VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

FACULTY OF MECHANICAL ENGINEERING

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

# NÁVRH APLIKACE PRO VÝUKOVÝ MODEL MANIPULÁTORU SE TŘEMI STUPNI VOLNOSTI

DESIGN OF APPLICATION FOR EDUCATIONAL MODEL OF MANIPULATOR WITH THREE DEGREES OF FREEDOM

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

| | |
|---|---|
| AUTOR PRÁCE<br>AUTHOR | Bc. DANIEL YOUSSEF |
| VEDOUCÍ PRÁCE<br>SUPERVISOR | Ing. DAVID KLIMEŠ |

BRNO 2014

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav mechaniky těles, mechatroniky a biomechaniky
Akademický rok: 2013/2014

# ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Daniel Youssef

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Mechatronika (3906T001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

**Návrh aplikace pro výukový model manipulátoru se třemi stupni volnosti**

v anglickém jazyce:

**Design of application for educational model of manipulator with three degrees of freedom**

Stručná charakteristika problematiky úkolu:

Práce se bude zabývat návrhem algoritmů řízení pohybu výukového manipulátoru se třemi stupni volnosti. Diplomant bude mít k dispozici již realizovanou platformu výukového modelu (včetně veškeré elektroniky). Práce bude tedy spíše na softwarové úrovni. Cílem bude zakomponovat do výsledné aplikace především počítačové vidění s využitím vhodné kamery. Využito bude především prostředí Matlab-Simulink.

Cíle diplomové práce:

1. Zlepšení algoritmu inicializace jednotlivých pohonů na výukové platformě manipulátoru.
2. Rozšíření manipulátoru o jeden, nebo více stupňů volnosti-přidání servopohonu na koncové rameno. Návrh koncového efektoru pro psaní.
3. Návrh kinematického modelu (s využitím vhodné numerické metody). Otestování přesnosti pohybu.
4. Vytvoření uživatelského rozhraní pro servisní ovládání a zobrazování aktuálního stavu manipulátoru (regulační odchylky, polohy, rychlosti, atd.)
5. Sestavení ukázkové aplikace "piškvorky", ve které bude manipulátoru hrát "piškvorky" proti lidskému protějšku. Při realizaci této úlohy bude využito počítačové zpracování obrazu z použitého kamerového systému. Součástí výsledné aplikace bude vhodné uživatelské rozhraní.

Seznam odborné literatury:

- Grepl, R.: Kinematika a dynamika mechatronických systémů, Vydavatelství VUT v Brně
- Hlaváč, V., Šonka, M.: Počítačové vidění, Grada 1992

Vedoucí diplomové práce: Ing. David Klimeš

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2013/2014.

V Brně, dne 30.10.2013

L.S.

| | |
|---|---|
| prof. Ing. Jindřich Petruška, CSc. | prof. RNDr. Miroslav Doupovec, CSc., dr. h. c. |
| Ředitel ústavu | Děkan fakulty |

## Abstrakt

Tato práce popisuje další vývoj výukového sériového manipulátoru se třemi stupni volnosti. Práce se zabývá jednoduchou mechanickou úpravou manipulátoru, ale především pak softwarovou částí. Výsledkem je pak hra piškvorky, kdy manipulátor umožňuje hru proti lidskému protějšku.

První část práce je věnována zlepšení inicializačního procesu manipulátoru a následně pak i návrhem vhodné polohové regulace.

V další části je manipulátor rozšířen o jeden stupeň volnosti. Součástí je i návrh koncového efektoru vhodného pro psaní. Z takto upraveného manipulátoru je sestaven kinematický model vhodný pro real-time řízení.

Dalším krokem v práci je návrh samotné aplikace hry piškvorky. Je navržen vhodný hrací algoritmus, včetně detekce a rozpoznání znaků v hracím poli pomocí kamery. Následně je vše implementováno do real-time aplikace, kde komunikaci s uživatelem zajišťuje navržené uživatelské rozhraní.

## Abstract

This thesis develops a series manipulator with three degrees of freedom. It improves its mechanism and creates new software to carry out a manipulator vs human being Tic-tac-toe application.

The first part of the thesis focuses on improving initialization of the relative encoders and the positioning control of the manipulator.

The second part analyzes expanding the manipulator to 4dof. It Explains building a new kinematics model and also includes the mechanical design and manufacturing of a suitable end effecter for the Tic-tac-toe application.

Next part explains the Tic-tac-toe application. It analyzes the used hardware and software including the Tic-tac-toe algorithm and the detection algorithm used by the camera.

Last part of the thesis deals with building user interfaces in order to make operating the manipulator and playing Tic-tac-toe more user friendly.

## Acknowledgement

I would like to thank my family for their trust and support, my supervisor Eng. David Klimes for his guidance and help, and all colleagues from Mechlab for providing a pleasant working environment.

## Statutory declaration

I statutory declare, that I wrote this paper by myself with the usage of stated literature and under the supervision of my instructor.

Daniel Youssef, Brno, 2014

# List of Figures

# Contents

# 1 Introduction

Robots are the language of the future that has been formed for the past few decades. They are gradually becoming more and more important to industry, science and even our everyday life. Series manipulators are an important branch in the field of robotics that has been increasingly used in legged robots, industry automation such as welding cars and many other applications.

The above motivated a number of my colleagues (students and graduates of mechatronics) to create an open chain series manipulator with three degrees of freedom. Their goal was for the manipulaotr to serve as an educational model mainly used for practical demonstration of theories of dynamics and kinematics of robots, which is a part of the of mechatronics curriculum of the fourth year.

The manipulator was created in the Mechlab laboratory. The mechanism was designed and built by Eng.Tomas Ripel, whereas Eng.David Klimes designed and realized the electronics as a part of his master's thesis, which also included programming some applications such as making the manipulator follow the movement of a black object using a real time camera. Data transfer back and forth between a Pc, which is the brains of the manipulator and the electronics is carried out by an I/O card (Mf624) created by Humosoft and can be accessed using the Real Time Simulink library.

A few essential blocks that secured the manipulator's basic functions such as receiving encoders' values, initialization of the manipulator, inverse kinematics and sending PWM signals to the electronics were built in Simulink environment by both Eng.Klimes and Eng.Suransky, whose thesis focused on applying different control theories to the manipulator's drives and comparing the results. The previous blocks were combined in a Simulink library called Manipulator (RRR[1]).

Improving the manipulator's mechanics and software to realize more complex applications such as Tic-tac-toe forms the goal of this thesis. These applications aim to increase students' interest in robotics, and serve as a good demonstration tool for using a set of skills obtained through various courses along the studying of mechatronics such as kinematics and dynamics of robots, artificial intelligence, control theory and programming.

[1]RRR:refers to the fact that the manipulator consists of three rotational joints

# 2 Goals Description

The following includes a detailed explanation of the goals set for this thesis to reach:

- **Improvement of the initialization process of the encoders**

  The previously mentioned **Manipulator (RRR)** library includes a block that is responsible for the initialization process of the relative encoders, which is accomplished by passing over or hitting index switches placed in already defined positions [1]. By modifying the relative output value of the encoder to match the position of the index switch, the absolute position is obtained. The initialization process should ensure passing over these switches, which is essential for the manipulator. It should produce a movement that will dependably make the manipulator pass over or hit all index switches regardless of the starting position.

- **Adding one more degree of freedom to the manipulator**

  A series manipulator with three degrees of freedom is able to reach an $(x, y, z)$ coordinate within the working space boundaries, without the ability to control the rotation angles of the end effecter, which is essential in many applications such as writing, where the pen is required to be kept perpendicular to the writing board. Adding one more drive to the manipulator (one more degree of freedom) enables controlling the rotation of the end effecter around one global axis.

- **Choosing and implementing the algorithm of inverse kinematics**

  Regardless of the application desired to be accomplished, inverse kinematics is used. Desired positioning and rotation is converted by inverse kinematics to the corresponding joints' coordinates that are the inputs of the controllers. Inverse kinematics suffers from various singularities, and choosing the right method that minimizes them is really essential. After creating inverse kinematics and choosing the right controllers, the precision of the manipulator, which is the difference between the required position and the real one, can be tested. Various factors play a role in the lack of precision the manipulator may suffer from, and measuring how precise the manipulator can be is important to verify how good the system works as a whole.

- **Creating Tic-tac-toe application**

  The previous steps form the core to any desired application planned to be carried out by the manipulator. Playing Tic-tac-toe against a human being seems as practical application that will be attractive to students, and make them interact with the manipulator. It demonstrates the practical use of theoretical knowledge obtained through various courses, and increases students interest in the field of robotics.The application includes creating a Tic-tac-toe algorithm, drawing X and O symbols by the manipulator, programming a real time camera to detect the person's moves, and connecting the previous sub parts together to make the application work.

- **Creating GUIs (Graphical user interfaces)**

  Making programs more user friendly is really essential, and Matlab enables this by using the Graphical User Interface (GUI) environment to create a control panel of the desired program. GUI can communicate with Simulink models, and change their parameters by simple slider gains, push buttons and boxes. It can also show generated data in the model through graphs. The goal is to create two GUIs:

  – GUI that enables the user to play Tic-tac-toe against the manipulator, and displays the progress of the game taking place on the drawing board.
  – GUI that enables the user to command the manipulator to perform some basic tasks such as drawing a symbol with a changing speed and size besides displaying data like current consumed by each motor and controllers errors.

# 3 solution procedures

In addition to Improving the initialization process to be more dependalbe, expanding the manipulator by one more joint and solving the problem of inverse kinemaitcs accomplishes the foundations for designing and programming the Tic-tac-toe application. This approach is explained through the following sections.
Figure below shows the initial state of the manipulator at the begining of work on the thesis.

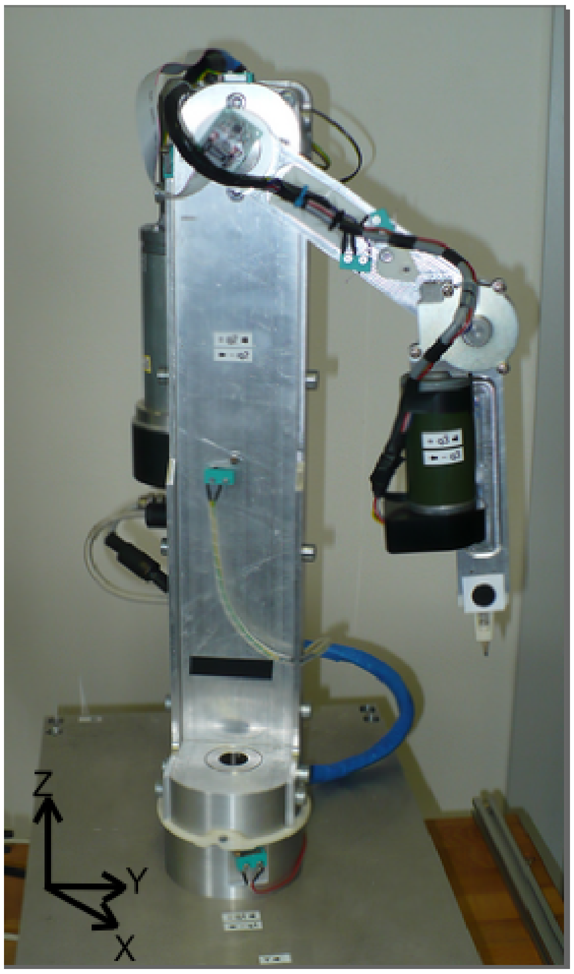


Fig. 3.1: 3dof series manipulator.

