (Figure 3) to an USB port and open the program. License connector must be connected continuously while using COSIMIR, because the software will crash if the connector is removed and COSIMIR may not recover even if the connector is connected again.



Figure 3: *The COSIMIR Industrial licence connector*

COSIMIR arranges the solutions as work cells. In these examples, you can use the ready work cell that can be found in the My Documents\Robot work cell\COSIMIR folder of the user account. A work cell contains the 3D-model of the robot and the other equipment of the work cell.

Save a copy of the .MOD file for your own use. All the exercises can be done in this same instance of the work cell. Then select *File > New > MRL Position list*. Do the same for a new *MELFA-BASIC IV-Program*. You can write the program without linenumbers, it is possible to automatically add them by pressing *CTRL + R*. When program is ready, it must be saved before sending it to the robot. The controller uses numbers as a naming custom. Save the files like “2.POS” and “2.MB4” respectively.

Note that you may have multiple windows open with different programs and position lists, and COSIMIR saves only the contents of the active window. Also, an active element is downloaded or uploaded from the robot when transfer buttons in the toolbar are clicked. When downloading the programs to robot, switch the pendant to “*disabled*” and stop the running programs from the controller. If the robot is switched

to teach mode and the teaching pendant is enabled, an error occurs. If there occurs an error while debugging your program, an error listing can be found with RCI Explorer tool in the right corner of toolbar. It has proven to be much more pleasant to be browsed than Mitsubishi's Troubleshooting Guide. Also much more data, like positions, speeds, servo voltages and currents as well as the robot's parameters can be collected and edited from RCI Explorer.

After the program has been downloaded to robot, positions must be taught by using the teaching pendant (see chapter "Teaching the positions")

Please see CONTROLLER SETUP, BASIC OPERATION, AND MAINTENANCE INSTRUCTION MANUAL for further information about robot programming.

Teaching the Positions

Every program saved in the controller has its own position list which has the same name (number) as the program and a .POS file name extension. The positions can be taught by moving the robot into the desired position with the Teaching Pendant (TB) and then recording the position data. The position list can be uploaded from the controller to PC using COSIMIR software.

With the pendant's mode switch in *ENABLED* position and the deadman's switch held down, press the *TEACH* key. Select the program you want to use. Press *POS* and *ADD* keys, the cursor moves to the X-line. Move the robot into the position you want to register.

Before moving the robot arm, make sure that speed is set to slow enough to ensure that the robot doesn't collide in the environment. The speed can be set by holding deadman's switch and by pressing *STEP/MOVE* key. When you hear the servos click on, press *+/FORWD* or *-/BACKWD* keys to change the speed.

You can move robot by holding deadman's switch and *STEP/MOVE* key and by pressing the joint keys. Jog mode can be changed like before, by pressing *TOOL*, *JOINT* or *XYZ* key. Joint mode moves only a single joint at a time. *XYZ* moves the robot in a three-dimensional world coordinate system. In the *TOOL* jog mode, the coordinate system is relative to the the tool coordinates.

To save your position, hold down the *STEP* key and press the *ADD* key. The buzzer beeps. Confirm the operation by pressing the *ADD* key again. The buzzer beeps again and the position is now registered. You can add another position by writing the position number or by browsing positions with *+/FORWD* or *-/BACKWD* keys and repeating the previous steps.

When every position is taught, you can debug your program by running it in steps. Press *COND* to view your code. Move to the first line. Hold *STEP/MOVE* and *INP/EXE* key (remember the deadman's switch!). Program is executed line by line and it can be stopped by releasing *INP/EXE*

After registering all the needed positions, press the *MENU* key to save the changes and to return to the main menu.

Changing the Tools and Checking the Tool Parameters

The tools can be changed easily with ready programs in slots 86 and 87. Program 86 is meant to be used as a subroutine, for further information, see page 24: Using Subroutines. When the program is called, integer is passed to it as a parameter. **Value 1 means that the vacuum gripper is changed to the hand gripper and value 2 means that the hand gripper is changed to the vacuum gripper.**

Program 87 is “manual” tool change, for functioning, it needs Server Console application to be started in the laptop PC. When the robot is connected to the laptop, just press number to execute the desired action.

When the robot's tool is changed, also its tooltip coordinate (z-axis) is changed. That is why it is extremely important to verify, that also the length parameter is correct to avoid collisions. Already installed tool change programs change these parameters automatically, but if there is some interrupt and a program has to be stopped and reseted before parameter is changed, the tool height may be incorrect (see figure).

The robot controller has five parameters for the tool coordinates. These are *MEXTL* and *MEXTL1 – MEXTL4*. Up to four tool coordinates can be pre-assigned to the *MEXTL1 – MEXTL4* parameters. **Tool number 1 is assigned for the vacuum gripper and tool number 2 for the hand gripper.** Numbers 3 and 4 are currently unassigned. The tool coordinate presets can be viewed or changed with the teaching pendant by holding the *TOOL* button and pressing numbers 1 – 4 respectively.

In programs, these predefined tool parameters can be used with *M_TOOL* command. This is better and easier way to change the tool coordinates inside a program. For example, *M_TOOL=1* will set the tool coordinates to match the vacuum gripper. *M_TOOL* can also be used in conditional statements if the currently used tool needs to

be checked. *TOOL* command changes the coordinates with typed values and saves them to *MEXTL* parameter. For example, **TOOL (0,0,88.28,0,0,0)** sets the tool length to **88.28 millimeters, which is the length of hand gripper**. **TOOL (0,0,68.78,0,0,0)** sets the tool length to **68.78 millimeters, which is the length of vacuum gripper**. See Figure 4 for example.

When changing tools, remember to check the tool parameter

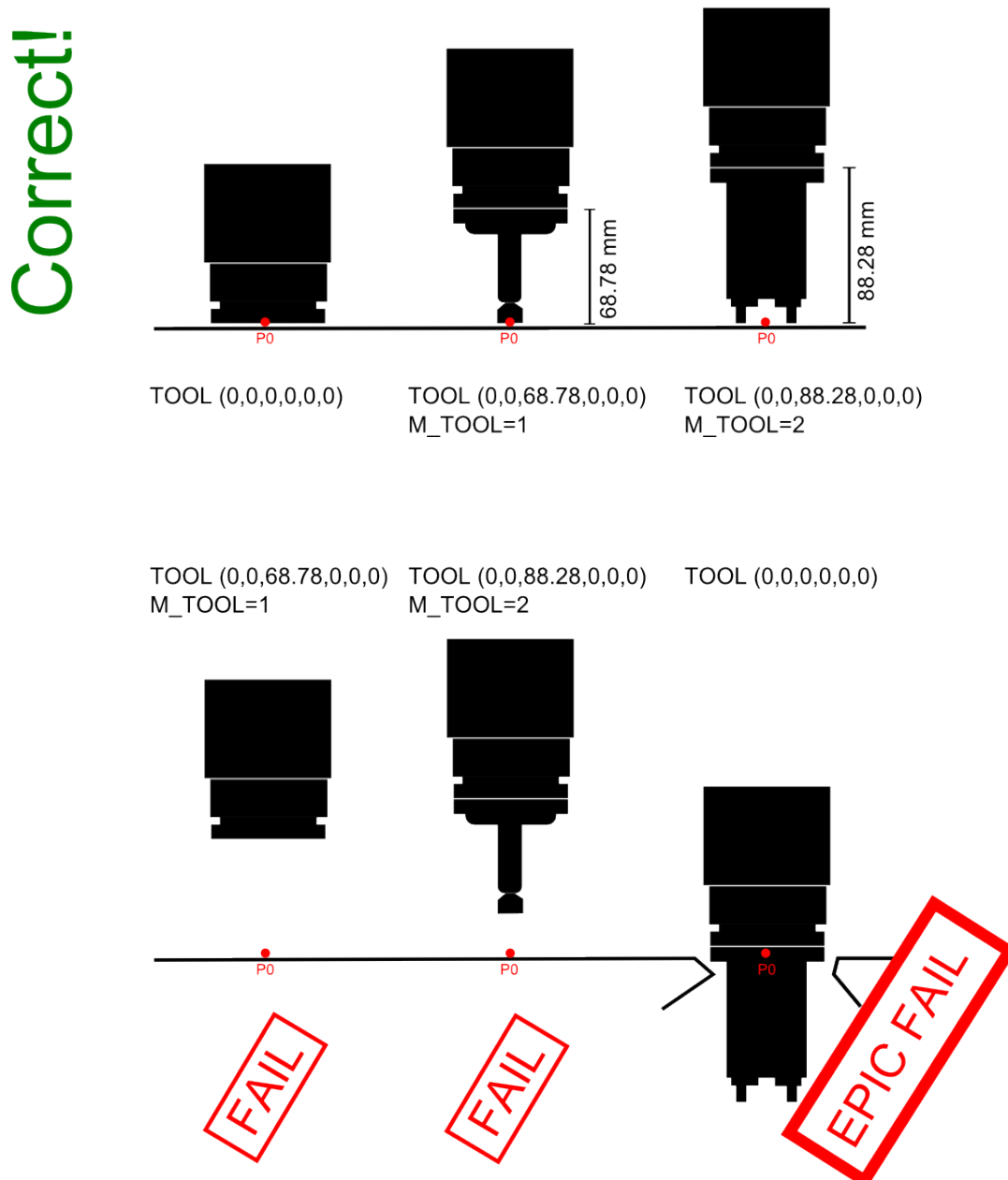


Figure 4: Changing the tool coordinates

Task 1: Pick and Place a Mosaic Tile

Difficulty: Beginner
Tool: Vacuum gripper
Other equipment: Mosaic tile
Example program number: -

Insert a mosaic tile on the conveyor belt. Make the robot to perform the following work cycle: (see Figure 5)

Move to P1, wait 0.5 s
Move 10 mm above the work piece
Move to P2
Pick up the work piece, wait 0.5 s (to get the maximum vacuum)
Move 10 mm above the work piece
Move to P3
Move 10 mm above the place where to set piece
Move to the release position
Release the work piece, wait 0.5 s (suction completely off)
Move 10 mm above the work piece
Move to P1

The Melfa-BASIC code for described operation is as follows:

```
10    MOV P1
20    MOV P2, -10
30    DLY 0.5
40    MOV P2
50    M_OUT(9) = 1
60    MOV P2, -10
70    DLY 0.5
80    MOV P3
90    MOV P4, -10
100   MOV P4
110   M_OUT(9) = 0
120   DLY 0.5
130   MOV P1
140   END
```

Now move the robot with the TB and joint / XYZ jog mode. Define the following positions:

P1	Initial position
P2	Wait position 1
P3	Waypoint
P4	Wait position 2

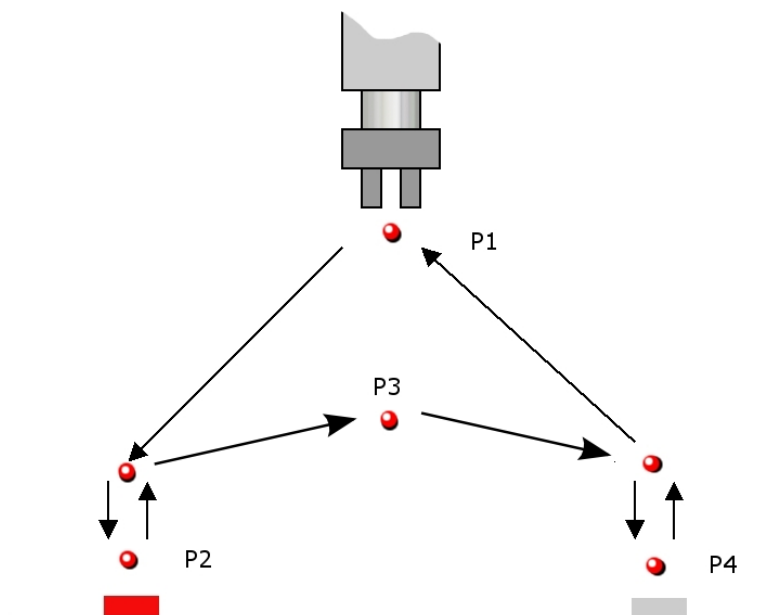


Figure 5: Positions for the first exercise

If you have created a program with COSIMIR, save the program and teach the positions with the teaching pendant.

Finally, when you think everything is ready and functioning, disable the teaching pendant and switch the controller to *AUTO (op)* mode. Press *CHNG DISP* until you see *P* letter in the controller's display. Select your program number with up and down buttons. Press *CHNG DISP* once. Now you can see the current line of the selected program. Press *CHNG DISP* again. With up and down buttons you can set the maximum speed of the robot, set this to 10 at the first time, you can raise this if you think you don't collide with the robot. Now press *CHNG DISP* again for two times to see your current line. Switch servos on (*SVO ON*), press *START*, keep your finger in the *STOP* button and hope for the best...

Using Palletizing Functions

Melfa-BASIC programming language contains ready-made functions for the use with pallets. Pallets are pre-defined arrays of points and they can be either linear, rectangular or circular. A pallet is defined with the command:

```
DEF PLT <Pallet no.> <Start point> <End point A> <End point B> <Diagonal point>
<Quantity A><Quantity B> <Assignment direction>
```

Pallet no.	The number of the pallet. Only constants from 1 to 8.
Start point	Pallet's start point.
End point A	The first ending point of the pallet. Transit point of the arc in arc pallet.
End point B	The second ending point of the pallet. End point of the arc pallet.
Diagonal point	The diagonal point from the start point. Not used in arc pallet.
Quantity A	The no. of workpieces between the start point and the End point A. In arc pallet, the no. of workpieces between start and end points.
Quantity B	The no. of workpieces between the start point and the End point B. Insignificant for an arc pallet, but must be designated. (For example, as 1)
Assignment direction	The direction of the number assignment (see Figure 6, Figure 7) 1 = Zigzag 2 = Same direction 3 = Arc pallet

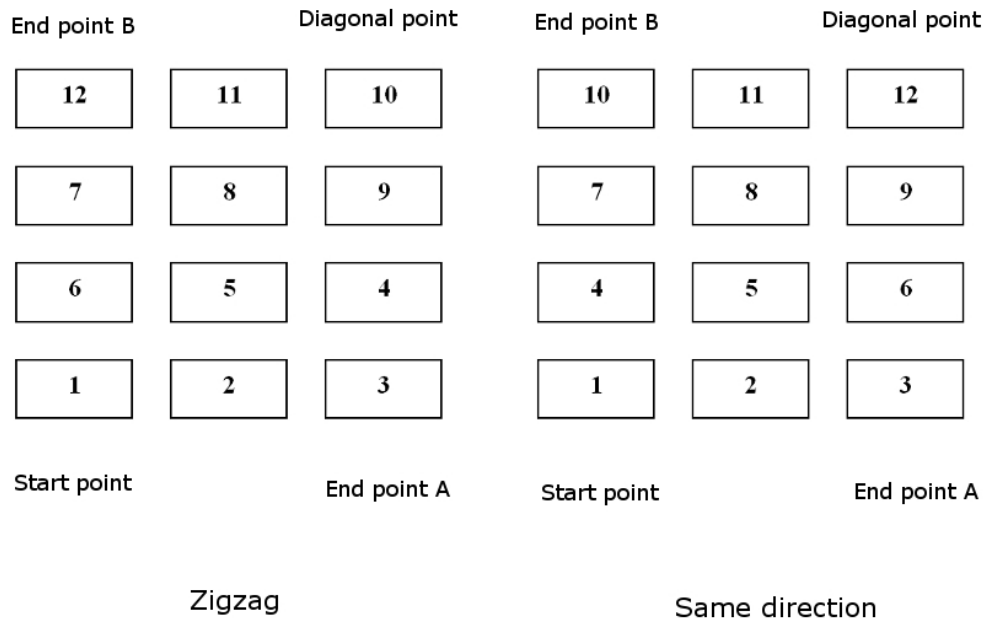


Figure 6: The assignment direction options 1 (Zigzag) and 2 (Same direction)

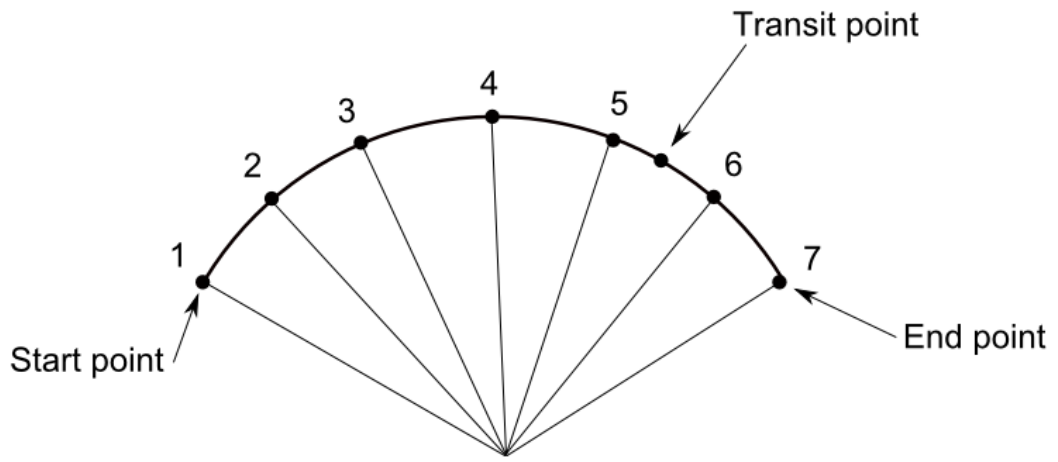


Figure 7: The assignment directions on the arc pallet

NOTES:

Quantity A and Quantity B must be non-zero positive numbers

The Quantity A \times Quantity B must not exceed 32 767, so the maximum size for a square pallet is 181 \times 181 cells

To use linear pallet, set Endpoint B to equal Startpoint and Diagonal point to equal Endpoint A!