## 3.1 Initialization of the manipulator

After studying the Manipulator (RRR) library, it was obvious that the initialization blocks need some modifications. These blocks should ensure for the parts of the manipulator to pass over or hit the index switches in order to get absolute position from the used relative encoders attached to each of the three motors[1].

### 3.1.1 Previous initialization blocks and their disadvantages

To initialize the manipulator each of the previous blocks creates a movement of the corresponding drive in both directions to an already defined angle $\theta$ without any information about hitting the limit switches that defines the working boundaries of each drive. This movement is generated by the**signal generator** block used in Simulink. What was mentioned above implicates two problems:

- These blocks do not always ensure passing over the index switch, because it might be out of range (the angle between the index switch and the starting position is bigger than $\theta$).
- In case the angle between the starting position and the limit switch is smaller than $\theta$, the initialization block will command the manipulator to exceed the defined working space and causes the securing electronics [1] to take over, which is not desirable.

### 3.1.2 Using State Flow to create more robust initialization block

Using classical Simulink blocks to carry out the initialization was a bit complicated [1]. Therefore State Flow[1] seemed as a good alternative. The main idea was to solve both problems that the previous blocks had, which implies a non limited initialization range and knowledge about hitting the limit switches. Taking into consideration the applications required by the manipulator needs the initialization process to avoid a certain area, where the drawing board and its stand are positioned, see fig 3.22. The following provides a detailed explanation of the improved blocks:

#### Initialization block of the first drive

The first drive rotates the manipulator around the Z axis, see fig 3.1. This block besides initializing the drive it also enables stopping it. To avoid collision with the drawing board it is initialized first, while both drives second and third are kept at their starting positions. The figure below shows the corresponding State Flow chart with its inputs and outputs.

[1] "State Flow is an environment for modeling and simulating combinatorial and sequential decision logic based on state machines and flow charts" [**?**]

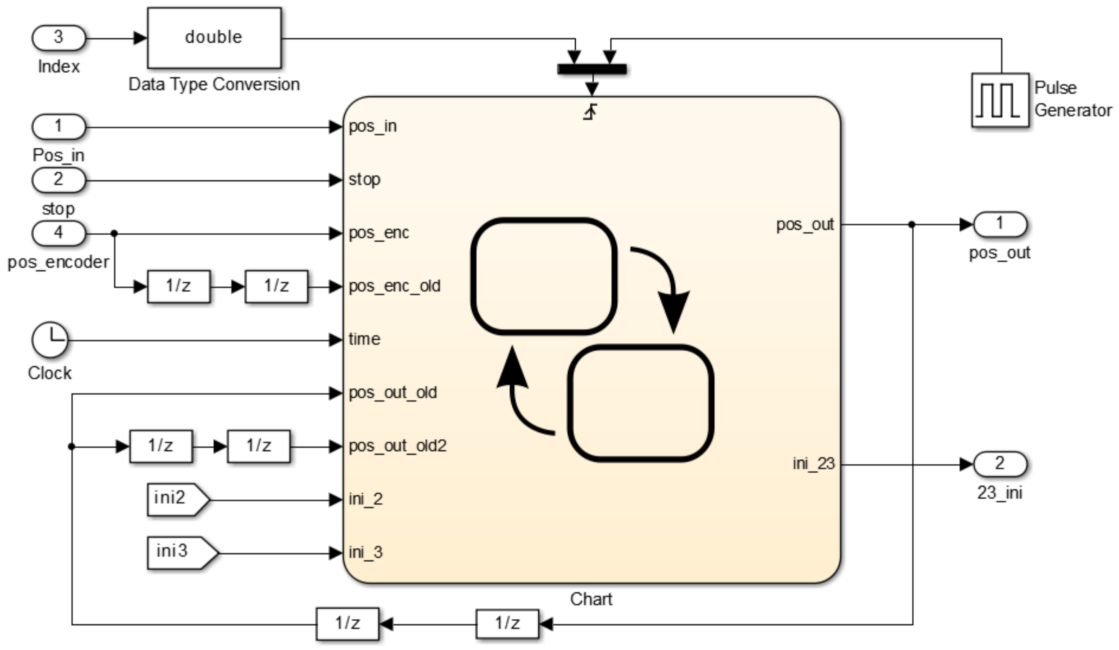


Fig. 3.2: Inputs and outputs of the initialization block of the first drive.

- **pos_in**: The joint's coordinate, which the state flow chart receives and should pass through after initialization is done (may be the output of inverse kinematics).
- **stop**: Enables to stop the first drive by applying a none zero value to this input.
- **index**: Signal received from the index switch after being processed in the **Encoder** block [1].
- **pos_Encoder**: position obtained by the encoder also after being processed.
- **ini2,ini3**: Signals indicating whether drives 2 and 3 are initialized.
- **ini_23**: Output signal that indicates initialization of the first drive.
- **pos_out**: Output position sent by the chart to the controller.

**Note**: Fig. 3.2 shows a pulse generator that runs the state flow diagram at a frequency that equals half the frequency, which the Simulink model runs at. Therefore getting a delay equal to the state flow diagram step requires two back to back delay blocks with a default delay time.

### Initialization block's structure

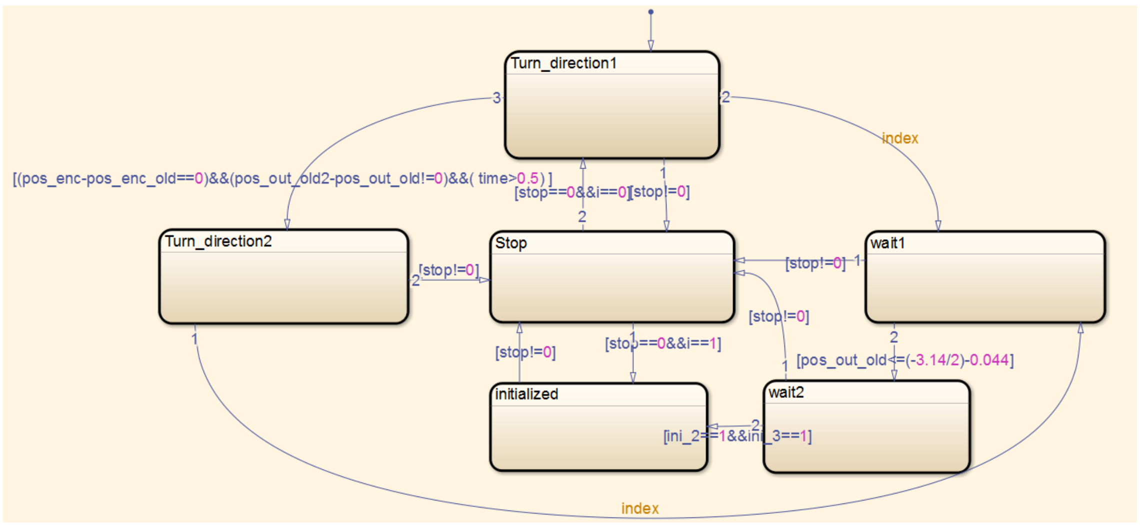


Fig. 3.3: State Flow chart of the initialization block of the second drive.

The main idea is to turn the drive in one direction till one of two situations occurs:

- Till it runs over the index switch, then the drive will be initialized
- Till it hits the limit switch, then the drive will be turned in the opposite direction till it runs over the index switch and it will be also initialized.

Turning the drive in one direction or the other is easily done by incrementing or decrementing the output position. Although the signal of the limit switch is not available (it is not fed to the I/O card), hitting the switch (stopping) can be figured out By comparing the output position of the encoder to the output position of the initialization block. If the output position of the encoder is not changing, while the output position of the initialization block still is, it indicates that the drive has hit its limit, and the program starts turning it in the opposite direction. When the drive runs over the limit switch it becomes initialized, and then it is turned to an already defined position suitable for drivers 2 and 3 to be initialized in. The first drive waits there till the initialization process of the remaining drives is complete. After that, a feedback signal is sent to the first initialization block indicating that the whole manipulator has been initialized, and the input signal can be passed.

### Initialization blocks of the second and third drives

These two blocks are really similar, therefore explaining one of them will be enough, so let it be the initialization block of the second drive. The inputs and outputs of the State Flow chart are similar to fig. 3.2 with one input missing, which is the pos_encoder. Also it receives a signal indicating whether the first drive is initialized and outputs a signal indicating its initialization state.

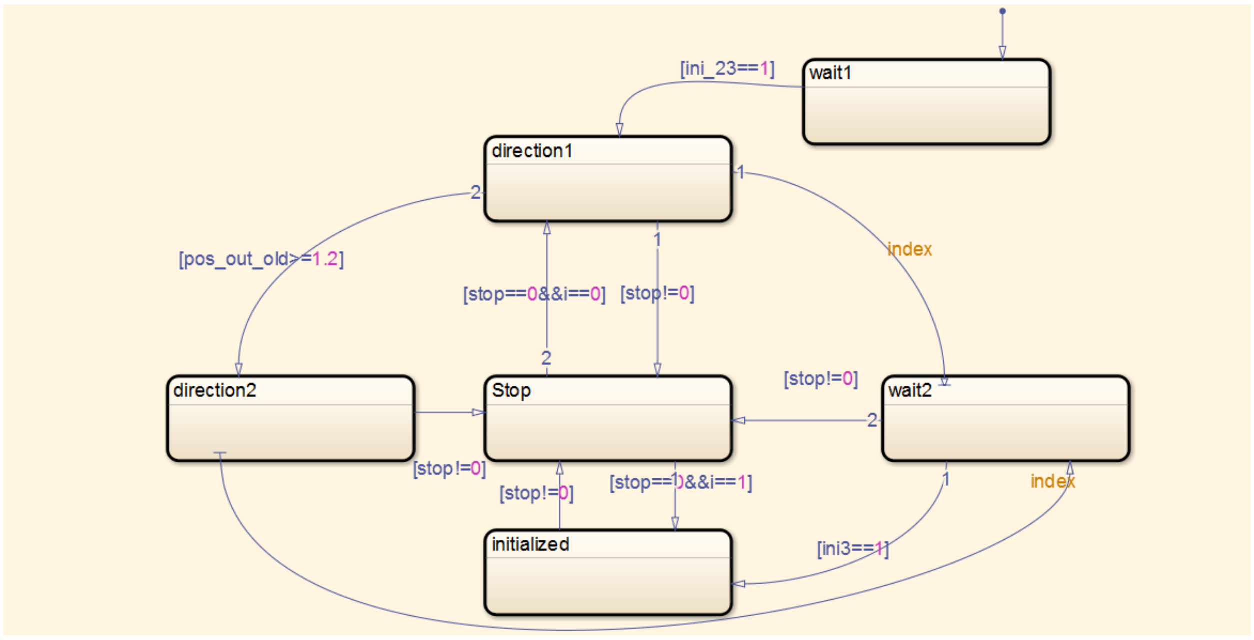


Fig. 3.4: State Flow chart of the initialization block of the second drive.

The second drive waits for the first drive to be initialized, and then it begins its initialization process. Starting position of the second drive is always near zero value because of the moment generated by the weight of the part attached to the motor. This means that it is enough to turn the drive by an already defined angle in one direction, and if it does not run over the index switch it starts turning in the opposite direction till it runs over it. This explains not using the output position of the second encoder, important for figuring out, whether the drive hits the limit switch. When the second drive is initialized, it waits till a signal indicating the initialization of the third drive is received, and then it starts passing the input position.

### Differences between initialization of the second and third drives

Index switch of the third drive is both an index and a limit switch, which means that the initialization block turns the drive in one direction till it hits the previous switch, and initialization is done. In order for the drive to be slow while colliding with the switch, the increment of the output starts big then it gradually decreases.

## 3.2 PID positioning controllers

The outputs of the initialization blocks represent the desired rotation angle of each drive. To position each drive according to the desired value, controllers with feedback loops are used.

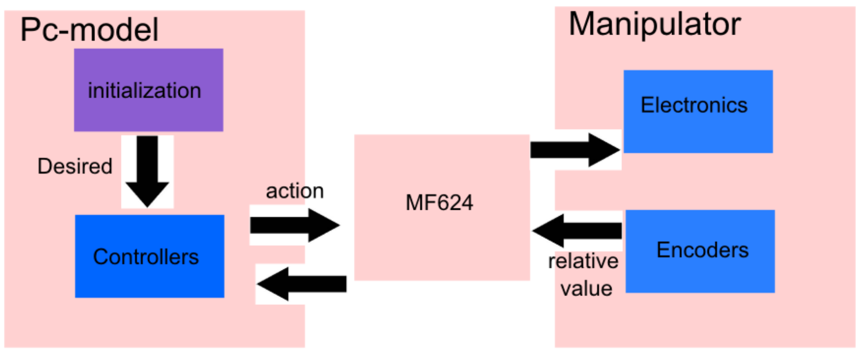


Fig. 3.5: Illustration of closed loop control.

Single loop control suffers from unacceptable static error, and the dynamic reaction is not swift enough. Series manipulator accumulates error due its design, and a small error in a joint coordinate can be reflected as a big deviation in the position of the end effecter. More accurate and fast controllers can be accomplished by using a cascade loop control as shown in the figure below.

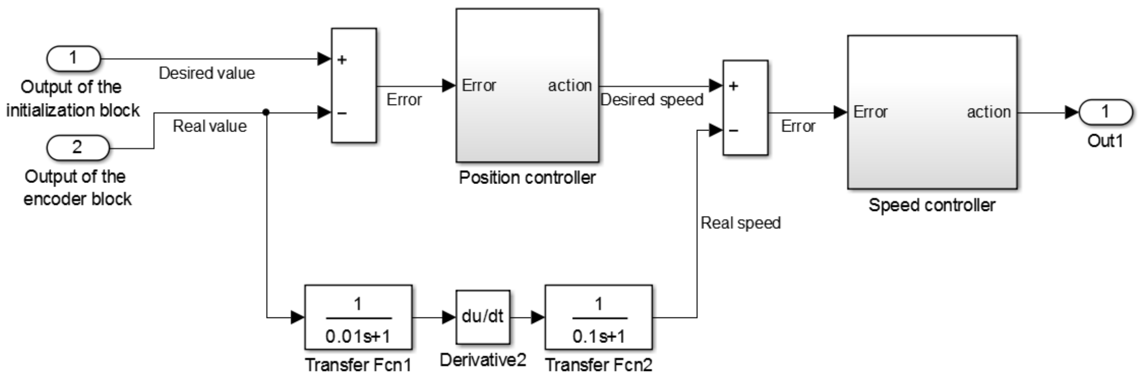


Fig. 3.6: Cascade control of drives positioning.

Instead of using only one PID [9] to control the joint's positioning, two controllers are used. The outer one controls position, and receives desired value by the initialization block, and real value processed by the encoder block. The inner PID controls a faster changing parameter which is the speed of the drive, its set point is controlled by the outer PID. Through using derivation and filtering the value of the actual speed is obtained from the actual value of the encoder without the need for an additional sensor. Controlling a faster changing parameter such as speed implies a faster dynamic reaction by the controller and a smaller static error.

**Note**: The encoders are directly attached to the motors, which implies that their values do not include the mechanical clearances of the gears. Therefore by comparing the desired value of the controller with the actual value of the encoder, the response of each controller can be precisely analyzed.

## 3.2.1 Step response of the first controller

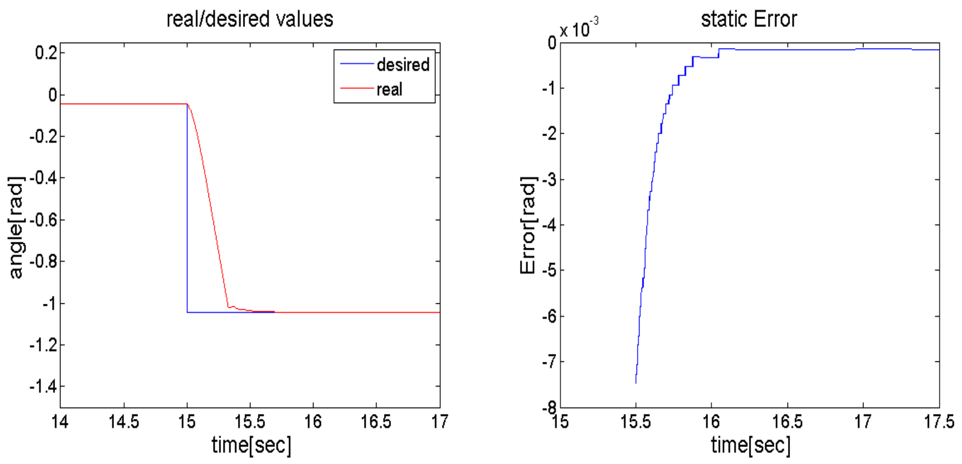


Fig. 3.7: Response of the first drive.

The graph in the left shows the response of the first controller to a desired value in the form of a step with an amplitude of one radian. Although the amplitude of the step is big the real value settles down in approximately one second, and in less than half of a second the error absolute value goes down to 0,007 radians, which indicates a fairly good dynamic response. The graph in the right shows the static error of the controller, which is really small (less than 0,0002 radians).

### 3.2.2 Step response of the second controller

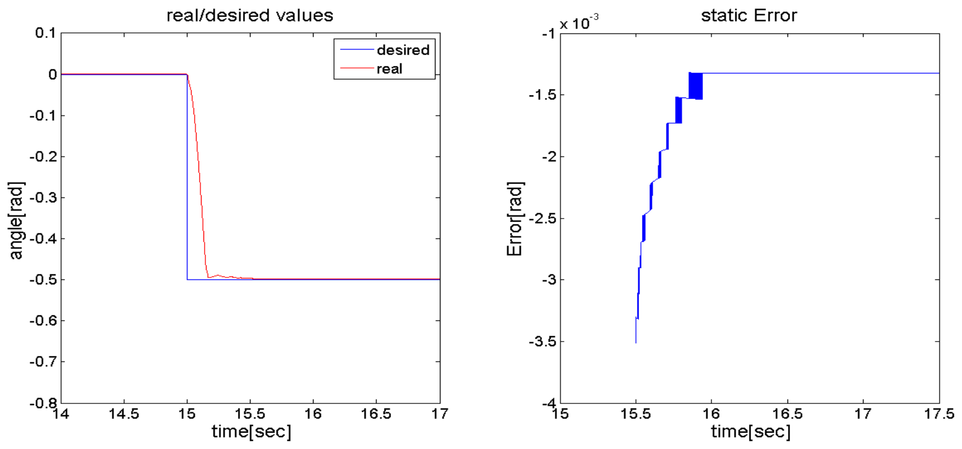


Fig. 3.8: Response of the second drive.

The static error of the second controller is bigger. Its absolute value is about 0,0015 radians, but it is still small enough. Dynamic response of the controller is similar to the previous response, and the real value settles down in approximately one second, and in less than half of a second the error goes down to 0,0035 radians.

### 3.2.3 Step response of the third controller

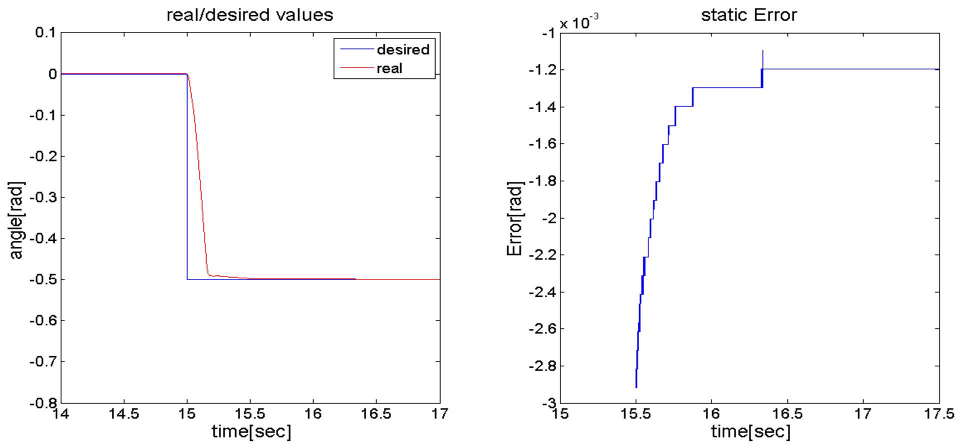


Fig. 3.9: Response of the third drive.

The graph in the left shows the response of the third controller to a desired value in the form of a step with amplitude of 0,5 radian. Although the amplitude of the step is big, the real value settles down in one second, and in less than half of a second the error goes down to 0,003 radians, which also indicates a fairly good dynamic response . The graph in the right shows the static error of the controller, which is really small (less than 0,0015 radians).

**Note**: The previous controllers will be used in applications, where the manipulator is expected to write. This means that the desired steps will be much smaller and the controllers will be fast enough to respond, though controllers with a slow dynamic response might cause deformations in written characters.

**Note**: Tuning the controllers has been done experimentally. Reached results can be improved, but they are sufficient for writing applications.

## 3.3 Adding one more degree of freedom to the manipulator

In series manipulators number of joint's n when less than 7 indicates number of Cartesian coordinates that can be controlled [2]. In writing and drawing applications, keeping the pen perpendicular to the board is essential. To control the rotation angle of the end effecter besides its (x,y,z) coordinates , one more joint should be added to the manipulator. This can be done by attaching a servo motor to the third arm of the manipulator, and controlling its rotation angle through the Mf624 card by sending a PWM with a changing duty cycle.

### 3.3.1 Mechanical design of the end effecter

To attach a pen to the servo motor a proper end effecter should be designed and realized.

#### Requirements set for the design of the end effecter

- The end effecter should ensure firm attachment to the servo motor.
- Should hold the pen tightly and ensure minimum clearance between the pen holder and the pen itself.
- Should allow the pen to be pushed upwards, and return it to its initial position after pressing is over, which is really essential when pressing against the board during writing.
- Should enable changing pens easily.

The design shown in fig. 3.10 consists of two parts attached by a tension spring, which allows the pen to be pushed and returned to its initial position. Mechanical clearance between the pen holder and the pen is only around 0,4 mm and the pen is held by a magnetic force. A magnet is glued to the top of the upper part of the design, and a small piece of iron is glued to the pen, which generates the needed magnetic force.

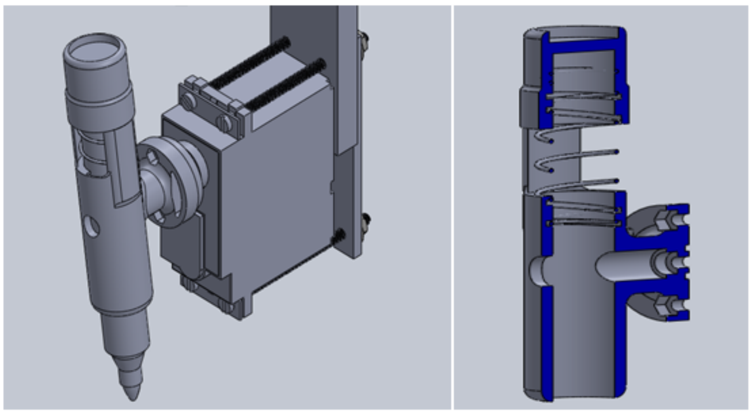


Fig. 3.10: Solid works design of the end effecter.