Task 2: Defining Pallets

Difficulty: Beginner
Tool: Vacuum gripper
Other equipment: Cylindrical plastic pieces
Example program number: -

Teach the points for two pallets. The first is a 3 x 3 rectangular pallet and the second an arc pallet with 9 cells. Use the pallet grid and 9 plastic pieces. Program the robot to pick each piece from the first pallet and place them into the second and after completing, back to the first pallet in reverse order. Repeat 3 times before returning to the initial position and ending the program. A short example of setting and using pallets is described below in Table 7.

Table 7: Example of palletizing

Example	Function
10 DEF PLT 1, P1,P2,P3,P4,2,2,2	Define 2x2 pallet with the same assignment direction, Set M1 to initial value
20 M1=1	Set M1 to initial value
30 *LOOP	Loop marker
40 P5=(PLT 1,M1)	Set P5 to the selected pallet index M1
50 MOV P5	Move to P5
60 DLY 1	Wait a sec
70 M1=M1+1	Add one to the pallet index
80 IF M1=5 THEN	If the index is greater than pallet, then reset it
90 M1=1	
100 ENDIF	
110 GOTO *LOOP	Loop ever and forever!

Task 3: Composing a Dot Matrix Character

Difficulty: Intermediate
Tool: Vacuum gripper
Other equipment: Mosaic tiles, mosaic platform
Example program number: -

Dot Matrix (see Figure 8) is a 2-dimensional array of dots used to generate characters and symbols commonly used in LED displays and old printers. The typical resolutions are 5×7 pixels or, if there is one line of blank space around each character, 6×8 pixels per character.



Figure 8: Example of Dot Matrix

Make the robot assemble one letter into 6×8 pixels pallet using two different coloured mosaics.

Task 4: Returning the Mosaic Tiles After Use

Difficulty: Intermediate
Tool: Vacuum gripper
Other equipment: Mosaic tiles, mosaic platform
Example program number: -

Make the robot to return the mosaics into the stock pallets after assembling the letter.

Task 5: Writing Complete Words

Difficulty: Intermediate
Tool: Vacuum gripper
Other equipment: Mosaic tiles, mosaic platform
Example program number: -

Program the robot to write the full words by combining the previous programs.

Hint: MID\$ function returns selected part of the string (MID\$("FUNCTION",1,3) would return "FUN")

Task 6: Receiving the Text From The User via Ethernet

Difficulty: Advanced
Tool: Vacuum gripper
Other equipment: Laptop PC
Example program number: -

Use the Ethernet interface (See Chapter "Ethernet Interface") and make the robot to receive the phrase to write as input from the PC's keyboard. Make the robot to send notification back to the PC after assembling the each letter.

Task 7: Digital Clock Using Dot Matrix

Difficulty: Intermediate
Tool: Vacuum gripper
Other equipment: Mosaic tiles and mosaic platform
Example program number: -

Apply the source codes of the previous Dot Matrix exercises and create a program which gets the current system time, and collates the mosaic pieces to form of a digital clock. When the time is updated, the robot removes only the numbers that need to be updated and arranges the removed pieces back to the stock pallet in reverse order. Program is looped. You can use smaller Dot Matrix for numbers, 3×5 is good size to fit all the four numbers to the same line, as can be seen in the Figure 9. A single pallet for each of the numbers may be easier to program than one large for the all.

Hint: C_TIME command returns system time in HHMMSS format.

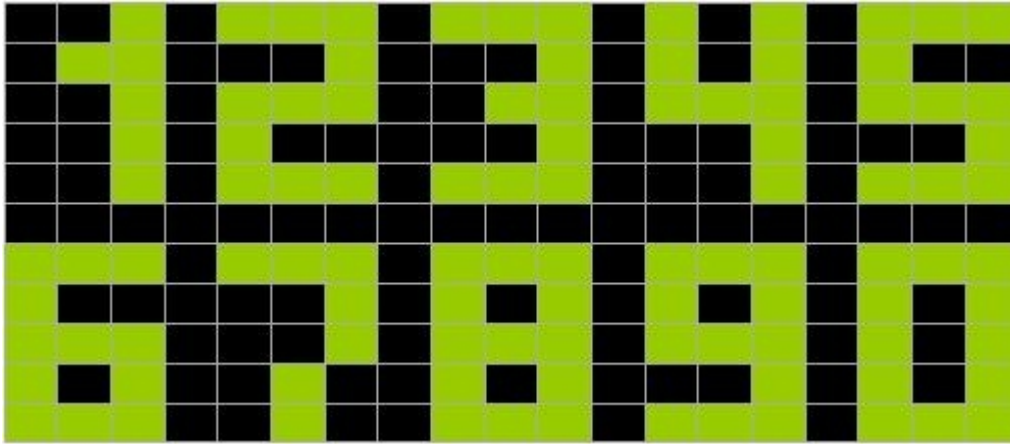


Figure 9: Dot Matrix clock

Task 8: Seven-segment Digital Clock

Difficulty: Intermediate
Tool: Hand
Other equipment: Rectangular plastic pieces
Example program number: -

There are small, green, rectangle shaped plastic blocks which can be used to display time faster than by using the mosaic pieces. Create a program which shows four 7 segment digits and updates the time. To make this option more interesting, update the time in three steps – first, move the segments needed to be removed to the places that are going to be filled, then remove the rest of the useless segments to the storage pallet or, if needed, add more segments to the digits.

Using Subroutines and Functions

As taught in the beginning of this handout, subroutines are very useful to clear the code and to split the program to easily understandable parts. In Melfa-BASIC, there can also be defined own functions to make calculations. A function has to consist *FN* letters to identify that it is a function and *M*, *C* (with \$ at the end) or *P* to identify the type. There is an example in Table 8:

Table 8: Defining functions

10 DEF FNMCAL(MS,ME)=(MS*ME)	Function multiplies integers
20 DEF FNCADD\$(CS1\$,CS2\$)=(CS1\$+CS2\$)	Function adds strings
30 DEF FNPPOS(PO,PC)=(PO-PC)	Function subtracts coordinates

If several programs have the same very complicated and long sequence repeated regularly, it would be much easier to write a separate program, which is called inside the main programs to ease the coding and to save the memory. To do this, *CALLP* command can be used. This command can also pass variables or positions to the called program. Order of the passed variables is defined with *FPRM* command in the second program. The next example in Table 9 shows how the command is used.

Table 9: Calling other programs

10 'THIS IS THE MAIN PROGRAM 20 CALLP "1", M1,M5 30 GOSUB *CHK	Program number 1 is called and integers M1 and M5 are passed to the subroutine
10 'THIS IS THE SUBROUTINE (1.MB4) 20 FPRM M01,M02	Get the variables from the main program (M1=M01 and M5=M02)

Note:

The following tasks utilize machine vision, which is covered in the Chapter “Intellect 1.5” and conveyor logic related Data Exchange, which is explained in Additional Instructions.

Task 9: Square path with multitasking

Difficulty: Intermediate
Tool: Vacuum gripper or hand gripper
Other equipment: -
Example program number: -

In this exercise, the robot moves on a square shaped path using functions. Create two programs. Define starting position in the main program. Pass the position to the another program. The second program compares the starting position and the robot's current position and calculates the trajectory to the next corner of the square. Call the second program four times to complete the square.

Task 10: Changing the Tool During Operation

Difficulty: Intermediate
Tool: Vacuum gripper and Hand
Other equipment: Cylindrical plastic pieces, rectangular plastic pieces
Example program number: -

Use two kinds of objects, the cylinder shaped plastic pieces that are grabbed with the vacuum gripper and the rectangular pieces that are grabbed with the hand tool. Use the “First match” and “Best match” settings in Intellect's identification tool settings. Identify the products one at the time and, if the current tool attached to the robot is not the right one, change it using the tool change subroutine located in program slot 86. The numerical value of 1 or 2 must be passed to the subroutine.

- 1: Change hand gripper to vacuum gripper
- 2: Change vacuum gripper to hand gripper

Creating Multitasking Programs

Multitasking programs will function simultaneously when two or more programs are assigned to the multitasking slots and started. If more than one of the programs are moving the robot, mechanism control must be released for the use of the operating program. When a multitasking program is ended, it must be stopped and the multitasking slot must be cleared. Programs can be loaded into multitasking slots from the robot parameters (see Instruction Manual) or by using Melfa-BASIC commands. Data can be passed between programs by using the external variables. Multitasking related commands are explained below in Table 10.

Table 10: Multitasking commands

Command	Example	Function
XLOAD	10 XLOAD 2,"1"	Command loads program to selected slot. Example loads program 1.MB4 to slot 2
XRUN	10 XRUN 2,"1",1	Starts multitasking operation. As in the XLOAD, program slot is inserted first and it comes the program name. Last number is operation mode (0=continuous, 1=cycle stop) If XLOAD is not executed before, XRUN also loads program to slot.
XSTP	10 XSTP 2	Stops program. Only needs a slot number
XRST	10 XRST 2	Resets a program. Program must be stopped before reset
XCLR	10 XCLR 2	Clears memory slot. Program must be stopped and reset before clear.
M_WAI	10 WAIT M_WAI(2)=1	Returns 1 when program slot is stopped. Example waits until program stops

M_RUN	10 WAIT M_RUN(2)=1	Returns 1 when program slot is started. Example waits until program starts
M_RUN	10 WAIT M_RUN(2)=1	Returns 1 when program slot is started. Example waits until program starts
RELM	10 RELM	Releases mechanism to be used by other programs
GETM	10 GETM 1	Gets mechanism. (1 is used unless there are multiple robots)
SERVO ON/SERVO OFF	10 SERVO ON	Set the servo state

Task 11: The Moving Blocks

Difficulty: Intermediate
Tool: Vacuum gripper
Other equipment: Cylindrical plastic pieces
Example program number: -

Write two separate programs, which both move a cylinder shaped plastic block in a pallet. The first program handles pallet 1, and the second moves pallet 2. The first program moves block from pallet position 1 to position 2 (Figure 10, Step 1). After this, the second program moves the block in the second position (Step 2) then the first program acts again, and so on. When the last pallet index is reached, the blocks are moved back in reversal order and the program is looped!

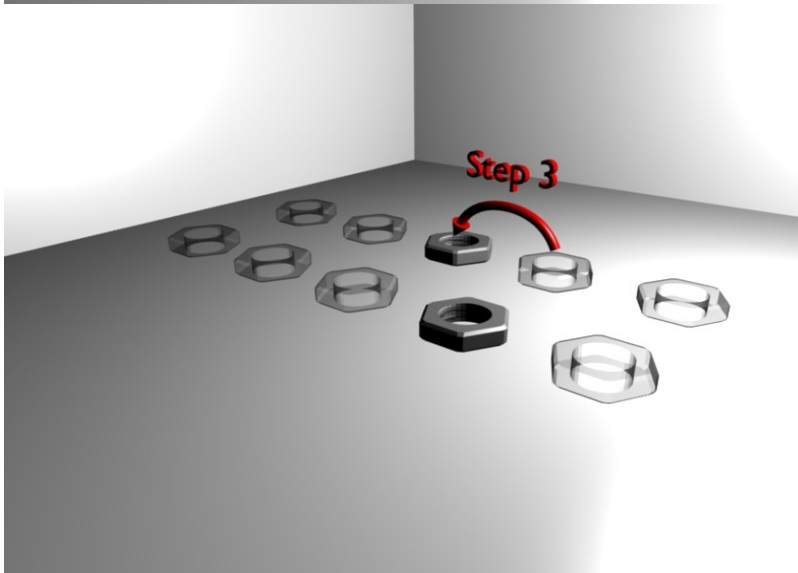
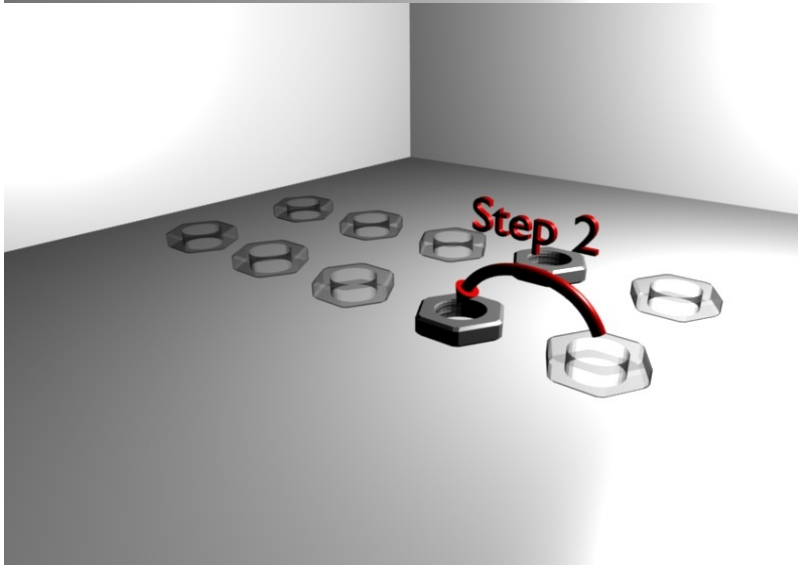
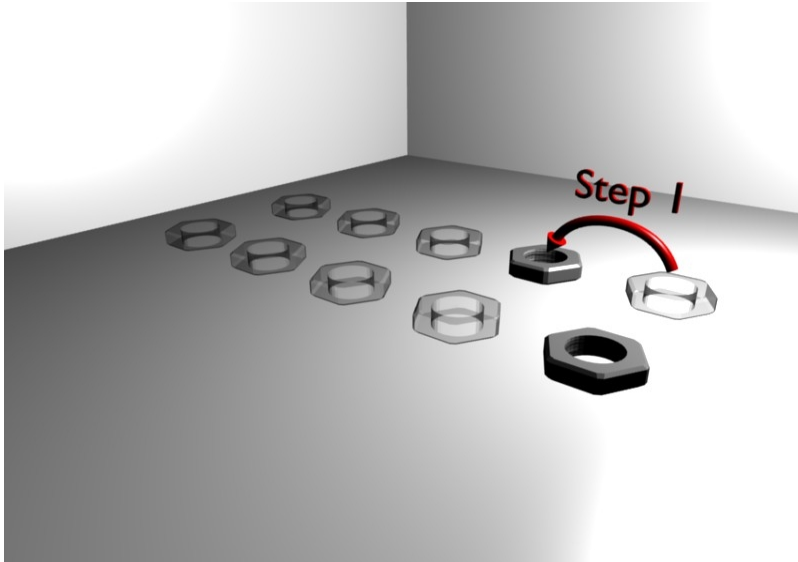


Figure 10: The moving blocks

Task 12: Camera Inspection with Multitasking

<i>Difficulty:</i>	<i>Intermediate</i>
<i>Tool:</i>	<i>Hand</i>
<i>Other equipment:</i>	<i>Mosaic tiles or M8 nuts</i>
<i>Example program number:</i>	<i>-</i>

Run two programs in parallel. The first program moves nuts from the conveyor belt to the pallet. The second program is locating pieces with the machine vision camera and moving the conveyor if there are no objects found (see E410 Data Exchange from Additional Instructions). Locations are forwarded to the first program by external variables. The second program also sends the state of program and the camera inspection results to the laptop.

Ethernet Interface

In the following exercise a PC software is created using C# and Microsoft Visual Studio. For those, who are unfamiliar with them, there are also complete tools that allow the user to concentrate on the programming of the robot rather than learning the C# and Visual Studio from the beginning. The programs are located in the laptop, \Documents\Robot work cell\Accessories folder of the user account.

Server Console is TCP server application which is used as a simple HMI interface for passing information between the robot programs and the computer. For debugging your programs, this may be great aid. Client Console can be used in connecting to DVT cameras (if cameras are set to work as servers), and also this program helps to find out the sources of connection problems.

Enabling the Ethernet Connection from Controller

In order to use Ethernet in passing data to the laptop, COM port must be enabled from the robot controller's parameters (COM #3 is used for connecting the laptop). This can be done with COSIMIR or using the teaching pendant. With teaching pendant, this can be done with following instructions described in Table 11:

Table 11: *Enabling ethernet connection using teaching pendant*

- 1 Press 5 to select MAINT
- 2 Press 1 to select PARAM
- 3 In the first field (Set param. name), type NETPORT
- 4 The parameters are in order of COM numbers, so line should look like {10000, 10001, 10002, ...} where third port 10002 refers to COM #3
- 5 Type NETMSK and make sure mask is 255.255.255.0
- 6 Type NETMODE. This is a setting to decide whether robot works as a server or a client for different communication ports. Number 1 means server and 0 means client. Line should be like this (1, 1, 0, 1, 1,...) , so robot is acting as a client for COM #3.
- 7 Type NETIP. This is robot's own IP. By default setting, this should be 192.168.0.1. It should not be changed.
- 8 Type NETHSTIP. This is parameter where server IPs are saved, in case that robot is a client. The third IP for COM #3 should be set to 192.168.0.2 as a default.
- 9 NETGW is gateway, default should be 192.168.0.254

To set parameters using COSIMIR, open RCI Explorer (Figure 11) from the right side of the toolbar and select Parameter node from tree view. This is faster and easier way to configure the network settings, just set parameters in the same way like before and restart the controller.