**Note**: Torque required to be generated by the servo in this application is really minimum, this is why it was chosen without calculating the maximum needed torque.

**Note**: Both previous parts were realized using a 3d printer with 0,2 mm tolerance. Therefore the mechanical clearance between the pen holder and the pen was set to 0,4 mm.

### 3.3.2 Calculating maximum deviation of the end effecter

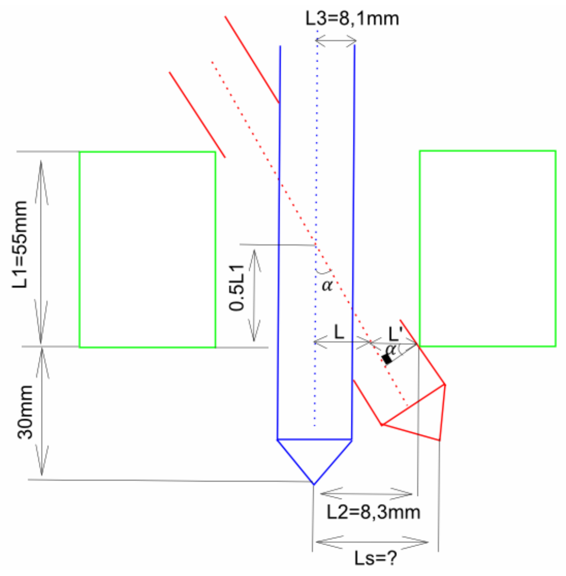


Fig. 3.11: Calculating maximum deviation of the end effecter.

Due to mechanical clearance the pen endures deviation that may cause an error in the positioning of the end effecter. It is really important to calculate the maximum error that could occur to decide how convenient this design is.

$$L = L_2 - L' \tag{3.1}$$

$$L' = \frac{L_3}{\cos\alpha} \tag{3.2}$$

$$\tan\alpha = \frac{L_2 - \frac{L_3}{\cos\alpha}}{\frac{L_1}{2}} \tag{3.3}$$

$$L_1.\sin\alpha = 2L_2.\cos\alpha - 2L_3 \tag{3.4}$$

$$L_1.\sin\alpha = 2L_2.\sqrt{1-\sin\alpha^2} - 2L_3 \tag{3.5}$$

$$(L_1.\sin\alpha - 2L_3)^2 = (2L_2.\sqrt{1-\sin^2\alpha})^2 \tag{3.6}$$

$$L_1^2.\sin^2\alpha + 4L_1.L_3.\sin alpha = 4L_2^2(1-\sin^2\alpha)^2 \tag{3.7}$$

$$(L_1^2 + 4L_3^2)\sin^2\alpha + 4L_1.L_3.\sin\alpha + 4(L_3^2 - L_2^2) = 0 \tag{3.8}$$

$\alpha$ is a small angle

$$\sin\alpha \approx \alpha \tag{3.9}$$

$$(L_1^2 + 4L_3^2)\alpha^2 + 4L_1.L_3.\alpha + 4(L_3^2 - L_2^2) = 0 \tag{3.10}$$

by solving equation 3.10

$$\alpha = 6,179.10^{-3}\ \text{rad} \tag{3.11}$$

$$\alpha \approx \sin\alpha \tag{3.12}$$

$$L_s = (\frac{L_1}{2} + 30)\sin\alpha = 0,35\ \text{mm} \tag{3.13}$$

$\alpha$: Maximum deviation angle.
$L_s$ : Maximum deviation of the end effector
**Note**: Maximum deviation of the end effecter is small and the previous design is acceptable.

## 3.4 Manipulator's kinematics:

Regardless of the desired application by the manipulator[2], inverse kinematics is substantial part of the designed program. Inverse kinematics transforms desired Cartesian coordinates of the end effecter that are much easier to work with to the corresponding joints' coordinates [2] needed as desired values for the controllers. For simple manipulators the problem of inverse kinematics can be solved analytically, but this solution is really hard to reach when more complex manipulators are analyzed. Therefore iterative algorithms that include forward kinematics in each step are used to solve the nonlinear equations describing the inverse kinematics problem.

### 3.4.1 Forward kinematics of the manipulator

The first step of building the algorithm of inverse kinematics is solving the forward kinematics problem of the given manipulator, which is the transformation of the joint's coordinates into the corresponding Cartesian position of the end effecter[2]. General forward kinematics [3] problem can be formaulated as follows:

$$\mathbf{X} = f(\mathbf{q}) \tag{3.14}$$

$\mathbf{X}$: The Cartesian coordinate vector.
$\mathbf{q}$: The joints' coordinate vector.
It can be solved for the transitional coordinates $(x, y, z)$ using the following equations:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = T_{n0}. \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.15}$$

$$T_{n0}(q_1, q_2......, q_n) = T_{10}(q_1).T_{21}(q_2).......T_{n-1,n}(q_n) \tag{3.16}$$

$T_{n0}$: Transformational matrix of the manipulator
$T_{10}, T_{21}, ......, T_{n,n-1}$: Transformational submatrices
$q_1, q_2, .....q_n$: Joints' coordinates.

[2]The manipulator analyzed by this thesis is an open chain series manipulator an RRRR type (consists of four rotational joints), see fig. 3.22

[3]Analyzed kinematics is related to open chain series manipulators

### Deriving Transformational sub-matrices of the manipulator

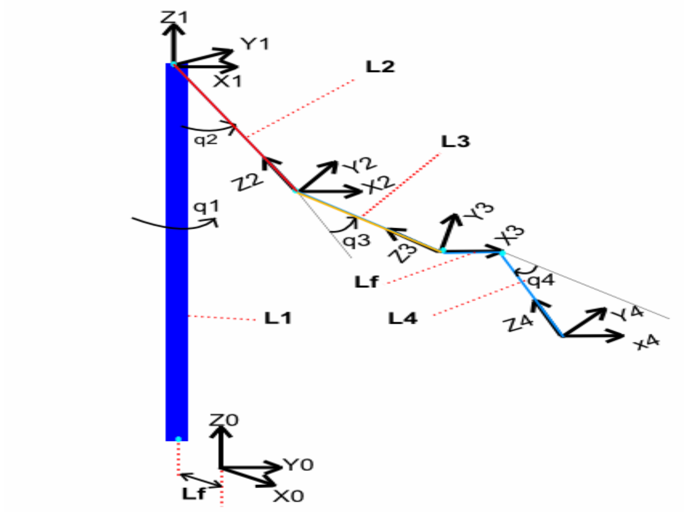


Fig. 3.12: The analyzed RRRR manipulator.

By analyzing figure 3.12, and using the general rotational matrices around Z and X axis [2] , the following transformational submatrices are obtained:

$$T_{43} = \begin{bmatrix} 1 & 0 & 0 & l_f \\ 0 & \cos q_4 & -\sin q_4 & L_4.\sin q_4 \\ 0 & \sin q_4 & \cos q_4 & -L_4.\cos q4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

$$T_{32} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos q_3 & -\sin q_3 & L_3.\sin q_3 \\ 0 & \sin q_3 & \cos q_3 & -L_3.\cos q3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

$$T_{32} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos q_2 & -\sin q_2 & L_2.\sin q_2 \\ 0 & \sin q_2 & \cos q_2 & -L_2.\cos q2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

$$T_{32} = \begin{bmatrix} \cos q_1 & -\sin q_1 & 0 & -l_f \\ \sin q_1 & \cos q_1 & 0 & 0 \\ 0 & 0 & 1 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.20}$$

$$T_{40} = T_{10}.T_{21}.T_{32}.T_{43} \tag{3.21}$$

Substituting $T_{40}$ into equation 3.15, forward kinematics of the manipulator for the transitional coordinates is solved, and the following results are obtained:

$$x = l_f.\cos q_1 - L_2.\sin q_1.\sin q_2 - L_4.\sin q_1.\sin(q_2+q_3+q_4) - L_3.\sin q_1 \sin(q_2+q_3) - l_f \tag{3.22}$$

$$y = l_f.\sin q_1 - L_2.\cos q_1.\sin q_2 - L_4.\cos q_1.\sin(q_2+q_3+q_4) - L_3.\cos q_1 \sin(q_2+q_3) \tag{3.23}$$

$$z = L_1 - L_2.\cos q_2 - L_4.cos q_4 + q_3 + q_2 - L_3.cos(q_2+q_3) \tag{3.24}$$

### Solving forward kinematics for the rotational coordinates

To control the angle at which the end effecter (pen) will reach the drawing board, we need to solve forward kinematics for the angle of rotation around the global X axis $\varphi$. Inverse kinematics input will be $(x, y, z)$ transitional coordinates besides $\varphi$ (rotation around the global X axis ). Two methods are used to solve the previous problem:

- Special(only relates to the analyzed manipulator): two possible sequences of rotation are considered to accomplish a desired movement by the manipulator:
  - First sequence: $q_2, q_3, q_4$ [4] change regardless to the changing order, so the end effector rotates around the global X axis. After that $q_1$ changes, and the manipulator rotates around the current $Z_c$ axis (Matches the global one). In the previous case obviously:

    $$\varphi = q_2 + q_3 + q_4 \tag{3.25}$$

  - Second sequence: $q_1$ changes first, the manipulator rotates around the global Z axis.
    After that $q_2, q_3, q_4$ change, and the end effecter rotates around the current $X_c$ axis (different form the global one).
    The second sequence can be proven to be equivelent to the first sequence using the following theory: Rotating a body around global axes in the $X, Y, Z$ order is equivelent to rotating it around the current axes in the $Z_c, Y_c, X_c$ order [2]. Therefore both previously analyzed sequences are equivelant. Hence in both cases,equation 3.25 is reached.
- General(using rotational matrix of the manipulator): equation 3.25 can be proven to be true using rotational matirx $R_{40}$, which is a submatix of the transforamtional matrix $T_{40}$:

[4] Analyzed manipulator and its rotational angles $q_1, q_2, q_3, q_4$ are shown in figure 3.12

$$R_{40} = T_{40}(1..3, 1..3) \tag{3.26}$$

Accoridng to the inverse problem of RPY [5][6] , $\varphi$ is obtained:

$$\varphi = \arctan r_{32}, r_{33} = \arctan \frac{\sin(q_2 + q_3 + q_4)}{\cos(q_2 + q_3 + q_4)} = q_2 + q_3 + q_4 \tag{3.27}$$

### 3.4.2 Algorithm of inverse kinematics

The problem of inverse kinematics can be formulated as follows [2]:

$$\mathbf{q} = f^{-1}(\mathbf{X}) \tag{3.28}$$

It represents a set of nonlinear equations, that can be solved by using iteration based on the Jacobean of the manipulator.

$$\Delta\mathbf{q} = \mathbf{J}^{-1}.\Delta\mathbf{X} \tag{3.29}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{q_1} & \cdots & \cdots & \frac{\partial f_1}{q_n} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \frac{\partial f_m}{q_1} & \cdots & \cdots & \frac{\partial f_m}{q_n} \end{bmatrix} \tag{3.30}$$

$\mathbf{J}$: Jacobean.
$(f_1 ... f_m)$: Cartesian coodinates expressed as functions of the joints' coodinates.
By substituting equations 3.22 3.23 3.24 3.25 into equation 3.30 the Jacobean of the manipulator is obtained,and then the inverse kinematics is solved using the following iterative process [2]:

$$\mathbf{x}_k = f(\mathbf{q}_k) \tag{3.31}$$

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \mathbf{J}^{-1}(\hat{\mathbf{x}} - \mathbf{x}_k) \tag{3.32}$$

$$\hat{\mathbf{e}}_k = (\hat{\mathbf{x}} - \mathbf{x}_k)\text{when } \hat{\mathbf{e}}_k < Error, \text{ position has been reached} \tag{3.33}$$

$\mathbf{x}_k$: End effecter Cartesian coordinates in the current step.
$\mathbf{q}_k$, $\mathbf{q}_{k+1}$: Joints' coordinates in the current step and the next one.
$\hat{\mathbf{x}}$: Desired Cartesian coordinates of the end effector.
$\hat{\mathbf{e}}$: Distance between reached position and desired one.
To use previous algorithm on the manipulaotr one more problem should be dealt with, which is sigularity.

$$det(\mathbf{J}) = 0 \tag{3.34}$$

[5]RPY(Roll,Pitch,Yaw): One form of Euler's angles used mostly in aviation

[6]Inverse porblem of RPY:refers to obtaining angles of rotation around X,Y,Z axes through an already known rotational matrix

Solving equation 3.34 shows that the Jocbean's determinant equals zero for various combinations of the joints' cooridnates. This means that the manipulaor suffers from singularity in some points within the working space boundaries, and the origin of the Global X,Y,Z system is one of them, which implies that the inverse of the Jacobean matirx can not be calculated at these points.
Therefore previous algorithm can not be implemented using usual Jacobean inverse or even pseudo inverse [7], which also is not possible to calculate for some joints' coordinates. Therefore Levenberg-Marquardt Damped Least Squares method has been used. This method expands the pseudo inverse by an identity matrix multiplied with a constant to maintain a non zero value near singularities.

$$\Delta\mathbf{q} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T + \lambda^2\mathbf{I}) \tag{3.35}$$

$\lambda$: Damping constant.
*"The damping constant depends on the details of the multibody, and the target positions, and must be chosen carefully to make the equation numerically stable. The damping constant should be large enough so that the solutions well-behave near singularities, but if it is chosen too large, then the convergence rate is too slow."* [3]
The Damping constant $\lambda$ was experimentally set to 0.001 , which ensured proper calculation near singularties and a fast convergence of the algorithm.
The algorithm of inverse kinematics was implemented into Matlab Simulink environmet using an embedded matlab function. The Simulink block is shown in the figure below [8].

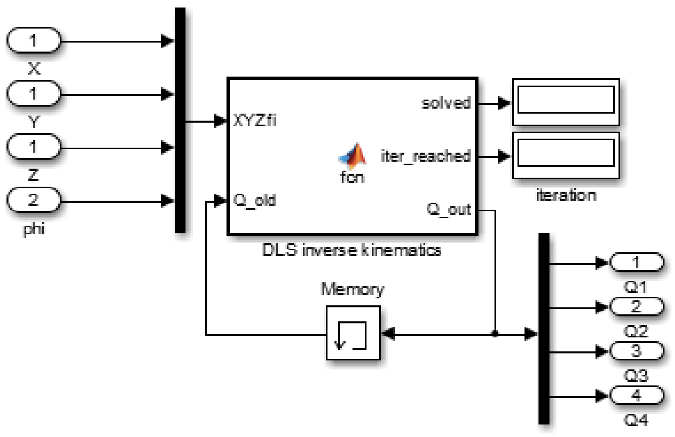


Fig. 3.13: Simulink block for inverse kinematics

[7]Moore–Penrose pseudo inverse: it can be used instead of the usual Jacobean inverse enables solution for rectangular matrices and redundant manipulators: $J^{\dagger} = J^T(JJ^T)^{-1}$

[8]Source code of the inverse kinematics block is in the attachments.

## 3.5 Measuring manipulator's precision

The main factors that affect the manipulator's precision are:

- Controllers: The used controllers have a static and a dynamic error, see section 3.2.
- Inverse kinematics: The equations of inverse kinematics are solved through iteration which implies a small error that the algorithm allows (1e-5). Also the lengths of the manipulator's parts that are used in the algorithm of inverse kinematics can suffer from measurement errors.
- Mechanical clearance: Gears Mounted on the shaft of the Dc motors and gears used in the servo motor suffer from mechanical clearance. Mechanical clearance also exists in the end effector 3.3.2.

To study the effects of the previous factors on the precision of the manipulator, a triangulation position sensor with high precision has been used. This sensor has a precision of $10^{-6}m$ and a range of 10 cm that starts at 5 cm away form the sensor and ends at the 15 cm mark. The manipulator has been commanded to move along one global axis with the sensor detecting the change in position of the end effector. The data collected by the sensor have been compared to the desired value, and the value calculated by forward kinematics applied on the angles measured by the encoders. This comparison enables specifying, how much each of the previously mentioned factors affects the manipulator's precision.

- Comparing the value calculated by forward kinematics with the sensor value shows error in position caused by mechanical clearances and forward kinematics, which is used to calculate end's effecter position that correponds to the encoders' measured angles.
- Comparing Desired value and the value calculated by forward kinematics shows error in position caused by controllers, inverse kinematics and forward kinematics. Blocks that seperate the previous two values.
- Comparing desired value with sensor value shows overall position error.

Fig.3.14 shows the Simulink model used to make the previous comparisons. A desired value that has the shape of a stair is applied. The triangulation sensor sends its signal to the Simulink mode through the I/O card, whereas the outputs of the encoder blocks are connected to the forward kinematics block to calculate the position that corresponds to the values of their rotation angles.

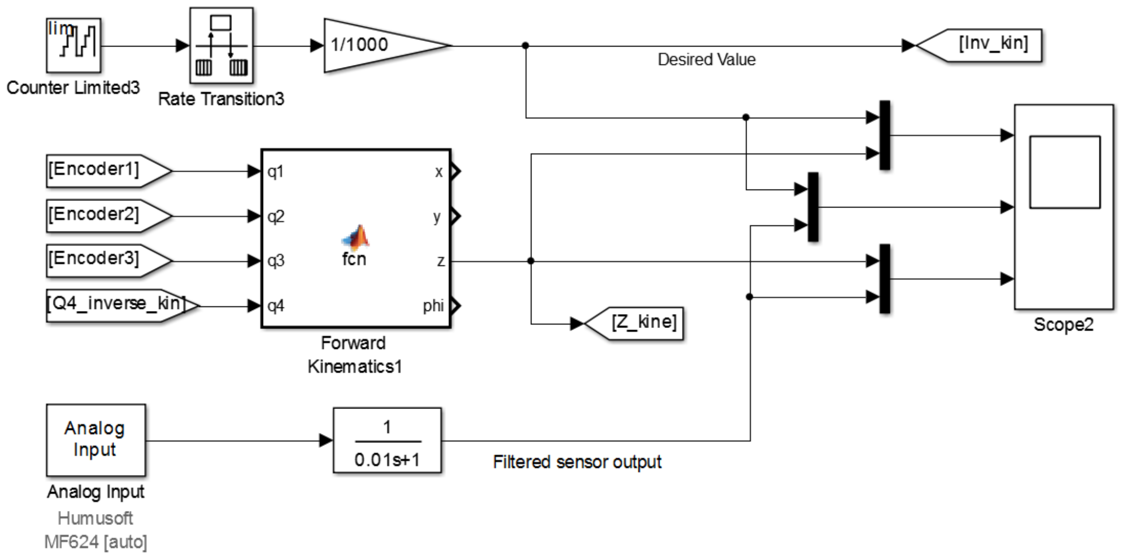


Fig. 3.14: Simulink model for measuring precision.

### 3.5.1 Measuring precision along the global Z axis

While moving along the Z axis the previously explained comparisons are made.

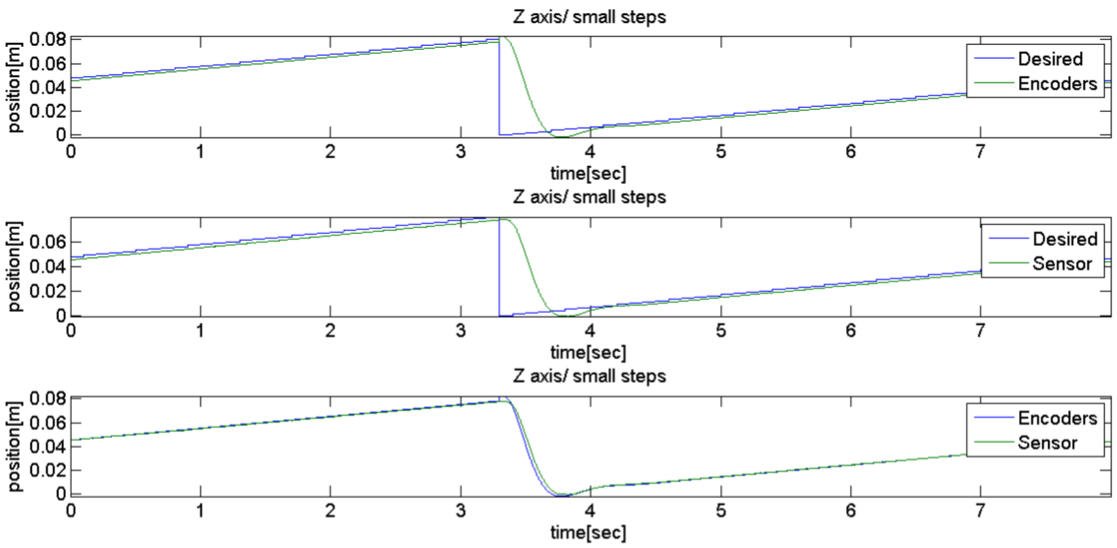


Fig. 3.15: Measuring precision along the Z axis/small steps.

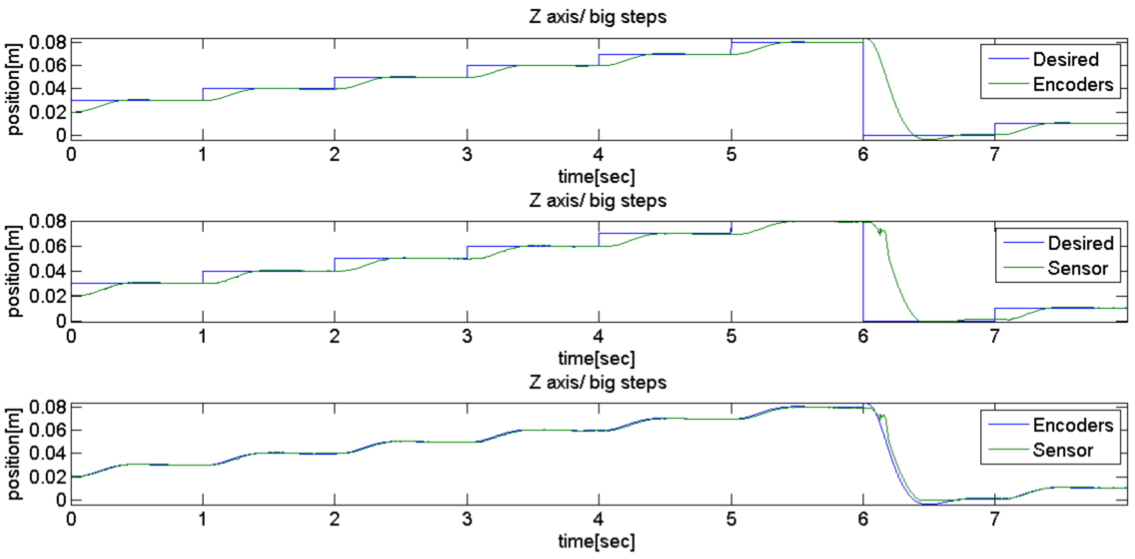


Fig. 3.16: Measuring precision along the Z axis/big steps.