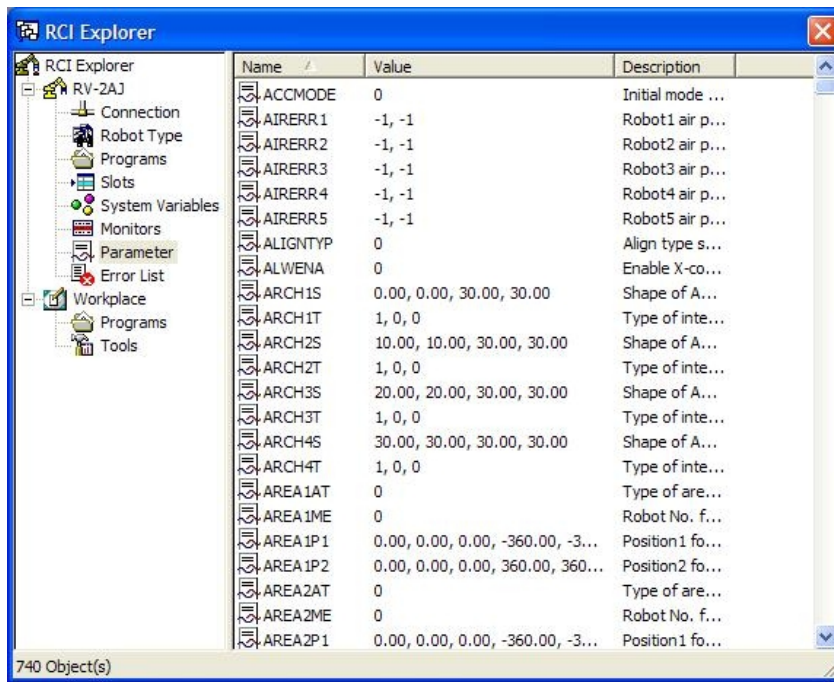


Figure 11: RCI Explorer

Creating a Server and Client Software

There are several terms to be understood before starting the communication between the robot and the external devices like machine vision cameras, PLC or computer.

- Protocol:	Used set of standard communication rules to presentate the transmitted data
- TCP/IP:	T ransmission C ontrol P rotocol and I nternet P rotocol, most commonly used network protocol.
- Server:	Host machine that connects all clients, receives data and also forwards it between clients.
- Client:	Client only connects to server and sends data to server or other clients (through server).
- IP Address:	Every device in TCP/IP network has own address that is used as identification
- Port number:	Port is a virtual or logical data connection used to exchange data between computers
- Endpoint:	Endpoint is the name for the one end of transport layer. It consists of computers IP address and port number.
- Socket:	Socket is a connection between two endpoints.

Network Functions of a Simple .NET Network Program (C#)

Server program can use synchronous or asynchronous communication. In the synchronous communication, server can only handle one client at a time, for an example, it cannot accept more clients and receive data from clients at the same time. That is why synchronous server/client programs are not preferred to be used with multiple clients. Asynchronous program can handle multiple clients, receive and accept new clients at the same time. This is done by using threads. In this case, as the robot is the only client, simple communication would not need asynchronous program, but these example listings are written in asynchronous method, because it is far more useful in other network applications as well.

These listings include all the basic functions needed to communicate with a TCP/IP client. To include network functions in a program, *System.Net*, *System.Net.Sockets* and *System.IO* namespaces must be added (Listing 1).

Listing 1: Added namespaces in the using directive section

```
using System.Net;
using System.Net.Sockets;
using System.IO;
```

Integer *port* is the opened communication port number. Byte arrays *rec* and *snd* are buffers used to save messages while sending or receiving. Socket *connect_socket* is the gateway between server and client (Listing 2).

Listing 2: Creating buffers and socket

```
private int port = 10002;
private byte[] rec = new byte[128];
private byte[] snd = new byte[128];
    private Socket connect_socket = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
```

Void function *StartServer()* initiates the connection by making local endpoint *own_iep*. *Bind* function associates socket with local endpoint and with *Listen* server starts listening for clients. *BeginAccept* is the first asynchronous network function, with it server starts to accept client. *AsyncCallback(Connecting)* refers to connecting function. (Listing 3)

Listing 3: Starting server

```
private void StartServer()
{
    IPEndPoint own_iep = new IPEndPoint(IPAddress.Any, port);
    connect_socket.Bind(own_iep);
    connect_socket.Listen(1);
    connect_socket.BeginAccept(new AsyncCallback(Connected),
connect_socket);
}
```

In first line of *Connecting()*, socket *connect_socket* is defined to handle the existing connection when the accept procedure is being ended. Instead, there can be another socket that receiving can be delegated for. In last line *connect_socket* begins to receive from client using the *rec* byte buffer. (Listing 4)

Listing 4: Callback for connecting the client

```
private void Connected(IAsyncResult iar)
{
    connect_socket = connect_socket.EndAccept(iar);
    connect_socket.BeginReceive(rec, 0, 1024, SocketFlags.None, new
AsyncCallback(Received), connect_socket);
}
```

Next, there is the server receiving function. In first line, *connect_socket* continues to handle receiving. Next, *receiveLength* integer gets the length of the incoming message and message is converted from byte buffer to string. In the last line, *connect_socket* begins to receive again. (Listing 5)

Listing 5: Callback for receiving from client

```
private void Received(IAsyncResult iar)
{
    connect_socket = (Socket)iar.AsyncState;
    int receiveLength = connect_socket.EndReceive(iar);

    string receivedMessage = Encoding.ASCII.GetString(rec, 0,
receiveLength);

    connect_socket.BeginReceive(rec, 0, rec.Length,
SocketFlags.None, new AsyncCallback(Received), connect_socket);
}
}
```

In Listing 6, server will send message. Message string is converted to byte array in the first line, and then it is sent referring to the last asynchronous function *Sending()*.

Listing 6: Sending to client

```
private void SendMessage(string message)
{
    snd = Encoding.ASCII.GetBytes(message);
    connect_socket.BeginSend(snd, 0, snd.Length, SocketFlags.None,
new AsyncCallback(Sent), connect_socket);
}
}
```

In the last listing, Listing 7, *connect_socket* starts receiving after message from the server has been sent:

Listing 7: Callback for sending

```
private void Sent(IAsyncResult iar)
{
    connect_socket = (Socket)iar.AsyncState;
    connect_socket.BeginReceive(rec, 0, rec.Length,
SocketFlags.None, new AsyncCallback(Received), connect_socket);
}
}
```

Transferring Data Between the Robot and PC Software

Melfa-BASIC program has a few connection related commands. Below is a list of the commands with simple examples.

C_COM

C_COM defines the way to communicate within a single program. In these exercises, only Ethernet connection is used. In the example, COM#3 communication channel is configured to IP 192.168.0.2 and port 10002.

```
10 C_COM(3)="ETH:192.168.0.2,10002"
```

OPEN

OPEN opens new connection to the communication channel. In the example, #1 is the number of connection. Remember the semicolon!

```
10 OPEN ("COM3:;") AS #1
```

M_OPEN

Function returns 1 if communication line is open to the selected COM. Can be used to halt program until connection is ready.

```
10 *LOOP
20 DLY 1.0
30 IF M_OPEN(1)<>1 THEN GOTO *LOOP
```

DEF ACT

Program interrupts can be defined with ACT. The example shows, how to halt program if connection to the server is lost.

```
10 DEF ACT 1,M_OPEN=0 GOSUB *PHLT
20 ACT 1=1 'ENABLE INTERRUPT
...
200 *PHLT
210 IF M_OPEN(1)<>1 THEN GOTO *PHLT
220 RETURN
```

INPUT

Receiving of data is done with INPUT function. There must be connection number after INPUT. Variables are separated with comma. In the example, program accepts position data from the sender.

```
10 INPUT #1, P1
```

PRINT

Data can be sent with PRINT function. Variables must be converted to string with STR\$ function. Example sends string with converted integer.

```
10 PRINT #1, "THE NUMBER IS "+STR$(M1)
```

Server Console application can be used to connect to the robot. It is located in the Documents\Robot work cell\Accessories\Server Console\ folder of the user account and there is a shortcut to the program in the Windows Start Menu. By the default, there should be the correct IP and port already configured and server starts just by typing "CONNECT" to the input field. Server Console should be started before robot's program. When connection is created, messages can be sent by typing in the input field.

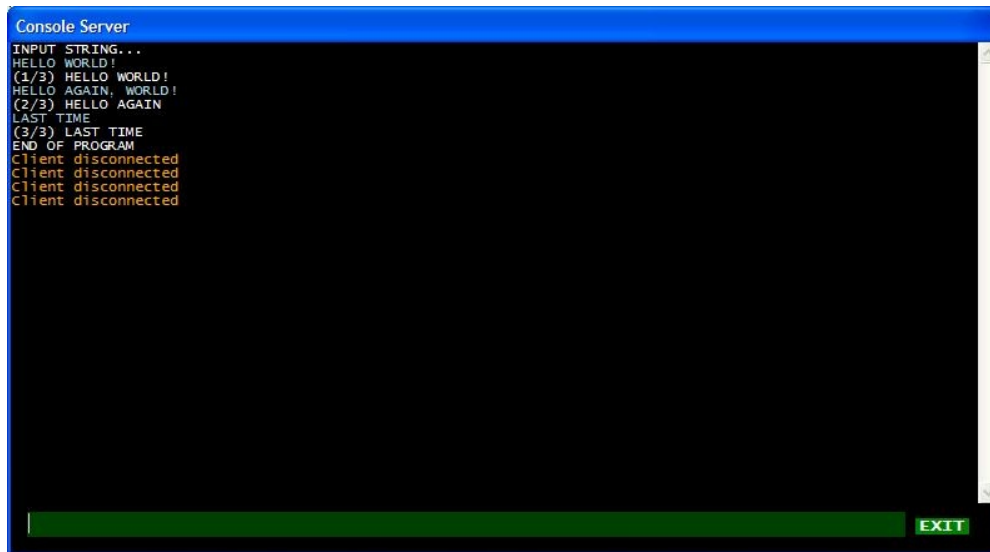
Task 13: Connection Test

<i>Difficulty:</i>	<i>Beginner</i>
<i>Tool:</i>	-
<i>Other equipment:</i>	<i>Laptop PC</i>
<i>Example program number:</i>	-

Create a program that

1. Creates a connection to the laptop
2. Has timeout property, if server is not found
3. Uses ACT to monitor state of connection. Program halts if connection is lost.
4. Gets messages from PC for three times and sends back the number of messages with latest message as a reply for sender.

Program should work like can be seen in Figure 12.



```
Console Server
INPUT STRING...
HELLO WORLD!
(1/3) HELLO WORLD!
HELLO AGAIN, WORLD!
(2/3) HELLO AGAIN
LAST TIME
(3/3) LAST TIME
END OF PROGRAM
Client disconnected
Client disconnected
Client disconnected
Client disconnected
EXIT
```

Figure 12: Connection test

Hint: `M_TIMER` is useful function which can be used in this task. `M_TIMER(1)=0` initializes the timer 1 and starts counting in milliseconds.

E-designer 7

E-designer 7 is a Human Interface Design software by Mitsubishi Electric Automation, Inc. It is used for creating programs for the operation panels. In this section, only the very basic use of the software is described. The manufacturer's excellent documentation with comprehensive examples can be found from path `C:\Program Files\E-Designer\`

Driver Update

After installing the software it is necessary to update the drivers for the operation panels. The latest drivers can be found from the Internet using the E-designer's own update tool which can be found in the *File* menu (*Update Drivers From > Internet*).

E-designer Workspace

Open the E-designer 7 software and choose *Next* from the *File* menu. From the *Project Properties* window which opens select the *Operator Terminal* as *E410 Landscape 6.2x* and the *Controller 1* as *FX CPU Protocol/FX Series 3.1.3.* and *Controller 2* as *Narc/P3 Ethernet.* A blank project opens (Figure 13)

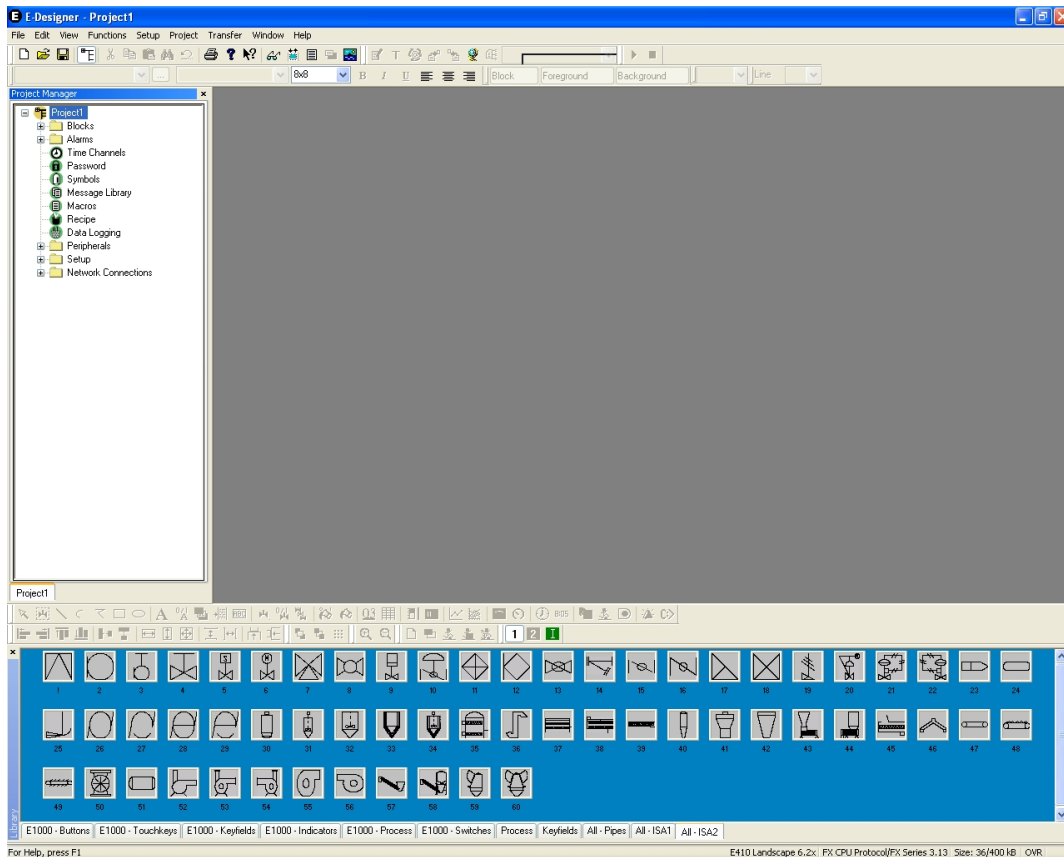


Figure 13: The E-designer workspace

E-designer Project and Blocks

In E-designer, the created software is arranged as a project. The *Project Manager* displays the contained blocks. One block corresponds one screen on the panel's software and they are linked to each other.

Open the *Block Manager* by double clicking the *Blocks* folder in *Project Manager*. New blocks are added by clicking on the desired parent block and then dragging the

arrow to the right from the block. *Create New Block* window opens. Block name and type can be selected here.

All user created blocks must also be linked back to the upper level so that the user is able to return to the main screen. This can be done by drawing an arrow back from the new block to the upper level block. Similar links can be created between blocks on the same level (Figure 14). The buttons are automatically added to each screen.

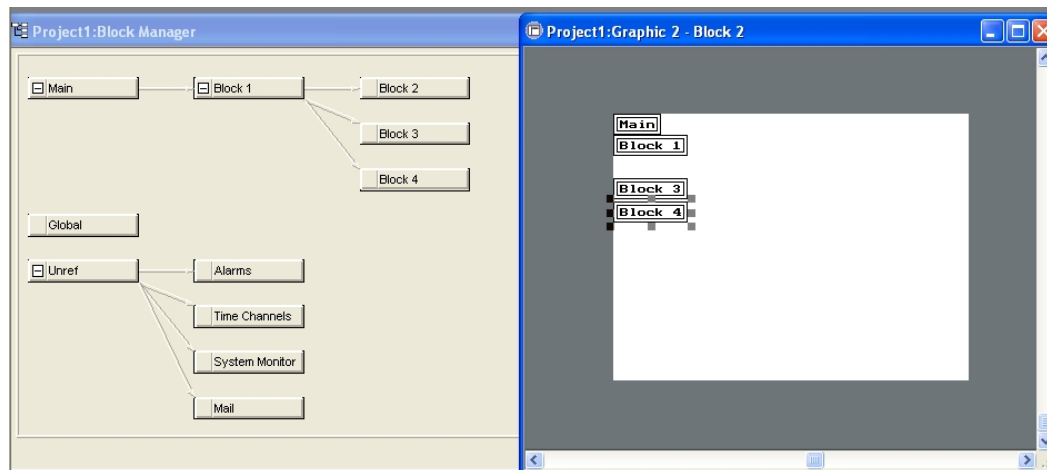


Figure 14: Links between blocks

There is no need to link the software's readymade blocks (System Monitor, Alarms etc.) back to the upper level. They have already ESC button that takes the user back to the previous screen.

Toolbar consists various elements to be added to your blocks. Digital symbol, digital text and ASCII are the most useful. Before you add components, make sure you have the right controller selected as a default by pressing the "1" or "2" buttons in the toolbar. With Controller 1 you can control conveyor's logic and Controller 2 allows access to some of the robot's features. To see a full list of possible signals to be assigned for elements, open *Help* menu and select *Controller Help*. You can load a sample program from E-Designer folder, there are programs for both terminals, E1070 and E410.

Connecting the Operation Panel and PC

The IP address of E410 operation panel is set constantly to be 192.168.0.8. To not to accidentally change the IP while transferring your own program, set TCP/IP connection 1 to match the correct IP and drag it under Ethernet controller (see Figure 15).

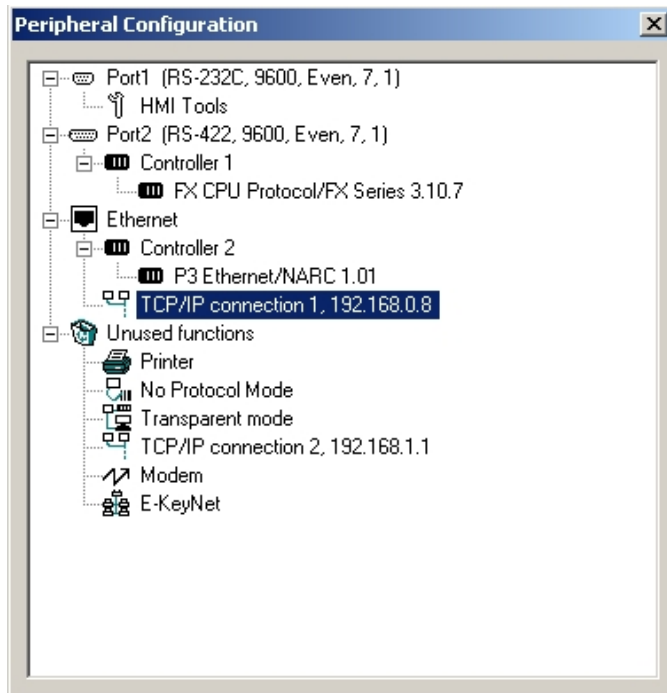


Figure 15: Setting the peripherals in E-Designer

Conveyor Operation

Task 14: Conveyor Operation Using Built-in Blocks

Difficulty: Beginner
Tool: -
Other equipment: -
Example program number: -

Create a new project as described above and name it “Conveyor operation”. Add a child block to the “Main” and name it “Conveyor operation” too. Remember to link the block back to the “Main” block. Open the “Main” block by double clicking it on the *Project Manager*. On the “Main” block there is now a link to the “Conveyor operation” block.