Applying two types of desired value:

- Stairs with small step figure 3.15.
- Stairs with big step figure 3.16.

Shows:

- Graph of the value calculated by forward kinematics, and the graph of sensor value are almost identical in both cases, which implies small position error due to the algorithm of forward kinematics and the mechanical clearances. Factors that cause the difference between both values.
- Comparing the graph of the desired value with big steps to the graph of the value calculated by forward kinematics shows a big dynamic error and a really small static error. The small static error implies a small error caused by inverse and forward kinematics, on the other hand the big dynamic error implies the responsibility of the controller for the error , see section 3.2 . The previous conclusion explains the offset between the graph of desired value with small steps on one hand and the graphs of the sensor value and the value calculated by forward kinematics on the other, which is due to the dynamic error of the controller. Note that previous offset does not occur in figure 3.16
- Comparing static error between the graph of the desired value with big steps figure 3.16 and the graph of calculated value by forward kinematics to the static error between the graph of the desired value with big steps and the

graph of the sensor value shows that the first static error is smaller, which reflects the mechanical clearances in the gears and the end effecter.

### 3.5.2 Measuring precision along the global X axis

Positioning of the manipulator parts while moving along the X axis differs form thier positioning while moving along the Z axis, which implies differences in the gears' mechanical clearance and the response of the controllers. Hence, the previously explained comparisons are also carried out while moving along the X axis.

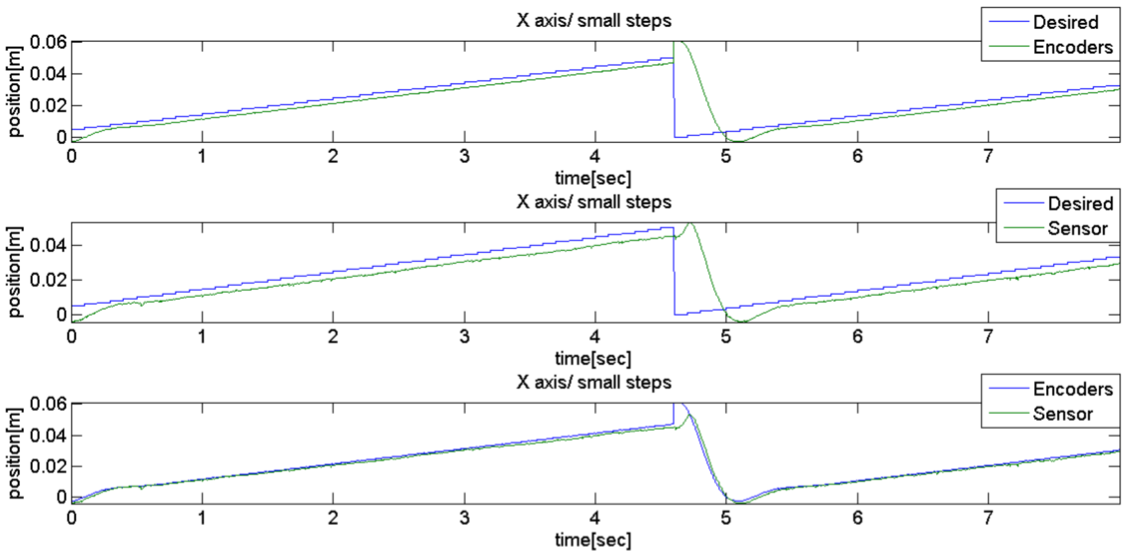


Fig. 3.17: Measuring precision along the X axis/small steps.

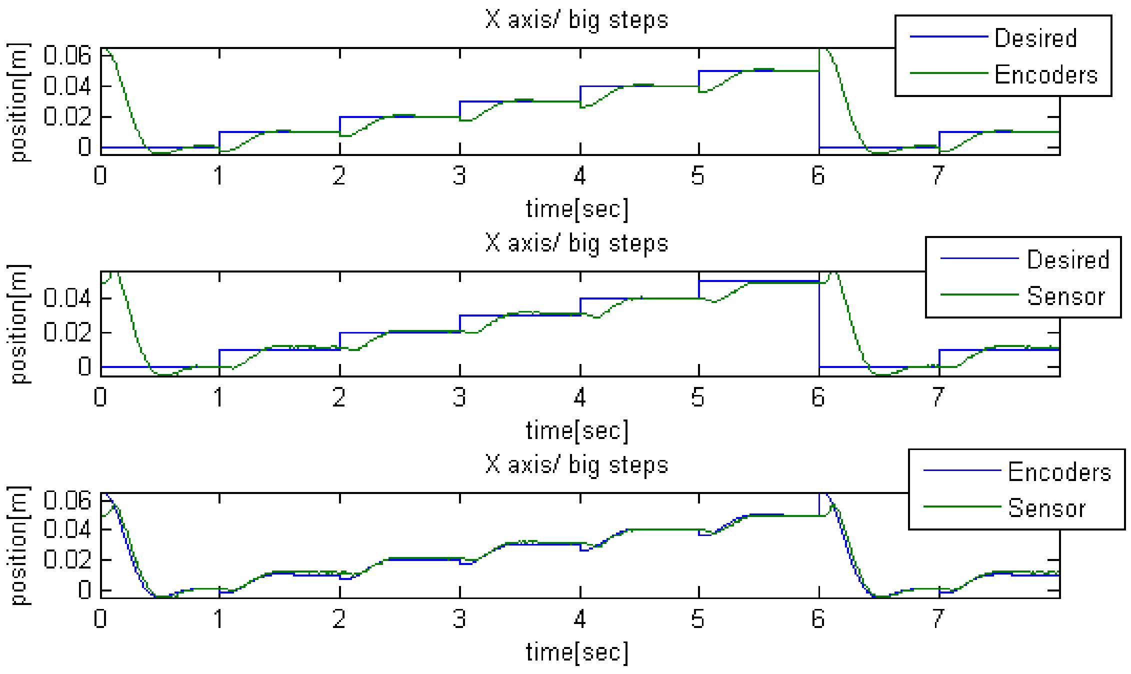


Fig. 3.18: Measuring precision along the X axis/big steps.

Figures 3.17 3.18 shows almost the same resutls reached along the Z aixs, but with some differences:

- Bigger dynamic error obvious in fig 3.17 due to the different positioning of the parts of the manipulaor during the movement along the x axis, which might cause a different response by the controllers .
- Static error is still really small, which implies the precision of forward and inverse kinematics.
- Also the difference in the positioning of the manipulator's parts during the movements along the X axis causes a change in the mechanical clearances of the gears, which explains the slightly bigger difference between the graph of the value calculated by forward kinematics and the graph of the sensor value compared to fig 3.15 3.16.

Note: Movement along the Y axis is identical to the movement along the X axis. The parts of the manipulator move in the same way, which implies reaching the same results.

### 3.5.3 Visual demonstration of the manipulator's precision

By using the manipulator to draw on a millimeter paper a more clear idea about its precision is formed.

#### Drawing two parallel lines

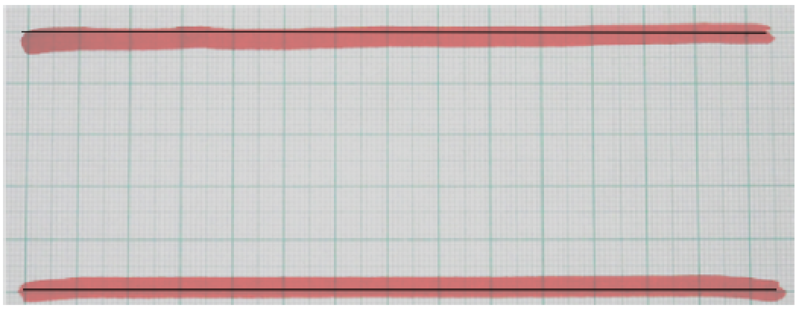


Fig. 3.19: two parallel lines drawn by the manipulator

Figure 3.19 compares two red lines drawn by the manipulator to the desired black lines. The lines are desired to be parallel to the X axis see fig 3.1, which is in turn parallel to the horizontal lines of the millimeter paper. The offset between both lines is desired to be 5 cm. Note that the previous requirements are fairly well fulfilled. Drawn lines and compared ones are almost identical.

#### Drawing a sine wave

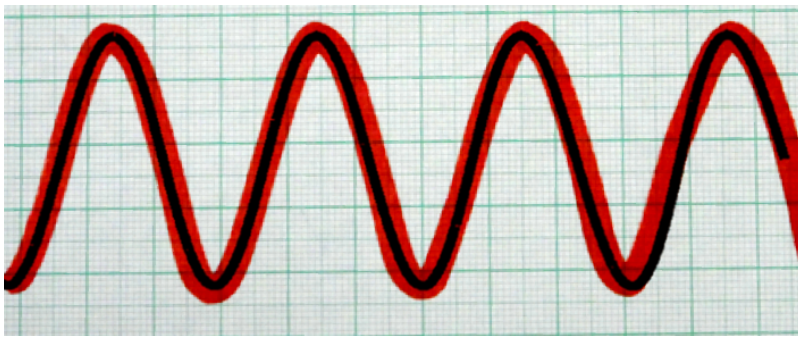


Fig. 3.20: A sine wave drawn by the manipulator

Figure above compares a red sine wave drawn by the manipulator to a desired black sine wave . The sine wave is desired to have an amplitude of 2 cm, and its average value parallel to the horizontal lines of the millimeter paper. Note that the previous requirements are fairly well fulfilled. Desired and drawn sine waves are really similar.

## 3.6 Programming the manipulator to play Tic-tac-toe

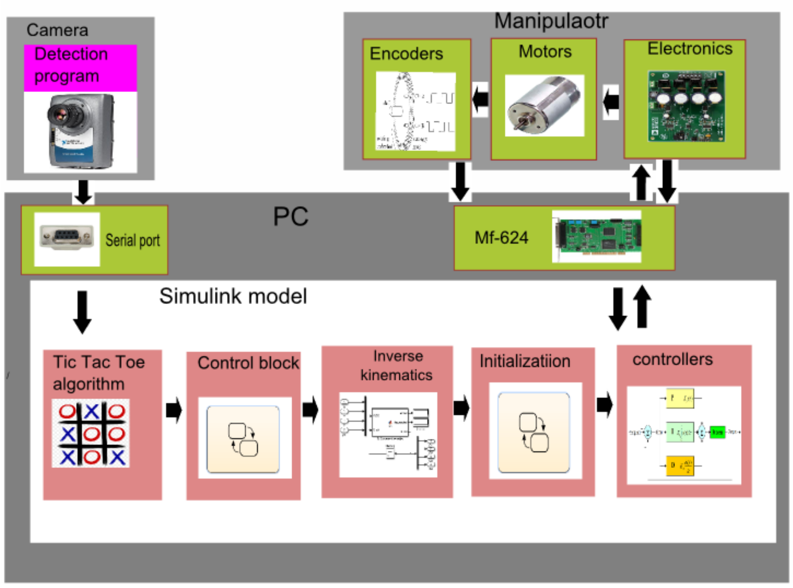


Fig. 3.21: Tic-tac-toe application diagram.

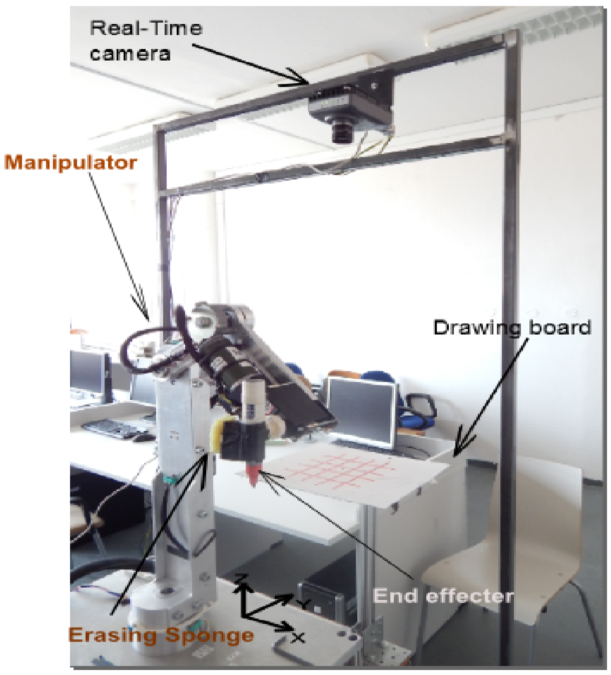


Fig. 3.22: Hardware components for the Tic-tac-toe application.

Figure 3.21 illustrates a block diagram of the designed solution for the Tic-tac-toe application, whereas figure 3.22 shows hardware components used to realize the previous solution. The application's goal is to enable the manipulator to play Tic-tac-toe against a human being, and it can be described as follows:

- The application is palying Tic-tac-toe person vs manipulator on a white erasable board showed in fig 3.22.
- The manipulator draws the playing field, which consits of five raws and five columns.
- The person chooses a playing symbol (X or O) and sets the manipulator to have the opposite one.
- When it is the person's turn to play, he should draw a symbol in one of the empty fields on the board.
- When it is the manipulator's turn to paly, the person should press a button in the model to allow the manipulator to perform its move.
- When the game is over, the manipulator erases the playing field.

In order to achieve the previous game requirments The following software and hardware components are needed:

- Real time camera: Programmed to detect the opponent's moves made on the playing board[9]. It sends the corresponding data to the Pc through serial port, see figure 3.22 3.21.
- The manipulator: See fig 3.22.
- PC: Equipped with an I/O card that enables communication with the manipulator's electronics and encoders. The Pc is the brains of the application. It sends through the I/O card driving signals to the electronics that in turns drives the motors on the manipulator, whereas the encoders provide necessary feedback for the program [10] running on the pc. The program consists of the following blocks:
  - Tic-tac-toe algorithm: By receiving data from the camera, it decides which move is best for the manipulator to play.
  - Control block: Generates Cartesian coordinates , responsible for drawing symbols, drawing the playing field and erasing. It receives the coordinates of the desired move and generates the corresponding Cartesian coordinates .

[9]The NI 1742 camera is used for this application [6]

[10]Matlab Simulink enviroment 2012b is used for programming. It enables accessing the Mf624 I/O card by using the Humosoft Real Time Windows Target toolbox

- Inverse kinematics: Converts Cartesian cooridnates to the corresponding joints' coordinates, for more information see section 3.4.2.
- Initialization blocks: Responsible for the initalization process, necessary for the relative encoders, for more information see section 3.1.
- Controllers: Used with the manipulator in a closed loop to control its joints' coordinates values, for more information see section 3.2.

## 3.6.1 Control block

It generates Cartesian coordinates that correspond to each desired sub-application. In order to enable switching between these sub-applications easily, State Flow was used for realizing this block.

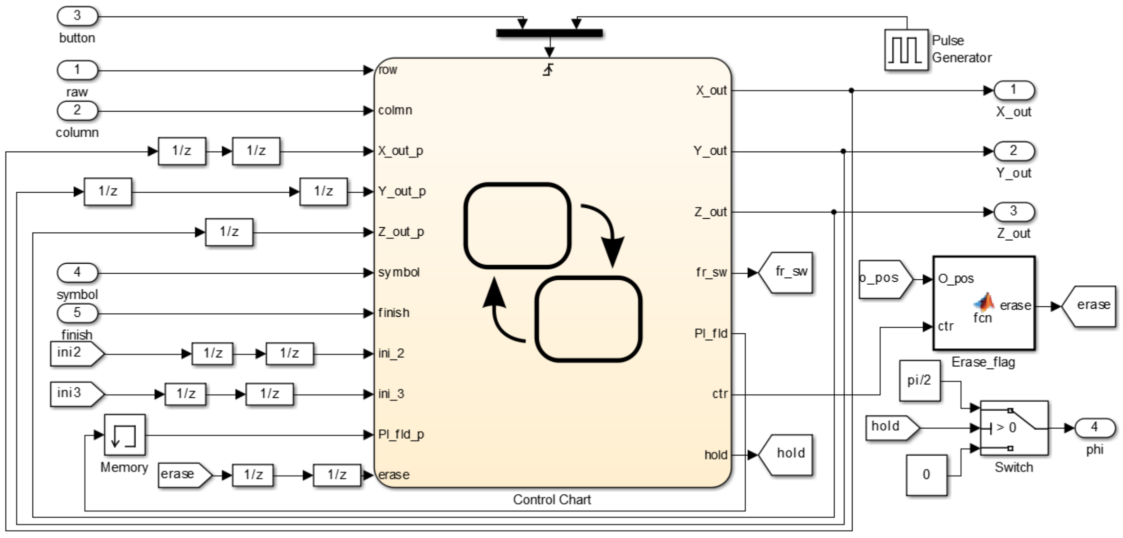


Fig. 3.23: Control block

The inputs and outputs of the control block shown in figure 3.23 are:

- **row,column**: They define the position of the desired move.
- **button**: A State Flow event, on its rising or falling edge, the manipulator starts drawing the chosen symbol in an already defined position.
- **symbol**: Sets the symbol to be drawn, 1 for (X) and 0 for(O).
- **finish**: Commands the manipulator to fininsh the application. 0 for normal mode, 1 for finishing.
- **X_out,Y_out,Z_out,phi**: The generated Cartesian coordinates.

- **X_out_p,Y_out_p,Z_out_p**: The generated Cartesian coordinates in the previous step of the real time process.
- **erase**: Erasing flag, tells when to start the erasing process.
- **Pl_fl_d**: Playing field flag, shows whether playing field has been drawn or not.
- **ctr**: Counter for the number of drawn symbols by the manipulator.

The following figure provides an insight into the Control block's State Flow chart, the main sub-charts it consists from, and how switching between them is done.

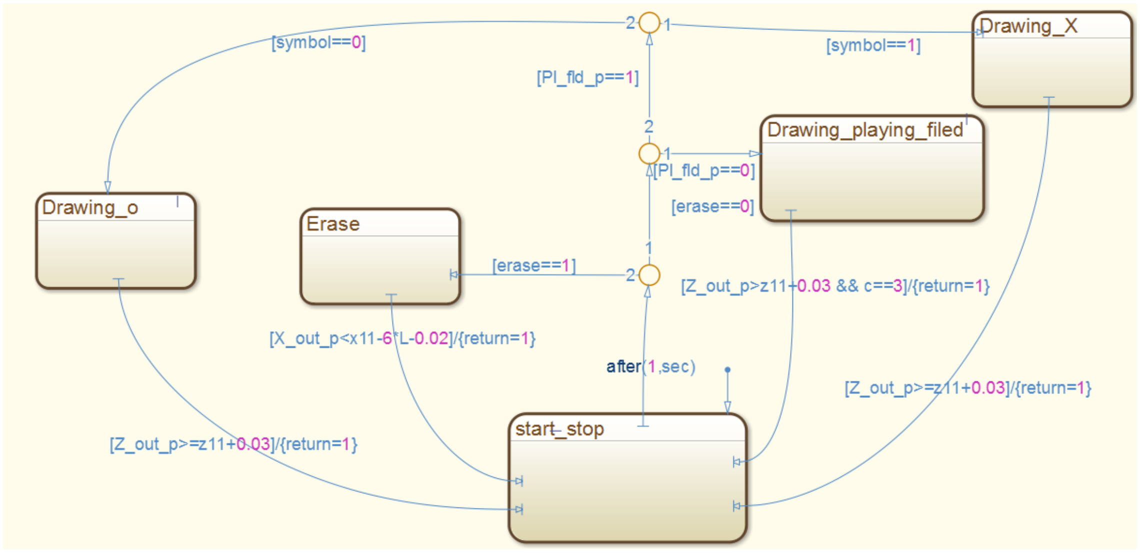


Fig. 3.24: The Control block's State Flow chart

Step by step the previous sub-charts will be explained:

- **start_stop sub-chart**

  The **control block's** chart shown in figure 3.24 starts executing at the **start_stop** sub-chart, which waits till initialization [11] is done, and then moves the manipulator to a position above the playing board, avoiding collision with the board during the process. It is also reponsible for finishing the application and returning the manipulator to its starting position, when the input **finish** is set to one.

[11]for more information about the initialization process see section 3.1

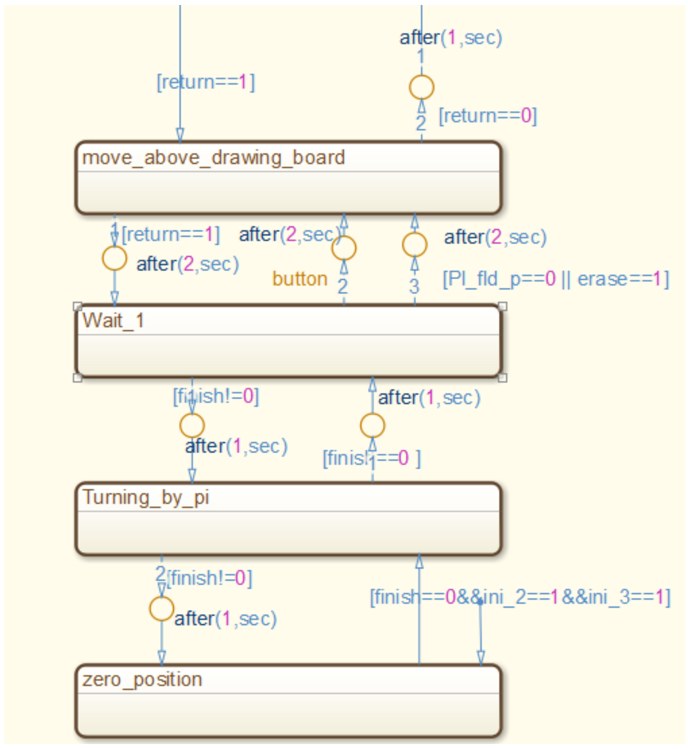


Fig. 3.25: start_stop sub-chart

The manipulator stays at the **wiat_1** block, which generates the Cartesian coordinates corresponding to a position near the drawing board, this allows the camera to capture the board without the manipulator blocking its vision. The manipulator waits in the previous position for the user's command, which can be applied by the event **button**. After drawing or erasing is done the **Control block** chart returns to **start_stop** sub-chart, which in turn returns to the **wait_1** block,see figures 3.24 3.25.