Open the “Conveyor operation” block. Add a digital symbol by clicking the *Digital symbol* icon on the *Objects* toolbar and then click on the “Conveyor operation” screen. A *Digital Symbol* window will open (Figure 16).

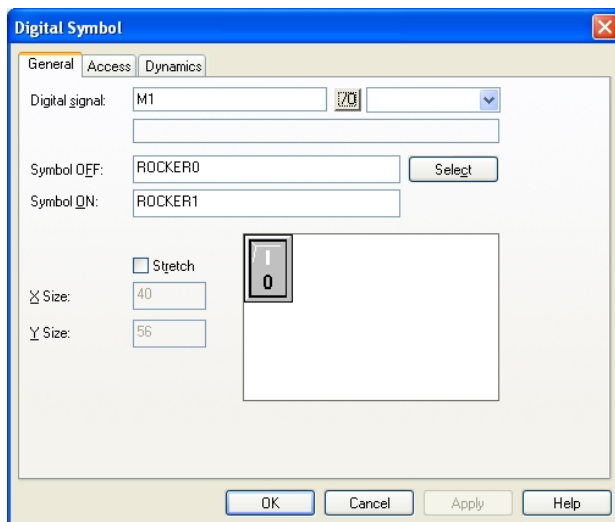


Figure 16: Digital symbol window

Write “M1” to the *Digital signal* textbox. Click the I/O button and write the “M1” to the *Address* field too. Make sure that Controller 1 is selected. Otherwise, select the

Controller systems option 1: FX CPU Protocol/FX1N 3.13 (Figure 17). Click OK to return to the *Digital Symbol* window.

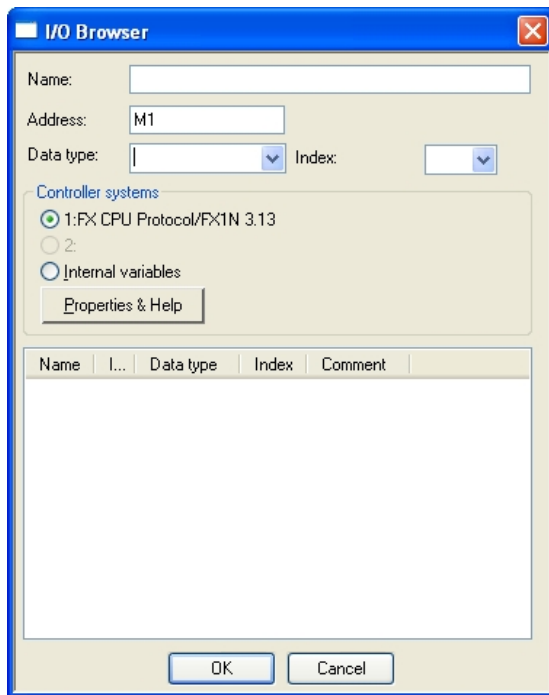


Figure 17: *I/O Browser*

Click the *Select* button next to the *Symbol OFF* field. Select a symbol named “ROCKER0” from the list and click OK. Click on the *Symbol ON* textbox and then the *Select* button. Now select the “ROCKER1” from the list.

On the *Access* tab check the *Enable operator input* checkbox and choose the *Security level 0*. Click OK to close the window.

Test the validity of the project by selecting *Test* from the *Project* menu. If the project is valid, save it and transfer it by choosing the *Project* from *Transfer* menu (Figure 18). Open the *Communication Properties* window by clicking the *Settings* button and select *TCP/IP transfer*. Click OK to close the window and click the *Send* button to transfer the project to the operation panel.

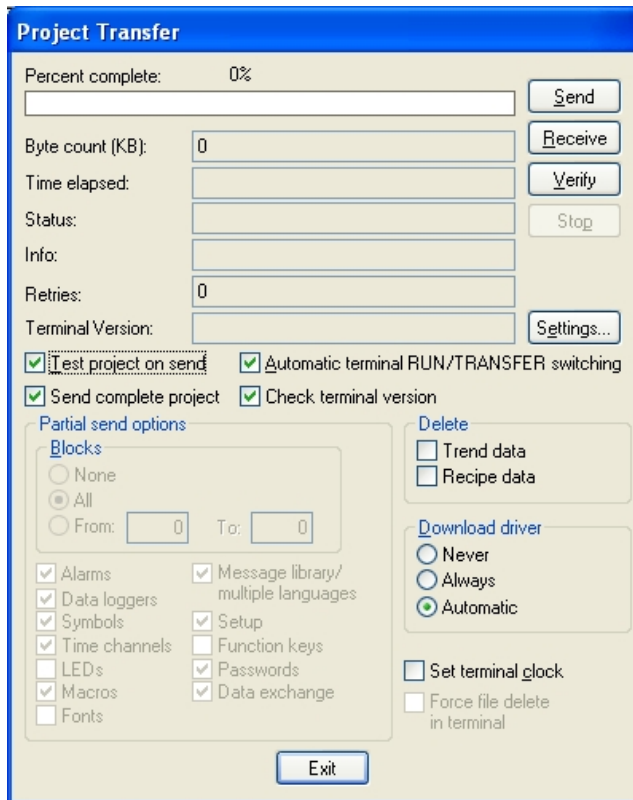


Figure 18: Project transfer

Task 15: The Savonia UAS Logo

Difficulty: Beginner

Tool: -

Other equipment: -

Example program number: -

Add a new child block for the “Main” screen and name it “Savonia”. Onto the new block add Savonia UAS logo by adding a *Symbol* onto it as described above. On the *Select Symbol* window, select the *Show symbols* option *User-created*. Find the picture from the Documents\Robot work cell\E-Designer\Graphics\Savonia.jpg folder and click the *Import* button.

Add a *Touch Key* on the “Main” block. On the *General* tab in the *Touch Key* window, select the *Jump to block* and the newly created Savonia block. On the *Text* tab set the *Text* as “Savonia logo”. Click OK to close window.

Open the “Savonia” block and add a *Touch Key* onto it too. On the *General* tab, set the action as *Other function* and select *Returns to previous block*. Set the block’s *Text*

as “Previous”, *Alignment* as *Left* and the *Placement* on the middle of the lowest row. On the *Symbol* tab, set the *Symbol* as *A_LEFT* and its *Placement* on the middle of the top row. Click the *Apply* button and the *Touch Key* should look like the one on the Figure 19.



Figure 19: The “Previous” touch Key including text and a symbol

Task 16: Security Levels

Difficulty: Intermediate
Tool: -
Other equipment: -
Example program number: -

E-designer allows different security levels to be set for different objects. These objects can be either full blocks or function/touch keys. The security levels range from 0 to 8 and they are associated to passwords. In order to use the object the panel user must login in to the security level in question, or higher.

The security level is set on the *Access* tab in the object’s *Properties* window. If the security level 0 is selected, the object is available without user authentication. Create security levels for the program created in the previous exercise. Use at least 3 different levels:

Level 0: Permission to access the Main menu and the Savonia graphic screen
Level 1: Permission to operate the conveyor
Level 2: Permission to access System monitor and Mail

The screenshot shows a 'Password' dialog box with the following fields and settings:

Security level	Password	Confirm question	Comment
1:	pswd1		
2:	pswd2		
3:			
4:			
5:			
6:			
7:			
8:			

Login signal:		I/O	
Logout signal:		I/O	
Login level req.:		I/O	
Current level req.:		I/O	
Login timeout:	3	minutes	
Password RUN/PROG:		<input type="checkbox"/> Automatic login	

Buttons: OK, Cancel

Figure 20: *The Password window*

The passwords are set in the *Password* window found in the *Functions* menu. The password and/or the security level can be commented and an automatic logout timeout set along with other settings as seen in Figure 20.

The *Confirm question* is an additional question that is displayed when login is required (e.g. “Start conveyor?”). The maximum length of the question is 20 characters.

If the *Automatic login* option is selected, a keyboard is displayed when the user tries to access a password protected object. The automatic logout timeout is the time, after which the user is logged out when the terminal is inactive. The user can be logged out also using a *Function key* with its *Other function* selected as *Logout* or using the Logout I/O signal. See E-Designer Help for more information.

Task 17: Localization

<i>Difficulty:</i>	<i>Advanced</i>
<i>Tool:</i>	-
<i>Other equipment:</i>	-
<i>Example program number:</i>	-

The applications for the operation terminal can support up to 10 different languages. Additional languages are added from *Setup > Multiple languages > New language* menu.

The steps to create a multi-language application are:

- 1 Select the number of the languages
- 2 Select the System language name and character set
- 3 Select other languages
- 4 Select the control register to be used for language control. Its value (0-9) determines the application language (0-9) the terminal uses. Register content of 0 shows the first language, 1 the second etc.

The Language Register

The language register must be a word type variable. To add an internal variable to be used in language control, open the Internal Variable configuration window from *Functions > I/O Configuration > Internal Variables*. Add a word device by clicking the arrow up in the numeric updown control (Figure 21).

The language register value can be changed with a touch key. Create a new touch key and on the *General* tab, select *I/O*, write the name of the language register into the first field and select the *Event* as *Sets Analog* (Figure 22). Write the correct *Value* for each language (i.e. 0 for the system language, 1 for the first additional language etc.). On the *Text* tab, write the language's name to the *Texts* textbox and make the desired changes in the other options as well. Make own touch keys for each language used in the application.

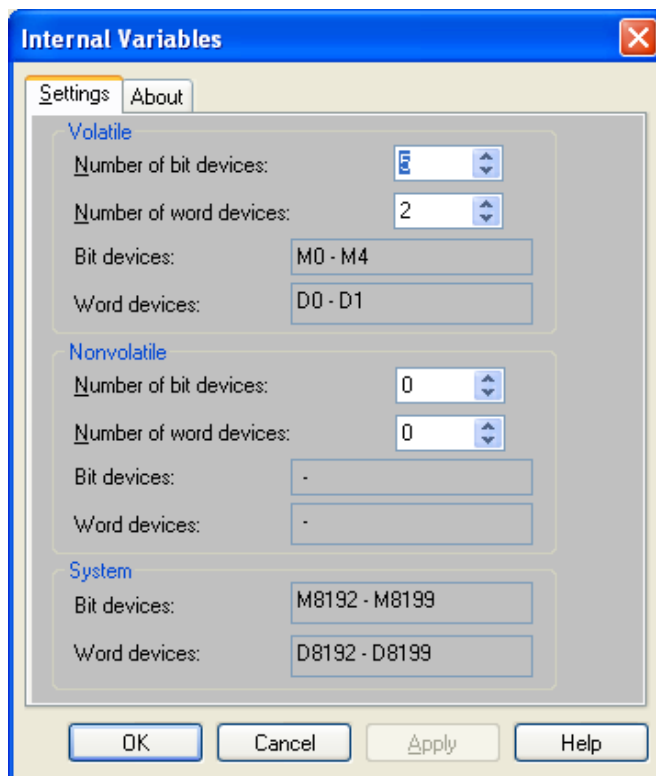


Figure 21: The Internal Variables window

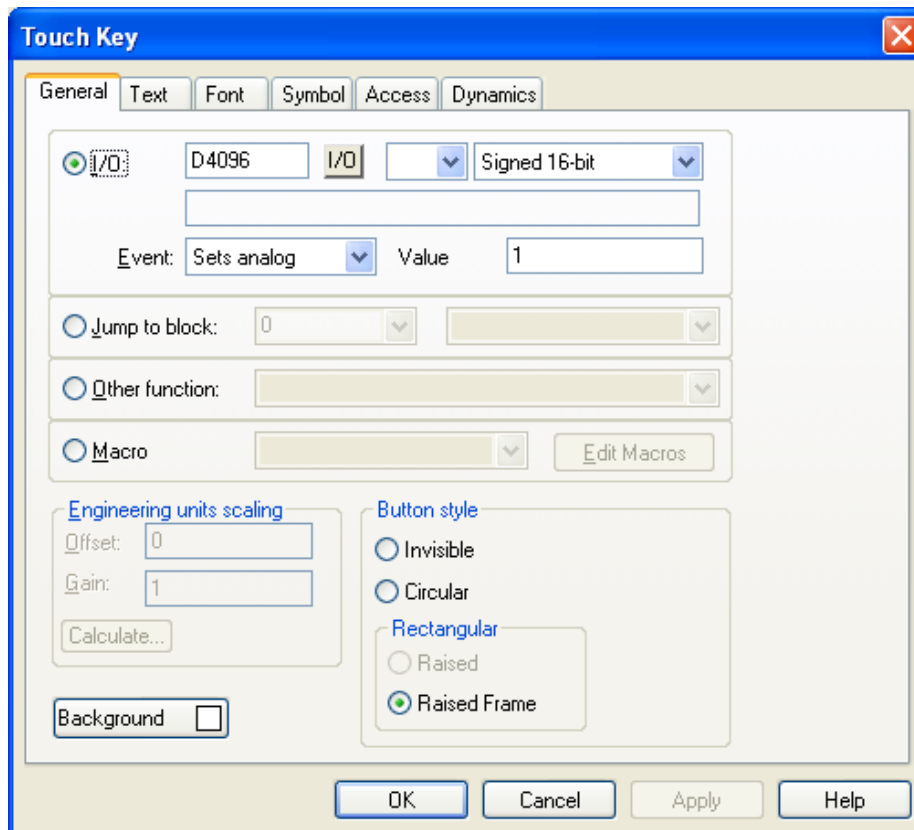


Figure 22: The Touch Key sets the register D4096 value as 1

Translating Texts

Choose *Setup > Multiple languages > Edit* to open the *Application Languages* window (Figure 23). The translations can be written directly to the table. Texts can be searched using the <Ctrl> + F command.

Languages can also be exported from E-designer into; for example, Excel to be translated there and then imported back.

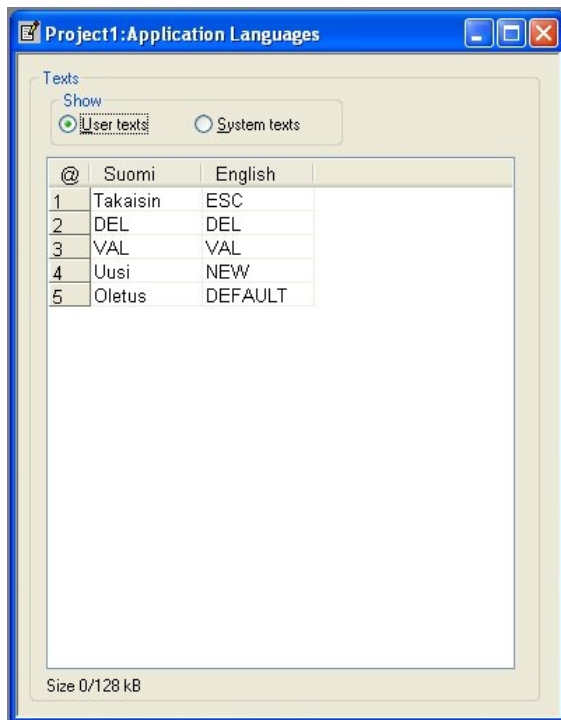


Figure 23: The Application Language window

Testing the Localization

E410 applications cannot be simulated in E-Designer, but the translations can be tested by changing the application language from the toolbar (See Figure 24).

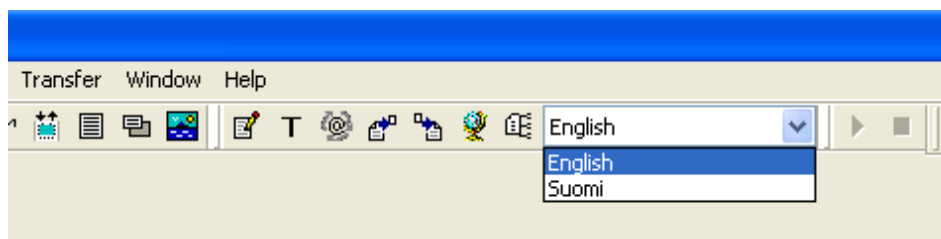


Figure 24: The language selection menu

Intellect 1.5

The machine vision systems for the DVT vision sensors are made using Intellect software. Intellect has an camera emulator which consists of *all* the characteristics of a real DVT vision sensor (e.g. crashing with no reason, randomly losing the connection to emulated hardware...). The machine vision system is built from small modules, *Tools*, each performing one specific task. There are tools for preprocessing of the image, identifying of shapes, measuring of distances etc. The tools can be chained and linked. The final inspection result is the combination of the results of the different tools.

The software can be downloaded from the manufacturer website at www.cognexsensors.com.

Task 18: Going Nuts (with DVT)

<i>Difficulty:</i>	<i>Beginner</i>
<i>Tool:</i>	-
<i>Other equipment:</i>	-
<i>Example program number:</i>	-

After completing this exercise, the student will know the basic operation of Intellect 1.5 software and is ready for the following exercises. This exercise includes the opening of the workspace, emulation of the camera, opening the picture sequence, adding the products, tool layers and tools and saving the system.

Open the Intellect 1.5 software and select “DVT 542C” as the camera to emulate. This is the model of the color camera of the school’s robot work cell. If the Network Explorer is not shown by default, open it from *System > Network Explorer* menu and double click the *Emulators* window open (Figure 25)

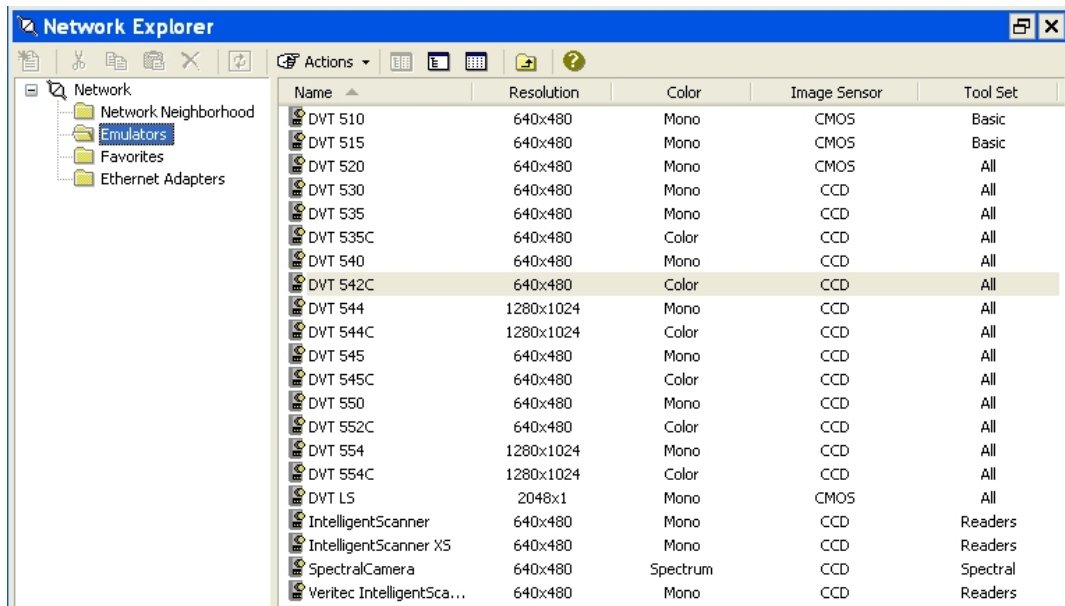


Figure 25: The Emulators window in Network Explorer

Measuring the Hole Diameter

In this exercise a nut is located from the conveyor, it is measured using Line Fit and Circle Fit tools and a script is used to calculate the hole diameter in millimeters. Use the picture sequence from Documents\Robot work cell\Intellect\Training images\nuts\. Open the images by selecting *Image > Configure Sequence > Browse for Images* from the main menu.

The Nut Location

Select *Product > New Product* to create a new product. On the *Properties* window and name the product as “Nut”.

Open *Tool layer manager* from the *Product* menu. Create a new layer by double-clicking the empty space on the right and name the layer as “Location”.

Create a new *Area Positioning* tool from the *Positioning* section of the *Toolbox* (or from the *Tool* menu) by dragging a rectangle over the nut. The *Parameters* window opens. Make the following changes in the *Area Based Positioning Parameters* window (Figure 26):

General: Name: Nut_location

Layers: Check the layer that was created earlier (Nut_location)

Shape: Entire Image on the Search region tabs

Options: Object locate, select a suitable Minimum Match Score

On the *Model Object* tab, set the *Fine...Coarse* setting to medium to decrease the number of the details the tool searches. Click the *Relearn* button and from the box below, select one characteristic and then click the *Display* button to highlight that line from the picture. Decide, which are the characteristics the system should use to make the decision and delete the unnecessary ones.

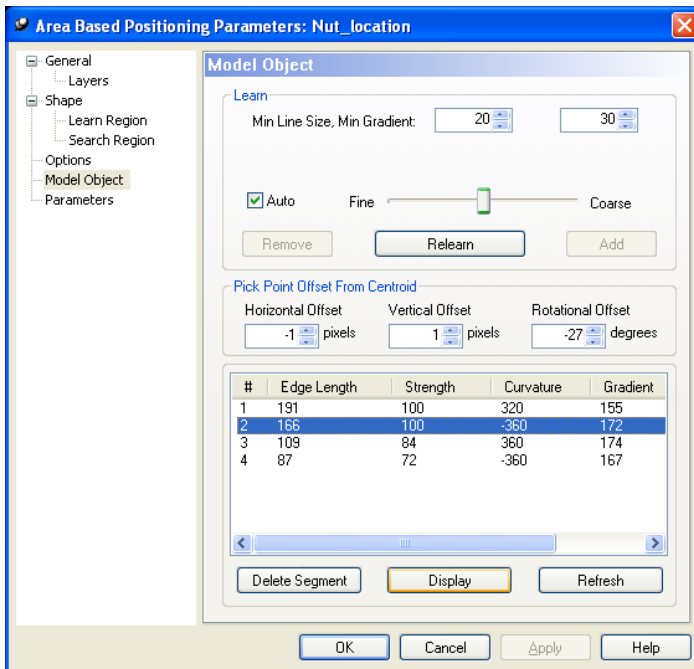


Figure 26: The Model Object tab in the Area Based Positioning Parameters window.

Ensure that the tool finds the nut from all the pictures in all the positions (Draws the green frame correctly, see Figure 27). Adjust the *Pick Point* (Blue cross and the line) correctly on the center of the nut with the *Horizontal* and *Vertical Offset* values on the *Model Object* tab. Use the *Rotational Offset* setting to turn the angle indicator towards an edge.

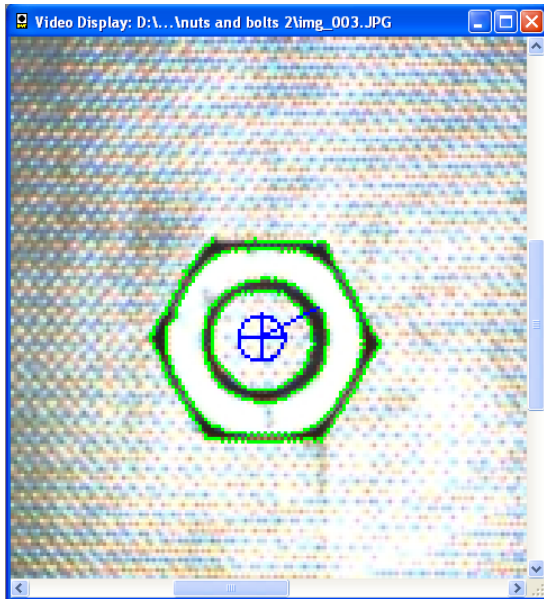


Figure 27: The pick point placed in the center of the nut

Measuring

Intellect can be used to measure distances from the picture. In this exercise, the diameter of the nut hole is measured. Because Intellect measures everything in pixels instead of, for example, millimeters, a pixel/millimeter ratio must be calculated before the diameter can be determined. The ratio is calculated from the outer diameter of the nut, which is known to be 17 mm. The diameter is measured in pixels and the ratio is calculated in a script.

Create a new tool layer named "Measure" and onto it a *Line fit tool* from the *Positioning* menu. The tool is defined by drawing a line over the edge and then expanding it parallel to the edge (Figure 28).

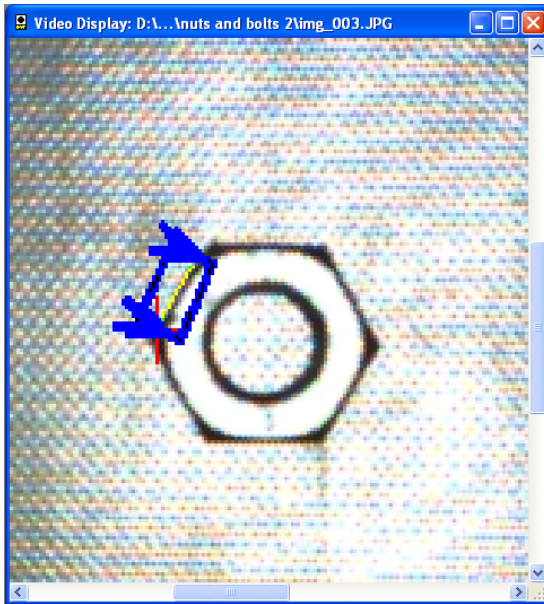


Figure 28: Defining the Line Fit tool

Name the tool as “Linefit 1” and carry out the following changes in the *Line Fit Parameters*:

Position Reference: Nut_location

Layers: Measure

Options

Task: Line Fit

Threshold: Intensity: Auto Bimodal

Scan Line Edge to Consider: 1st Edge Found

Edge Type to Locate: Any Edge

Pass:

Minimum contrast: 15.00 %

Create a new similar tool (“Linefit 2”) to search the opposite edge. Adjust the settings if needed. Add a new *Measure with Points and Lines* tool from the *Measurement* menu. Make the following changes in the options:

Select an operation: Distance

Reference Line 1: Tool: Linefit 1

Reference Line 2: Tool: Linefit 2

The radius of a round object can be measured with a specific tool. It is named *Circle Fit* and it is found in the *Positioning* menu. The tool is applied by selecting it, clicking on the center area of the circle to be measured and then dragging outwards until the actual inspection area's inner edge is reached. There the mouse button is released and the cursor moved to the inspection area's outer edge. Clicking on that distance finishes the area selection. See Figure 29. The location can be finetuned using the tool's settings.

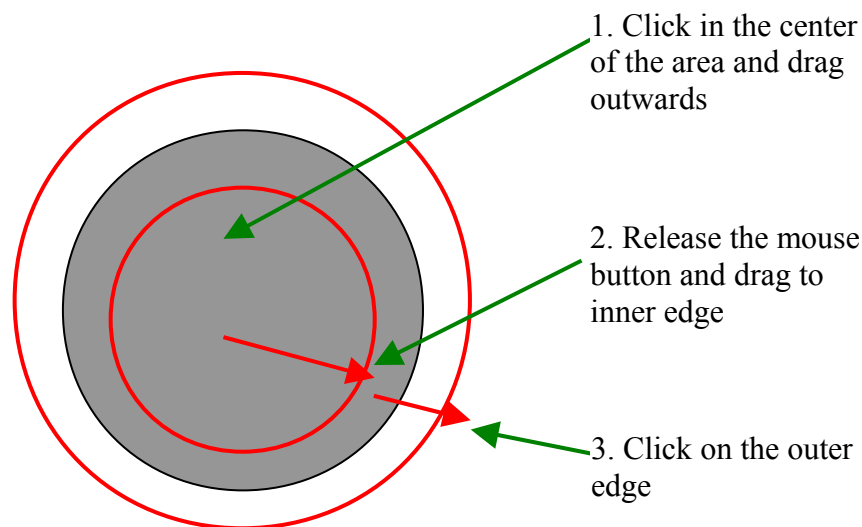


Figure 29: Defining the Circle Fit tool

Set the tool's *Position Reference* as *Nut_location* and adjust the settings so that the hole is found correctly from all the pictures in the sequence.

Hole Diameter Measure Script

In Intellect 1.5 also user-defined scripts can be used. They are added just like any tool from the *Tool > New Tool* menu. Intellect has a built-in script editor (Figure 30). The script editor is opened by clicking the *Script Tool Parameter* window's *Edit* button. The language is object oriented and the syntax is very similar to C++. The language contains all common variable types, flow control structures and also an option for user-defined functions.

A script can have many outputs, which are shown in the *Result Table*. The outputs are selected on the *Outputs* tab in the *Script Tool Parameters* window.

Create a new *Script* tool, name it “Measure” and on the *Output* tab, select output as *String*. Click the *Edit* button on the *General* tab to open the editor.

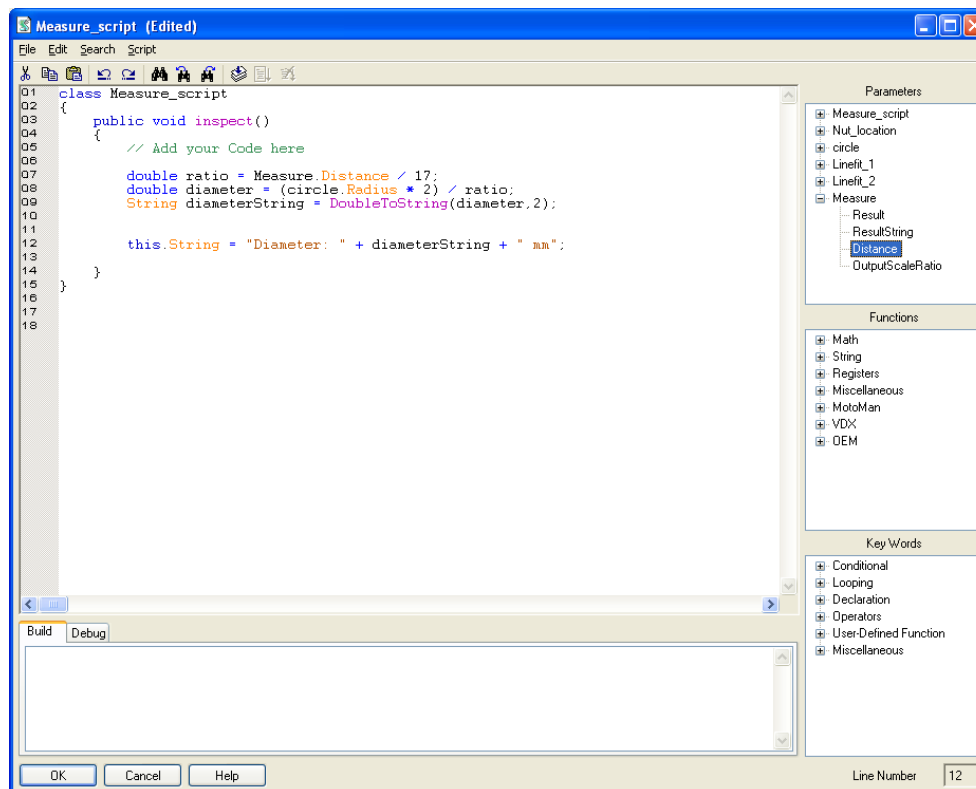


Figure 30: The script editor interface

Available parameters, functions and key words can be found in the tree view boxes on the right-hand side. They can be added to the code by double-clicking them. Use the *Measure* tool's parameter *Distance* to calculate the pixel/mm ratio and then calculate the diameter from the *Circle* tool's *Radius*. Compile the script by clicking the *Compile* button from the toolbar or from the *Script* menu. The result string is displayed in the *Result Table* and the calculated diameter can be used, for example, as a parameter for other scripts.

Task 19: Bolt Action

Difficulty: Intermediate
Tool: Hand
Other equipment: M8 bolts
Example program number: -

In this exercise, the length of a bolt is measured. The pixel/mm ratio is calculated using a calibration image instead of a known distance from the object, as was done in the previous exercise. After measuring the bolt, the pick point and the bolt angle are sent to the robot and the bolt picked from the conveyor.

Use the image sequence from path Documents\Robot work cell\Intellect\Training images\bolts\

Create a new product *Bolt* and three tool layers: *Preprocessing*, *Location* and *Measure*. Open the image “short bolt 1”. Locate the bolt from the image using a Area Positioning tool.

Bolt Location

Make the following changes in the *Area Based Positioning Parameters* window:

General: Name: Bolt_location

Layers: Check the layer that was created earlier (Location)

Shape: *Entire Image* on the *Search region* tabs

Options: *Object locate*, select a suitable *Minimum Match Score*

Model Object: Set the parameters on the *Pick Point Offset From Centroid* so that the blue x-mark in the circle is on the center of the head of the bolt and that the line starting from the mark aligned with the bolt.

Ensure that the tool finds the bolt from all the pictures in all the positions (Draws the green frame correctly).

Preprocessing

Onto the preprocessing layer, add a *Preprocessing* tool *Filter* with the following parameters:

General:

Name: Fill_light_holes

Position Reference: Bolt_location

Shape: Change the values so, that the bolt is inside the frame in every picture

Options: Preprocessing Operation: Morphology

Morphology type: Fill light holes

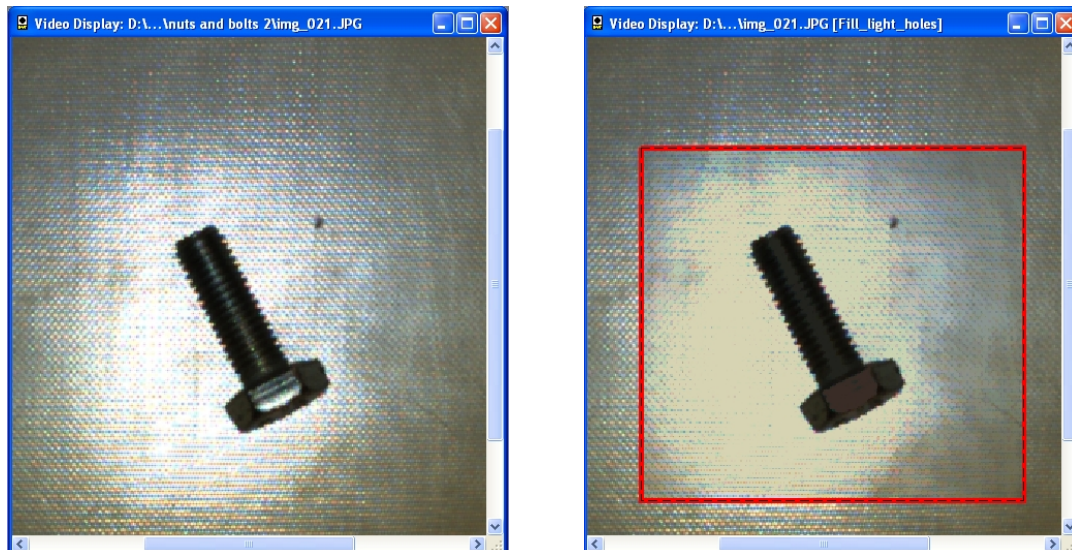


Figure 31: The bolt before and after applying the Fill Light Holes filter.

Measuring the Bolt Length

Create two *Line Fit* tools named *Linefit_1* and *Linefit_2* to search the ends of the bolt. Set their *Position Reference* as *Bolt_location*. Set the *Image Reference* as *Fill_light_holes*. This way, the tool uses the preprocessed image (see Figure 31). Adjust other settings if needed.

Create a *Measure with Points and Lines* tool and make the following changes in the options:

Select an operation: Distance

Reference Line 1: Tool: Linefit 1

Reference Line 2: Tool: Linefit 2

Sending the Pick Point to the Robot

The pick point data can be sent to the robot from the DVT Vision Sensor via Ethernet connection using DataLink. Open the *Communication Settings* from the *System* menu. Open the *Ethernet Terminals* and add an new item (“Terminal 1”). Open it's properties by double-clicking the item and ensure the settings are as follows:

Enabled	True
Start at Powerup	True
Terminal Type	SERVER
Port	3247

Open the *DataLink Settings* window from the *Product* menu. Add a new item (“DataLink String 1”) and open the *String Expression Editor* by double-clicking the item. See figure 32. The parameters from the tools are displayed in the box on the right and can be added to the *String Definition* box by selecting the desired parameter and clicking the *Insert* button.

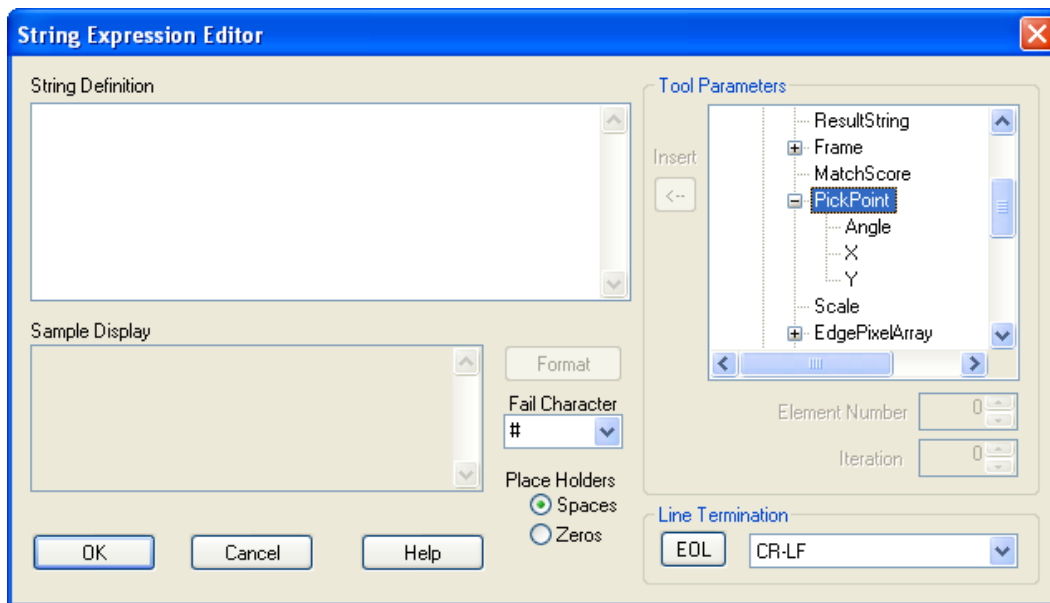


Figure 32: *The String Expression Editor*

The pick point is sent to the robot as a position variable. A position variable consists of 8 values, which are: x, y and z coordinates, the angle, the angle of the J5 joint, the angle of the J4 joint (always 0), the angle of the additional axis, the angle of the second additional axis (always 0). Position variable can be a following string:

10 P1 = (100, 30, 0, 180, 35, 0)(0,0)

However, this position is not usable in this form. The coordinates are relative to the camera's origin and measured in pixels while the robot's origin is different and the coordinates measured in millimeters. Thus, the actual coordinates must be converted into a correct form before sending. This can be done by subtracting the camera coordinate from the maximum value of each axis (480 or 640).

Note, that the camera's x-axis is inverse to the robot's y-axis and the robot's x-axis to the camera's y-axis (see Chapter “The calibration of the coordinate system” in the Additional Instructions). The pixel/mm ratio can be calculated in this conversion script using a known distance. To do this, open image “calibration 1” from the “Calibration” folder. Measure the length of the 10 mm bar using a *Measure Along Line* tool and calculate the pixel/mm ratio.

The pick point data the camera sends can be read in a variable in the robot's program with the following code (Table 12):

Table 12: Input position data from camera

```
10 C_COM(4)="ETH:192.168.0.125,3247"  
20 OPEN "COM4:" AS #1  
30 INPUT #1, P1  
40 CLOSE #1
```


Task 20: Quality Control

Difficulty: Advanced
Tool: Vacuum gripper
Program number: 41
Example program number: -

The aim of this task is to inspect the assembly of the product using machine vision and remove the faulty products from the conveyor. The product (Figure 33) contains a 14 mm wide recess, two holes, a text label and a Datamatrix label. There are 10 objects, of which 3 are defect-free.



Figure 33: An example of a fault-free product

The inspection items are:

- 1 Diameters of the holes
- 2 Distance between the holes
- 3 Distance of the holes (from the edge)
- 4 Holes drilled all the way through
- 5 Number of the screws
- 6 The 2D code (part number)
- 7 OCR text

The characteristic of a proper piece are:

Diameter of the holes	8 mm
Distance of the holes	20 mm
Hole distance from the edge	7 mm
Datamatrix	Number from 1 to 10
OCR	3 letters, 5 numbers

The faulty products are lifted from the conveyor into the waste chute. The passing products are palletted onto the side table.

Create a Intellect system to inspect the products. Send the needed data to the robot via DataLink. Program the robot to operate the conveyor and to move the pieces.

See Chapter “E-designer” in Additional Instructions for more information about Data Exchange between the robot and the PLC.

Additional Instructions

The equipment needed in the exercises

Laptop PC

Laptop contains the required software, additional files and the example solutions to the exercises. User account has path

C:\Users\user\My Documents\Robot work cell\

containing the folders for the accessories such as Server Console & Client Console and the webcam software (Robot Camera and Camera Viewer). All the images used in the exercises are located in their own folders. There are also folders for the user made exercise solutions and manuals for different software.

Mosaic tiles and mosaic platform

There are dozens of different colored mosaic tiles available to be used with the robot. Use the mosaic platform to keep the pallets organized. Mosaic tiles and platforms can be found from the compressor room located next to the robot cell. The chipboard on the top of robot cell contains also a platform attached, and because of the firm surface, it is easier to use.

Plastic pieces

There are two sets of plastic pieces. Both of them, as well as the another chipboard with the grids for the plastic pieces, can be located on the top of the workcell.

Set 1: Black, cylindrical plastic pieces

Set 2: Green, rectangular plastic pieces

Quality control products

Quality control products can be located under the worktop. The following products should be passing the check:

BIA57222

CAN62768

CUB40933

OLM10339

And the following products should be failed:

CER91258 – Hole not drilled through

CRE10345 – Screw missing

FEL80045 – Hole too close to the side of the product

GIA75661 – Hole too large

KON32321 – Hole too far from the side of the product

TRE20023 – Hole too small

Nuts and bolts

Exercises also use regular M8 nuts and bolts. There is a ready set of them in the compressor room. You can also use other sizes, if those are not available, but it means that you have to make a new machine vision systems for them.

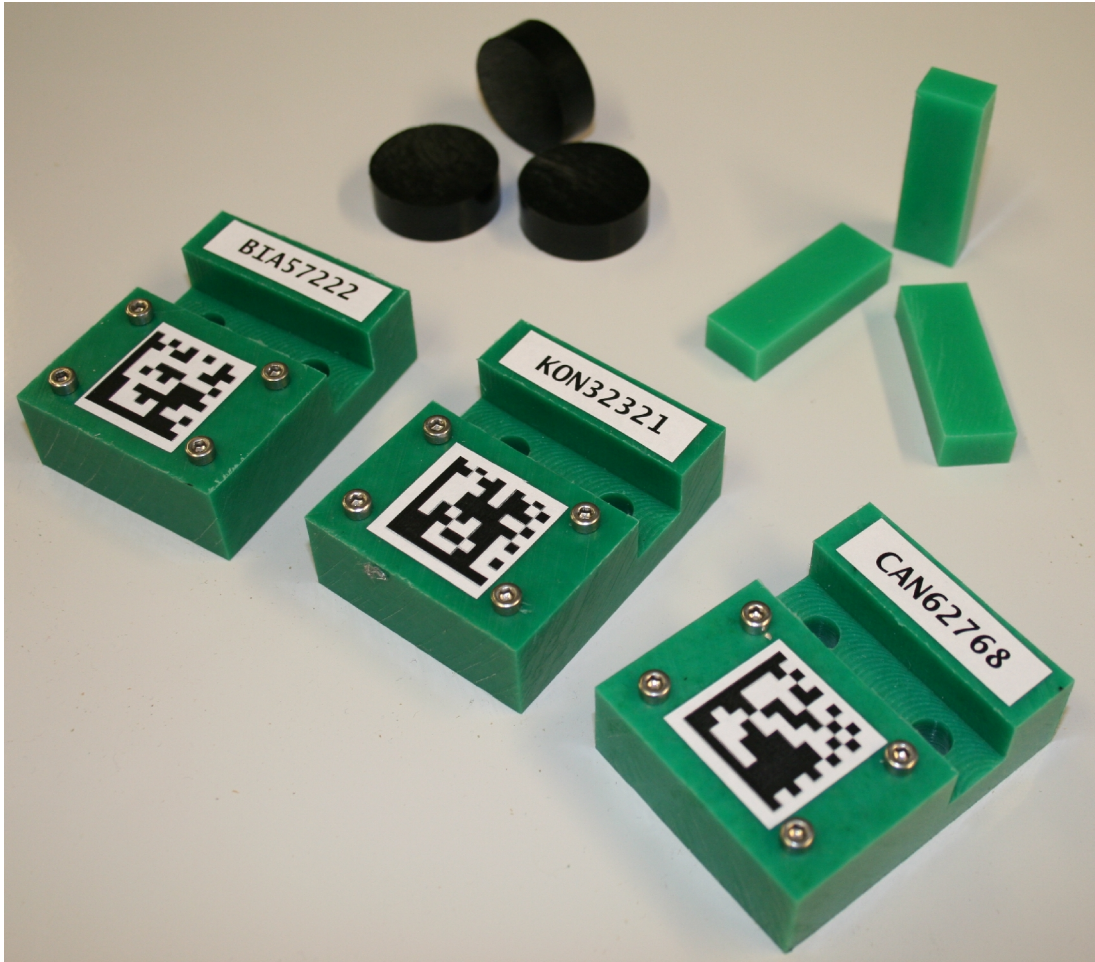


Figure 34: The equipment used in the exercises

Mitsubishi RV-2AJ

Melfa-Basic

The top-30 functions to get started with Melfa-BASIC

CLOSE

Closes a file or a communications line

CNT

Continuous movement. Designates continuous movement control for interpolation. Shortens operation time.

DEF PLT

Define pallet

DEF POS

Define position

DEG

Degrees. Converts the unit of angle measurement from radians (rad) into degrees (deg)

DIST

Distance. Calculates the distance between two points (position variables).

DLY

Delay

ERROR

User defined error

FOR...NEXT

Repeat

GOSUB (RETURN)

Subroutine jump

GOTO

Jump

HLT

Halt

HOPEN/HCLOSE

Hand open/ hand close

IF...THEN...ELSE...ENDIF

If-statement

INPUT

Input data

LEN

Returns the length of the string.

M_TIMER

Timer

MID\$

Returns the string of the specified length from the specified position of the string.

MOV

Move

MVS

Move, linear interpolation (use with CNT)

ON...GOSUB

Subroutine jump according to the value

ON...GOTO

Jump according to the value

OPEN...AS

Open a file or communication line

PRINT

Outputs data

SELECT...CASE...BREAK...END SELECT

Select statement

SKIP

Skip while moving

SPD

Speed specification during joint interpolation movement

WAIT

Waits for conditions

WHILE...WEND

Conditional statement

WTH

With. Additional instruction of movement instruction

Changing the tool

As of spring 2009, Savonia UAS has two different tools for the robot arm. Both are pneumatic, one is a vacuum gripper and the other is a two-fingered hand. The tool is held in place by the magnetic valve M_OUT(10). The manual tool change procedure is as follows:

- 1 Use the TB to position the tool carefully into the rack
- 2 Release the tool by setting magnetic valve M_OUT(10) as 0. The tool is dropped.
- 3 Move the robot above the another tool, and with care, position the arm so that the tool is in it's place. Make sure, that the tool is in correct angle etc.
- 4 Set M_OUT(10) as 1. The tool connects to the arm.

Alternatively, there are two complete tool change programs. The program number 87 is used to change tool manually. Server Console is used to select the executed operation. The another tool change program (86) can be run as a subroutine if any other program needs to change the current tool.

Changing the batteries

The robot's servomotors save their absolute positions while the controller is turned OFF. For this, there are total of 6 batteries, 5 of them in the base of the robot arm and one in the additional axis unit. The batteries are 3,6 V lithium batteries. These batteries must be replaced once a year or the servo position data is lost and the robot must be recalibrated. The controller also has a battery life timer, which shows how many hours the batteries have been in use.

To change the batteries, the cover of the lowest part of the robot arm must be removed. Turn the J2 joint into a position, which allows removing the cover and using a hex wrench to open the screws. Remove the battery cover. Detach the battery bracket and the cables. Dismount the batteries from the bracket. The cables are soldered directly onto the batteries. Replace the batteries and assemble parts in inverse order.

CAUTION!

While replacing the batteries, the encoder position data is saved by the power supplied from the robot controller. If the controller is turned OFF or the cable disconnected, the position data will be lost.

Please refer to the **ROBOT ARM SETUP & MAINTENANCE INSTRUCTION MANUAL** for more information.

About COSIMIR Industrial

RCI Explorer

The RCI Explorer (Figure 34) provides versatile information about the robot during operation. Everything from the robot's position to motor currents is available to be viewed and recorded to file (Figure 35). RCI Explorer can also be used to modify some of the values, such as parameters.

The RCI Explorer can be opened by clicking the RCI Explorer icon on the toolbar. If the values are not displayed when the window is opened, right-click the item on the tree view on the left and select “Refresh”.

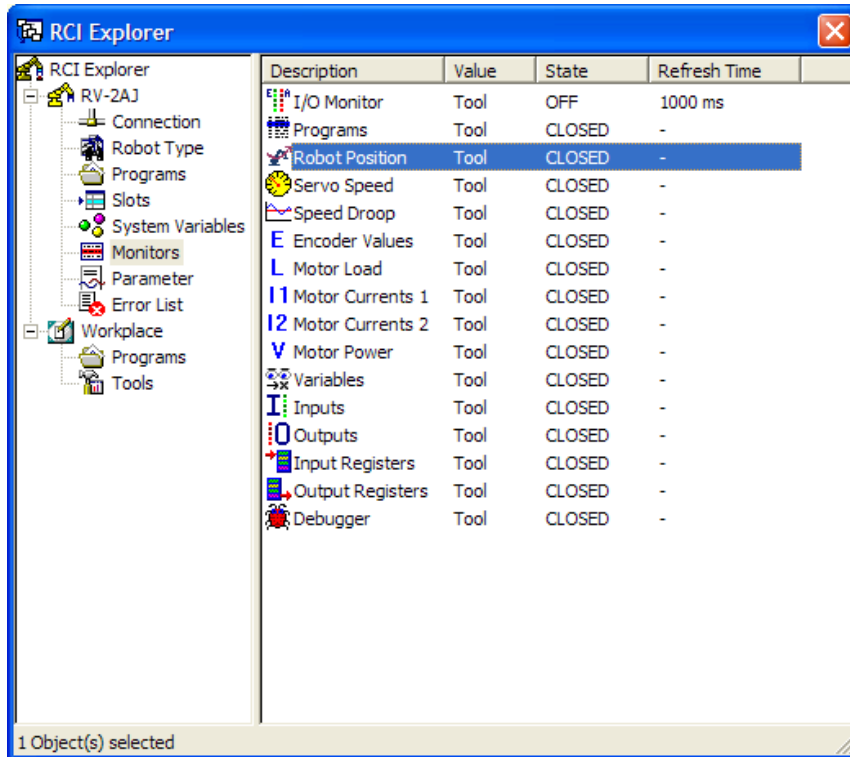


Figure 35: The Monitors window in RCI Explorer

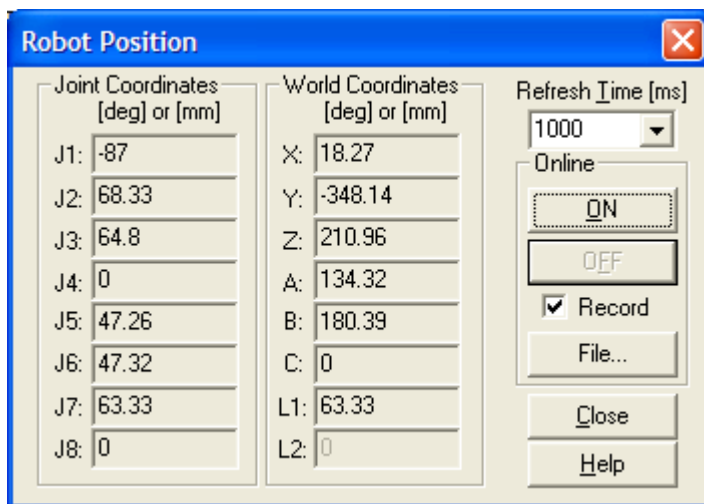


Figure 36: The Robot position window displaying the Joint and World Coordinates. The values can be recorded to a file by clicking the “ON” button on the right-hand side.

GX IEC Developer

Connecting the PLC and the PC

Melsec FX1N PLC controls the conveyor. It can be programmed with Mitsubishi GX IEC Developer software.

To start new project, manually create a new project folder somewhere, GX IEC Developer can't create new folders. Click *Project > New*. Select "FX series" and "FX1N" type for PLC and proceed. Next, select *Project Structure* and click OK. As *Prefix* for *Main tasks* type "Conveyor" and click *Next*. *Cyclic Main tasks* should be set to 1 and we don't want to modify task naming, so select "No, automatic naming is just fine!". Now select "FBD" (Function Block Diagram) and click *Next*. Type "Conveyor" again to name the selected *POU* and proceed. Now, select "No, skip this step" and click *Finish*. To add variables to program, double click on *Global_Vars*. In *Class* row you can select whether to add a variable or constant. *Identifier* is name of the variable and type can be any of common variables (Figure 36).

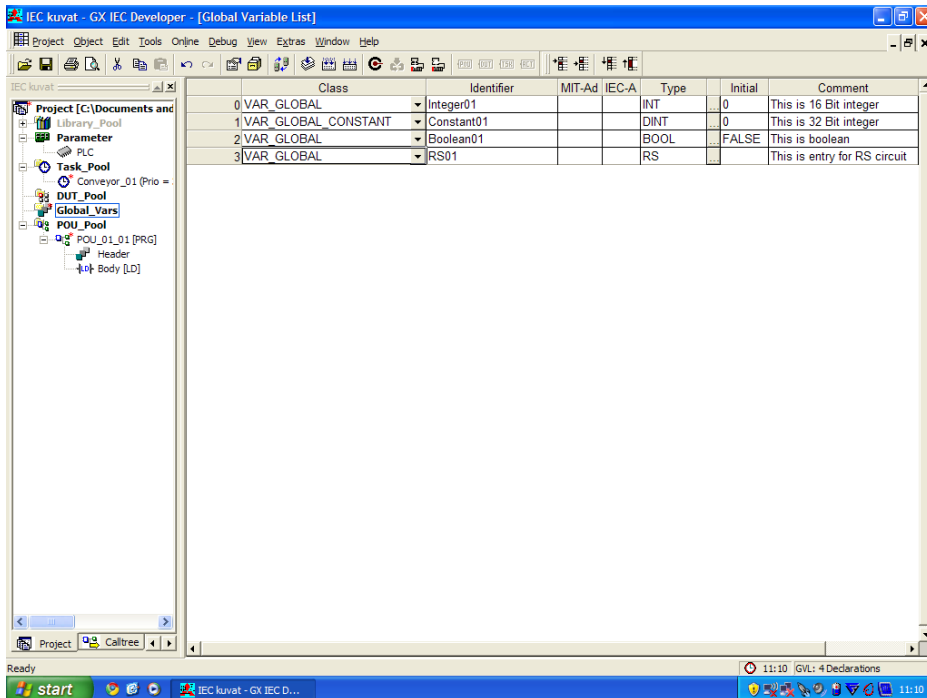


Figure 37: Inserting variables

To create a function block diagram, browse treeview to *POU_Pool* > *POU_01_01[PRG]*. Double click on *Body[FBD]*. You can add function blocks with *Add Function Block* that can be found in the toolbar. Blocks can be connected with line tool and inputs and outputs can be inserted with variable buttons. All buttons are surrounded with red in Figure 37.

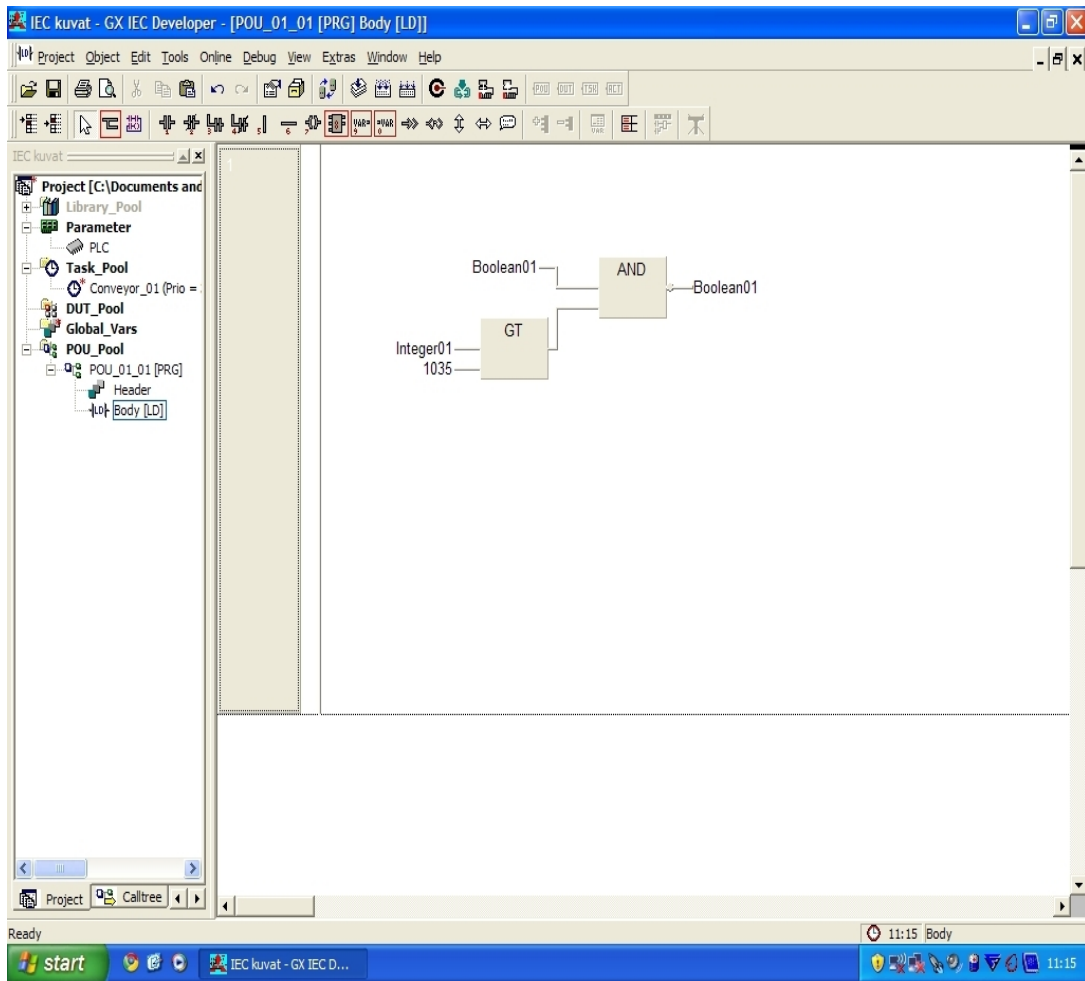


Figure 38: *Creating Function Block Diagram*

The program is transferred by connecting the PC directly to PLC with cable which can be seen in Figure 38. *Download* and *Upload* buttons are in the toolbar. In the *Online* menu there is a useful *Monitor Mode* to debug program from PC as it is run on PLC. *GX Simulator* is also great tool to test own programs. These are shown in the Figure 39.



Figure 39: The cable and the USB-to-Serial adapter used to connect the PLC and PC

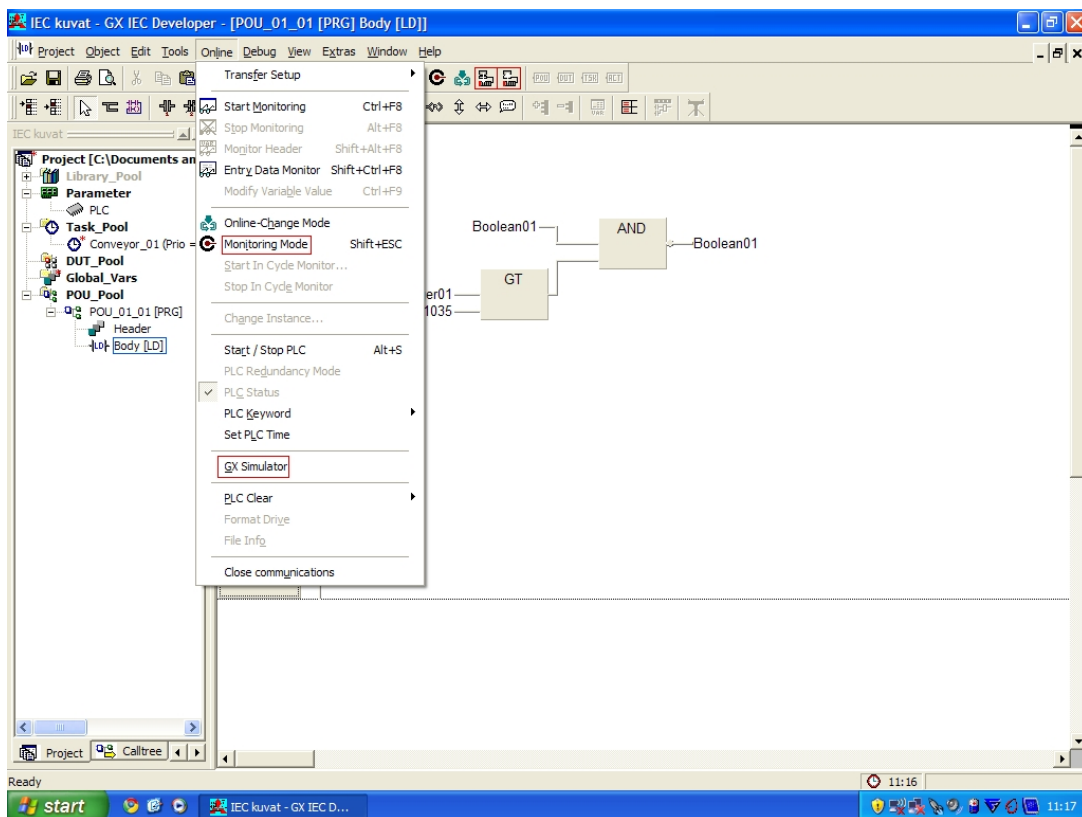


Figure 40: Testing and transferring programs

E-designer 7

Namelists

To easily handle the signals of your programs, they can be listed in namelist. Namelist can be opened from View – Namelist (namelist window in Figure 40). Signals can be named and commented individually and data types and indexing numbers can be set. Remember to choose right controller system to name correct signals!

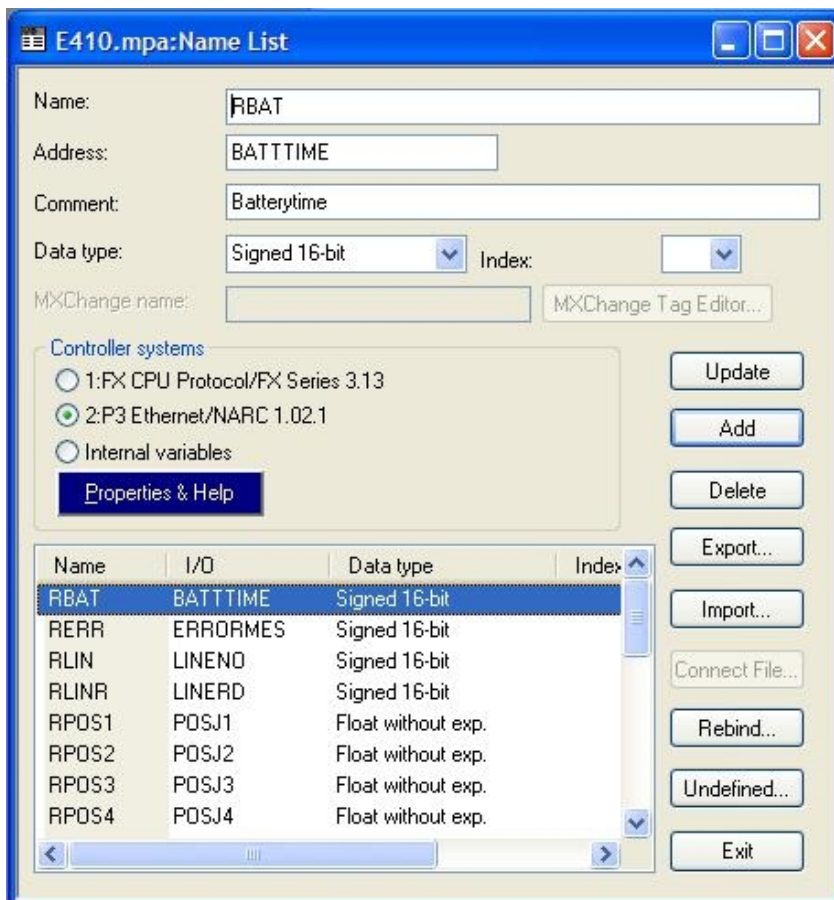


Figure 41: Setting up namelists

Data exchange with E-Designer

Controller outputs can be linked to conveyor logic, for example, to use the conveyor with warning light outputs. This can be done using E-Designer's Data Exchange property. To get to the data exchange screen, open *Functions* menu and click *Data Exchange*. There you can set start signals for I/O #1 and I/O #2. Below, you can set

flow triggers. This means, that when you turn on the triggering signal, value of the both start signals is changed to same, according if you are changing from #1 to #2 or #2 to #1. Figure 41 shows example where robots M_OUT(6) signal (red light) controls the conveyor. When M_OUT(6) value is changed, M1 is changed to same as M_OUT(6).

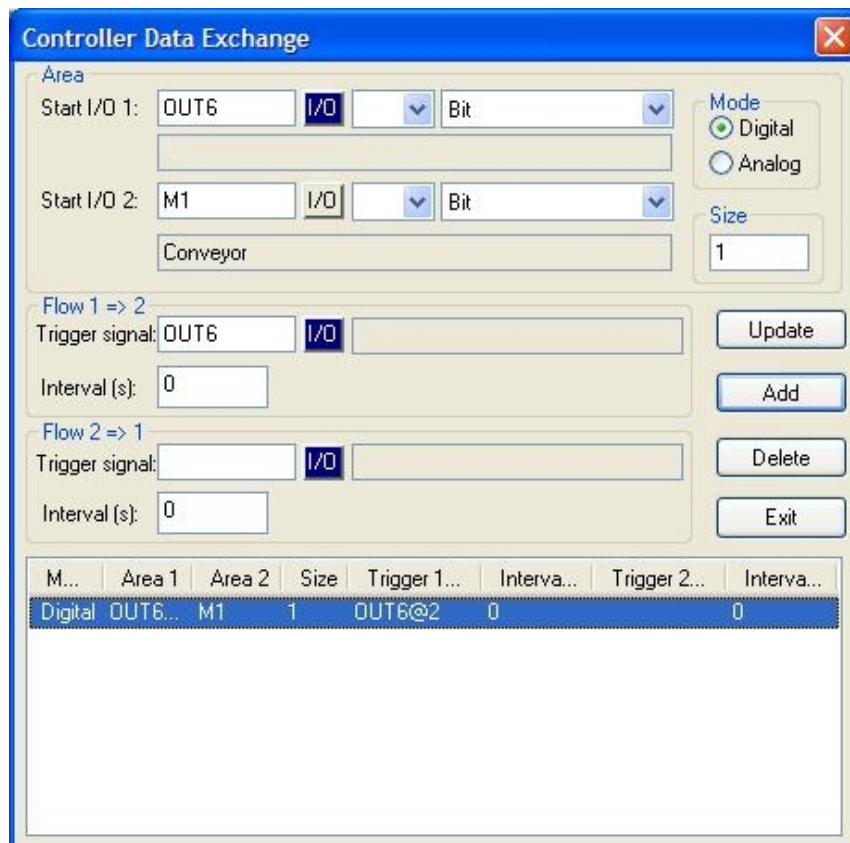


Figure 42: Setting up triggering signals

Graphics

The E410 operation panel can also display user made graphics. The resolution of the display is 320 x 240 pixels. The panel supports the most common picture formats, like TGA, JPG, BMP and TIF. For the best result, the pictures should be black and white; grayscale images tend to be left too pale. Lighter shades can be created with halftone techniques as shown in Figure 42.

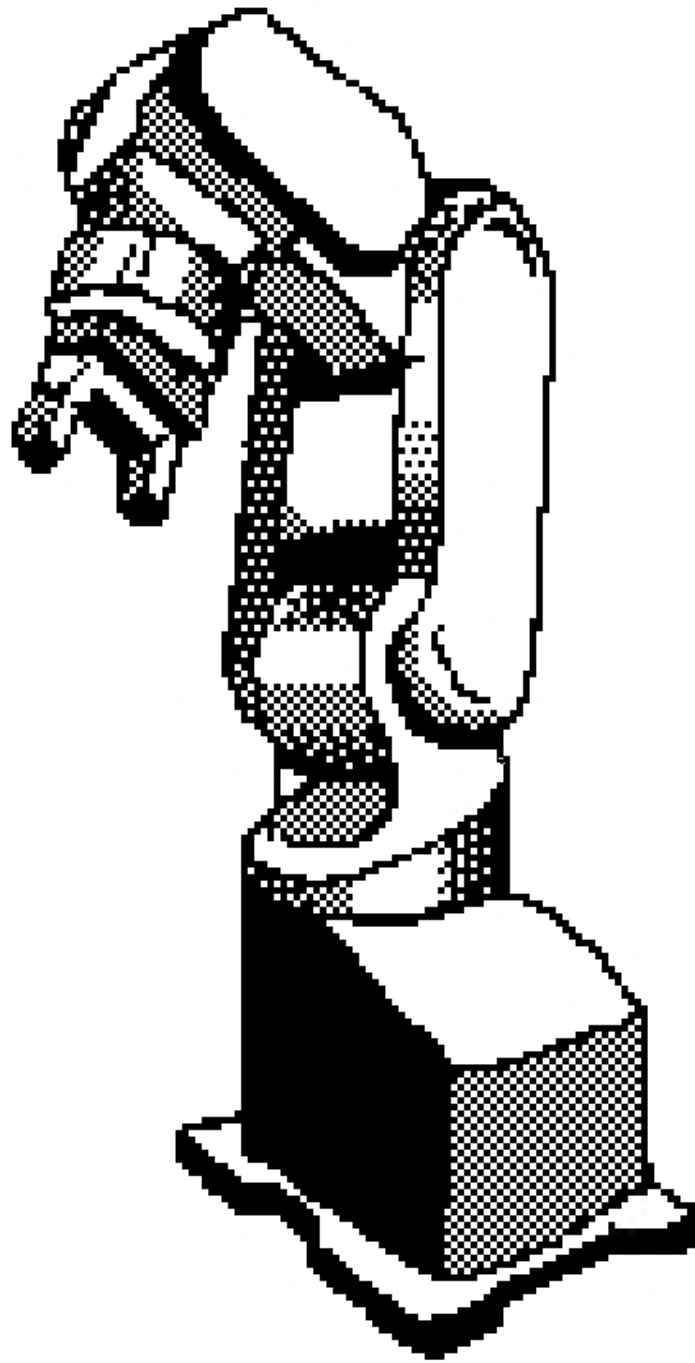


Figure 43: A magnified version of a 320 x 240 pixels image where the checkered areas appear as different shades of gray when viewed in normal size.

User-created graphics can be added to projects by adding a *Digital Symbol* from the tool bar in the bottom of the main window (Figure 43). From the *Digital Symbol* window, click *Select*, then choose the *User-created* from the *Select Symbol* window and click *Import* to select the image from file.

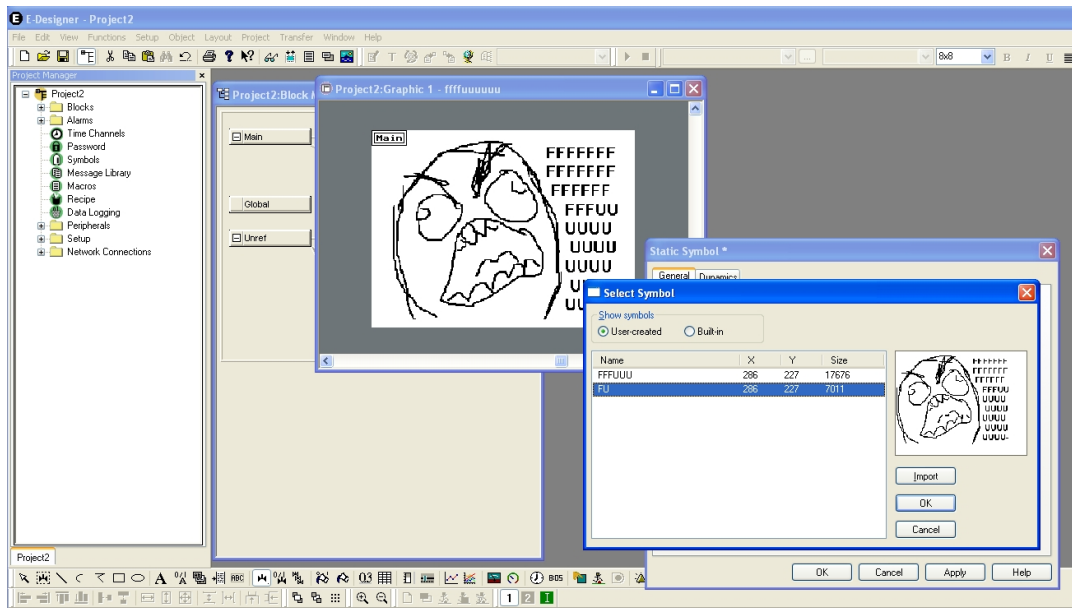


Figure 44: Adding user-created graphics

Intellect 1.5

About the workspace

Use the *View* menu to select the window you want to keep open. The windows can either be kept floating in the main work area or docked, when they are arranged into a set of tabs on the edge of the screen. Right-click the window's titlebar to dock the window. The windows can be docked on the right and left sides and on the bottom of the screen, as in Figure 44, where there are total of 7 different windows arranged in two groups of tabs.

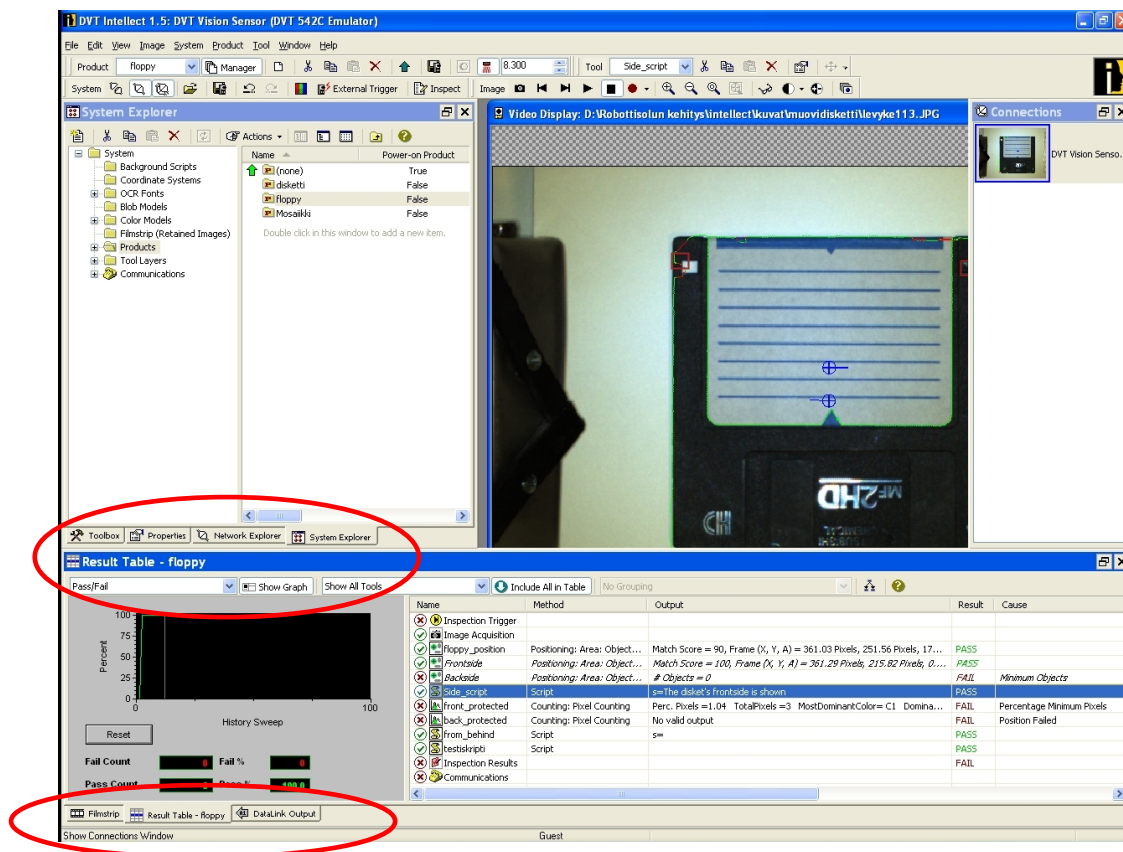


Figure 45: Docked windows in Intellect 1.5

The *System Explorer* and the *Network Explorer* are not found in the *View* menu, but in the *System* menu. Also these windows behave as other windows (i.e. they are dockable).

Tool Referencing Diagram

The *Tool Referencing Diagram* is an useful tool, which represents the DVT system and the tools and their relations visually (Figure 45). The Tool Referencing Diagram can be found in the *Tool* menu.

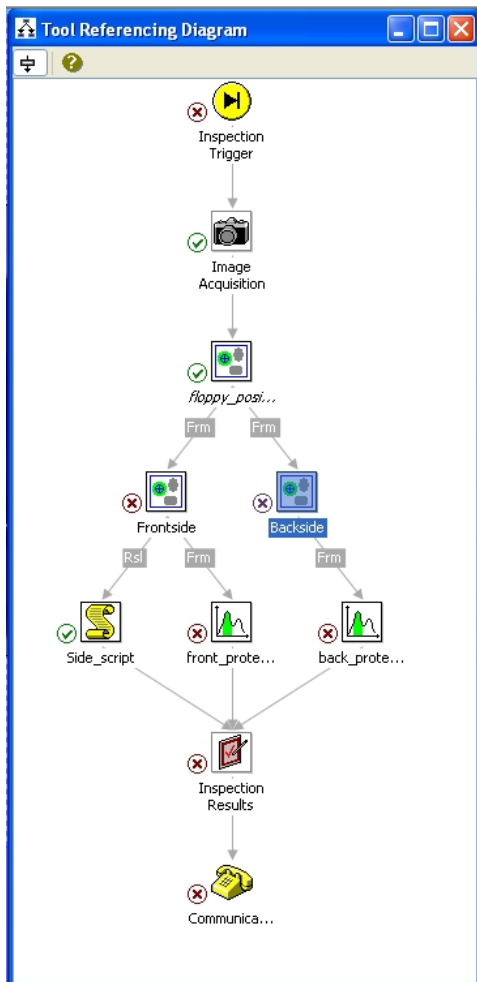


Figure 46: The Tool Referencing Diagram

About recording the sequence

The images are saved to file by opening the *Record* menu by clicking the arrow icon next to the record button on the toolbar. Select the *Set Up Recording* to open the settings window.

In the “Set Up Recording” window (Figure 47) you can choose the destination directory, image type and the filename prefix. The image number settings allow to record images with the same prefix into the same directory without overwriting existing images in that directory.

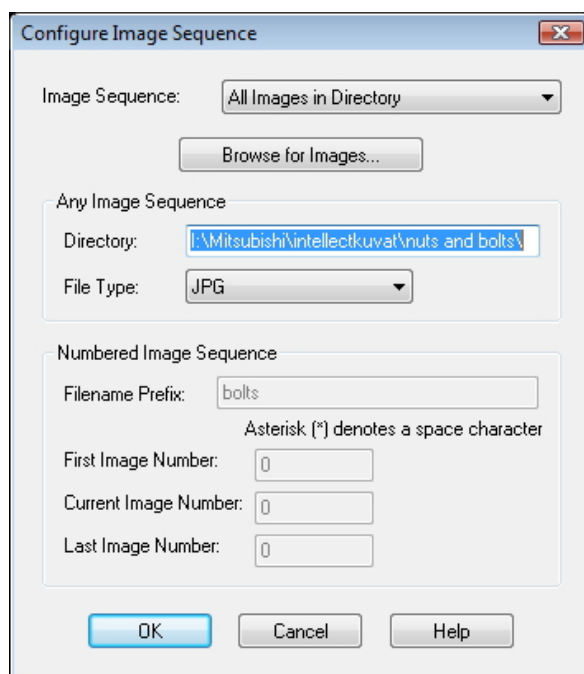


Figure 47: *The Set Up Recording window. The images can be recorded in different image formats.*

The calibration of the coordinate system

Calibration grids can be downloaded from the Cognex website <http://www.cognex.com>. Also detailed instructions for the use of the grids are available in the same address.

Print the 10 mm calibration grid from DVT website and take a picture of it using the camera to be calibrated.

Open the *Coordinate Systems* window from the *System Explorer*. Add a new item (“Transform 1”) and right-click on it. Select *Calibrate Coordinate System*. Calibration takes a few moments, after that a confirmation message appears. Information about the coordinate system is shown in the Transform 1's *Properties* window.

To successfully use the coordinates send by the camera to move the robot, the robot's coordinates must somehow be linked to those of the camera's. The best way to do this would be to station the robot's tool precisely above the upper right corner of the area the camera sees (See Figure 36). That point will from now on be referred to as “camera's origin”

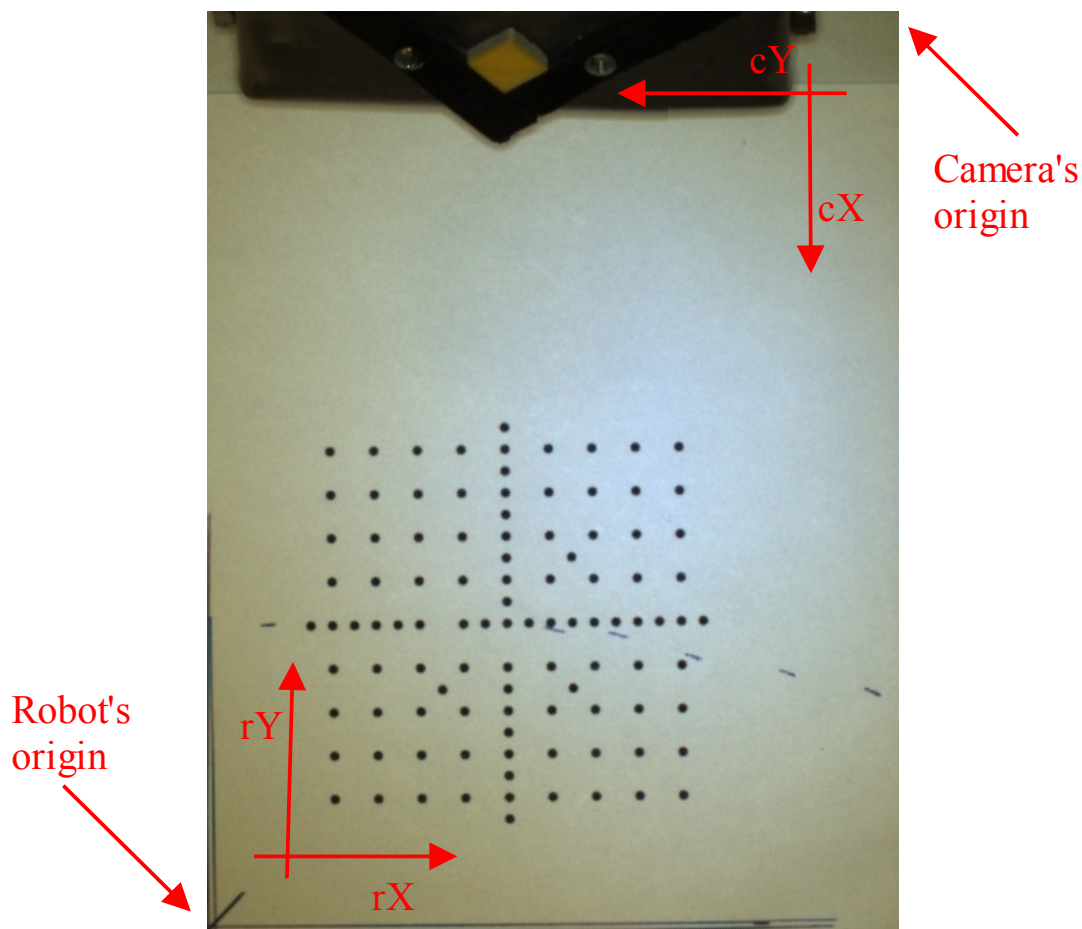


Figure 48: DVT 100 mm calibration grid and the coordinate systems.

However, the robot is not able to reach that point, so the only option is to position it above the lower left corner and then convert the coordinates before using them. This point will from now on be referred to as “robot's origin”. As seen in Figure , the robot's x-axis (“rX”) is reverse to the camera's y-axis (“cY”) and the robot's y-axis (“rY”) is reverse to the camera's x-axis (“cX”).

Explanation of the conversion script

The robot coordinates (“rX” and “rY”) are calculated by subtracting the pick point coordinates from the maximum values of the camera's coordinates (480 for the camera's y-axis and 640 for the camera's x-axis) as show on lines 4 and 5 in listing. After that, the robot coordinates are converted from pixels to millimeters using the pixel/mm ratio 3.6169 to be used in the *DataLink String* that is sent to the robot.

Listing 8: Pick point conversion script

```
1    double cY = Nut_location.PickPoint.Y;
2    double cX = Nut_location.PickPoint.X;
3
4    double rX = 480 - cY;
5    double rY = 640 - cX;
6
7    this.Point.X = rX / 3.6169;
8    this.Point.Y = rY / 3.6169;
```


Name: _____ Class: _____

Points from the exam: _____ / 30 Grade of the exam: _____

Points from exercises: _____ / 120 Grade of the course: _____

1. Explain the following terms briefly

- a) Tool Referencing Diagram
- b) Tool Coordinates
- c) RCI Explorer
- d) DataLink
- e) Data Exchange
- f) OCR

2. What is singularity and when does it occur? Why should it be avoided? How can it be avoided?

3. Explain the difference between linear and joint interpolation. In what situation would you use linear interpolation for moving the robot? In what situation would you use joint interpolation?

4. Explain the difference between Joint jog mode, Tool jog mode and XYZ jog mode

5. Explain the following robot parameters, commands and status variables:

- a) NETIP
- b) MEXTL
- c) C_MECHA
- d) RELM
- e) OVRD
- f) XSTP

6. Explain the usage of the TOOL function. (4 p.) What is the alternative for using the TOOL function? (1 p.) What is the system variable used in this method? (1 p.)

7. Find and correct the errors in the following code samples.

1)

```
10 M1=1
20 *LOOP
30 IF M1<10 THEN
40 M1=M1+1
50 GOTO *LOOP
60 ENDIF
```

2)

```
10 GOSUB *MOVP
20 M1=10
30 *MOVP
40 IF M1<>10 THEN M1=10
```

3)

```
10 IF M1>10 THEN
20 MOV P1
30 ELSE
40 MOV P2
```

4)

```
10 IF P1.X>255.1 AND P2.Y< 180.6 THEN
20 M1 = (M1 * M2) / 2
30 ENDIF
40 GOTO SUBPRG
```

5)

```
10 'THIS IS NASTY EXAM PROGRAM
20 'ROBOT ONLY MOVES TO P1
30 AND STILL SOMETHING FAILS...
40 MOV P1
```

6)

```
10 INPUT #2, CSSTR$
20 IF CMES$="START" THEN
30 INPUT #2, CSSTR$, CPER$, P1
40 PRINT #1, "MATCH PERCENT IS " + CPER$
50 MOV P1
```

(#1 is laptop and #2 is machine vision camera. Camera sends string: START,67,
(134.45,156.14,0,90,0,0)(0,0)