**wait_1** block generates movement by applying a desired position in the form of a step. In order to slow down the movement of the manipulator to the desired position, the cut off frequency of the filters[12] that process the desired input of the controllers is decreased. This is carried out by the **fr_sw** output of the control chart, see figure 3.23.
**Note**: The user only commands the manipulator to move from waiting position to draw a symbol, but when erasing flag or drawing a playing field flag are on, the manipulator moves to perform the given task without waiting for the user.

[12]Filters are used to provide safe operating of the manipulator.

- **Drawing_playing_field sub-chart**

  When the **Pl_fld_p** flag (indicates whether the field has been drawn or not) is off the state flow chart moves from the **wait_1** block to the Drawing_playing_field sub-chart,see figures 3.24 3.23. The previous sub-chart generates Cartesian coordinates for drawing the playing field of the Tic-tac-toe game. It is a 5 by 5 game so the manipulator draws 4 lines parallel to the global X axis and four lines parallel to the global Y axis, each of them is 15 cm long with an offset of 3 cm between every two neighboring lines parallel to he same axis. To draw one of the previous lines one coordinate stays constant and the other is increased slowly with each time step of the real time model. The increment is done by using the (**X_out** or **Y_out**) outputs as delayed inputs (**X_out_p** or **Y_out_p**) of the Control chart, see figure 3.23, and the following equation is used:

$$\text{X\_out} = \text{X\_out\_p} + \text{inc (same fo the Y coordinate)} \tag{3.36}$$

  *inc*: Used increment or decrement.
  After drawing each line, the pen is lifted off the paper. The constant coordinate value is changed, and the pen is lowered back to the paper. Lifting and lowering the pen is done by incrementing or decrementing the Z coordinate, which is carried out in the same way as with the X,Y coordinates. When drawing the playing field is done the playing field flag is set to one, and the State Flow chart moves back to the **wait_1** block in the **start_stop** sub-chart, and waits there for the user's command,see figures 3.24,3.25.
  **Note**:The complete sub-chart is in the attachments.

- **Drawing_X sub-chart**

  By changing the input of the event button the State Flow chart moves form the **wait_1** block to the **Drawing_X** sub-chart, which generates Cartesian coordinates for drawing an X in a field that corresponds to the generated coordinates of the Tic-tac-toe algorithm block. Every X symbol consists of two intersecting lines wiht a slope of $45^o$ and $135^o$. Both lines are drawn by incrementing X coordinate, and calculating the Y coordinate using the line's equation. The drawn lines are offseted according to the field's coordinates to be drawn in the desired position.

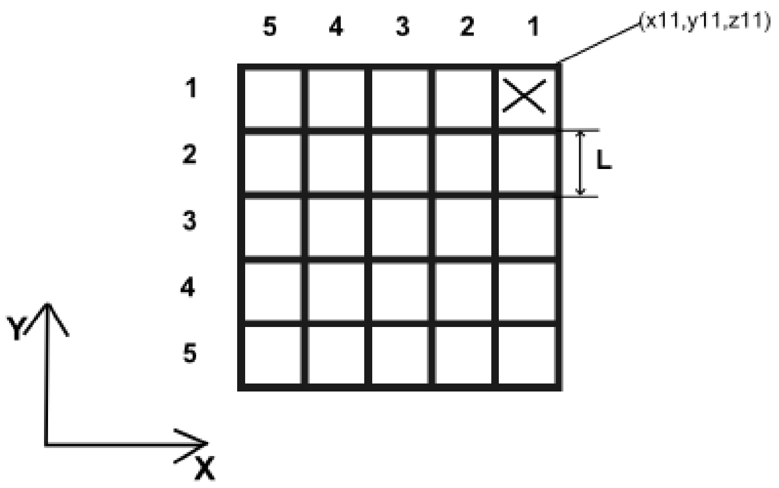


Fig. 3.26: Drawing an X

To draw an X according to the coordinates generated by the Tic-tac-toe algorithm in the playing field showed above, the following steps are implemented into the **Drawing_X** sub-chart:

– Calculating the coordinates of the begining point for drawing the first line using the following equations:

$$x\_ini = x11 - (column - 1)L - \frac{L}{5}; \qquad (3.37)$$

$$y\_ini = y11 - (row - 1)L - \frac{L}{5}; \qquad (3.38)$$

**x_ini,y_ini**: Coordinates of the starting point.
**x11,y11**: Cooridnates of the upper right corner of the playing field, see figure 3.26
**row,column**:Coordinates generated by the Tic-tac-toe algorithm.
**L**: Length of one field

– Drawing the first line using the following equations in every time step of the real time process:

$$h1 = y\_ini - x\_ini; \qquad (3.39)$$

$$X\_out = X\_out\_p - inc; \qquad (3.40)$$

$$Y\_out = X\_out\_p + h1 \qquad (3.41)$$

$h1$: Point of intersection of the first line with the Y axis.
$X\_out, X\_out\_p, Y\_out, Y\_out\_p$: Inputs/outputs of the Control chart, see figure 3.23.
$inc$: An already set increment.

– drawing the second line, which starts at a point that has the same Y coordinate as the ending point of the first line, but its x coordinate is x_ini.

$$h2 = Y\_end + x\_ini \tag{3.42}$$

$$X\_out = X\_out\_p + inc \tag{3.43}$$

$$Y\_out = -X\_out\_p + h2; \tag{3.44}$$

$h2$:Point of intersection of the second line with the Y axis.
$Y\_end$: The Y coordinate of the end point of the first line.

– While drawing an X the pen is needed to be lowered and lifited off the paper, which is done by incrementing or decrementing the Z coordinate.

When drawing an X is done, the state flow chart moves back to the **wait_1** block in the **start_stop** sub-chart, and waits there for the user's command,see figures 3.24,3.25.

- **Drawing_O sub-chart**

This sub-chart generates Cartesian coordinates that correspond to drawing a circle in a field defined by the Tic-tac-toe algorithm's output. By changing the input of the event button the state flow chart moves from the **wait_1** block to the **Drawing_O** sub-chart, see figure 3.24 3.25. Drawing a circle is devided into four parts, and each part represents a bit more than a quarter of a circle. Deviding the drawing into the previous parts has two reasons:

– To ensure closing the circle (connecting its beginning with its end)
– In each part one coordinate is incremented or decreased, while the other is calculated using a single equation derived from the equation defining the circle [13], which would not be possible if the circle was drawn as a whole.

The circle's center is offseted according to the coordinates generated by the Tic-tac-toe algorithm in order for it to be drawn in the desired field. The **Drawing_O** sub-chart consits of the following steps:

[13]The circle equation is: $(X\_out - x0)^2 + (Y\_out - y0)^2 = R^2$

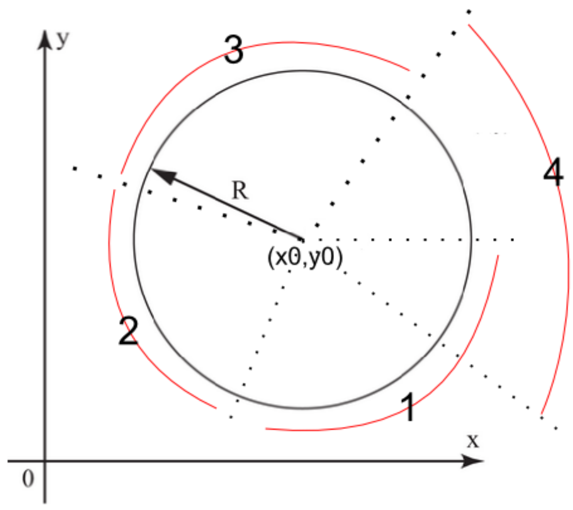


Fig. 3.27: Steps of drawing a circle

– Calcualting coordinates of the circle's center and its radius:

$$x0 = x11 - (column - 1)L - \frac{L}{2} \tag{3.45}$$

$$y0 = y11 - (row - 1)L - \frac{L}{2}; \tag{3.46}$$

$$R = \frac{L}{4}; \tag{3.47}$$

$x0, y0$: Coordinates of the circle's center. $R$: Circle's radius.

– Drawing the first part of the circle, see figure 3.27 :

$$X\_out = X\_out\_p - inc \tag{3.48}$$

$$Y\_out = -\sqrt{R^2 - (X\_out\_p - inc - x0)^2} + y0; \tag{3.49}$$

These equations are used in each time step of the real time process, till the first part is drawn.

– Drawing the second part, see fig 3.27:

$$Y\_out = Y\_out\_p + inc \tag{3.50}$$

$$X\_out = -\sqrt{R^2 - (Y\_out\_p + inc - y0)^2} + x0; \tag{3.51}$$

These equations are used in each time step of the real time process, till the second part is drawn.

- Drawing the third part is similar to drawing the first part, but $X\_out$ is incremented, and in the equation of $Y\_out$ the square root has a positive sign.
- Also drawing the fourth part is similar to drawing the second part, but$Y_out$ is decreased, and in the equation of $X\_out$ the square root has a positive sign.

After drawing the fourth part is done, the pen is lifted of the paper, and the state flow chart moves back to the **wait_1** block in the **start_stop** sub-chart, and waits there for the user's command,see figures 3.24,3.25.

- **Erase sub-chart**

To erase what was drawn on the playing board a small sponge that is attached to the lateral surface of the the end effector is used, see fig3.22. In order For the sponge to contact the playing filed, the rotation angle of the end effecter around the global X axis has to be equal to $90^o$, whereas it was equal to $0^o$ while drawing. The **Erase_flag** Matlab function turns on the erasing flag based on the number of symbols drawn by the manipulator and the number of symbols detected by the camera ,see figure 3.23. When the number is equal to 25, the flag is turned on and the State flow chart, which receives it as an input moves from **wait_1** block in the **start_stop** sub-chart to the **Erase** sub-chart,see figures 3.24 3.25. Simultaneously the switch block in fig 3.23 changes the rotation angle around the glotbal X axis to $90^o$.
The sub-chart generates Cartesian coordinates that correspond to the following steps :

- Lowering the end effecter while turned by $90^o$ around the X axis in order for the sponge to contact the board.
- Moving the end effecter along the Y axis back and forth with an offset of 3 cm between two consecutive passes.
- When the whole board is erased, it sets the playing field flag and the manipulator's symbols counter back to zero, which Along side the number of symbols detected by the camera turns off the erasing flag.
- The end effecter is turned back to its writing angle (rotation around axis X equals zero) and returns to the **wait_1** block in the **start_stop** sub-chart.

### 3.6.2 Tic-tac-toe algorithm

This algorithm is the brains of the Tic-tac-toe application. It receives the coordinates of the opponents moves , and the previous moves made by the manipulator as an input, whereas it outputs the coordinates of the manipulator's next move. It is designed for a 5 by 5 field, where player wins when having four consecutive symbols. It uses the minimax method with a static evaluation function. To reduce calculation time the algorithm only looks three steps ahead and through using the static evaluation function, it chooses the best move to play. When it is the manipulator's turn, the algorithm is activated, and the following expansions take place

- Expanding the current state to all the possible states, which the manipulator can choose from.
- Expanding each previous state into all the possible options, which the opponent has.
- Again expanding each state into all the possibilities available for the manipulator.

Using the static evaluation function, each state obtained after three expansions is given a score that represents the probability of winning :

- $-\infty$ for definite loss (Manipulator's perspective)
- $+\infty$ for definite win
- The state is not a definite win or loss: then the evaluation function equals number of possibilities for winning minus number of possibilities for loosing from the manipulator's perspective

When it is the opponent's turn the state with the lowest score is chosen, whereas when it is the Manipulator's turn the state with the highest score is chosen[14]. The figure below shows an example of the backpropagation of the minimax method in the Tic-tac-toe algorithm:

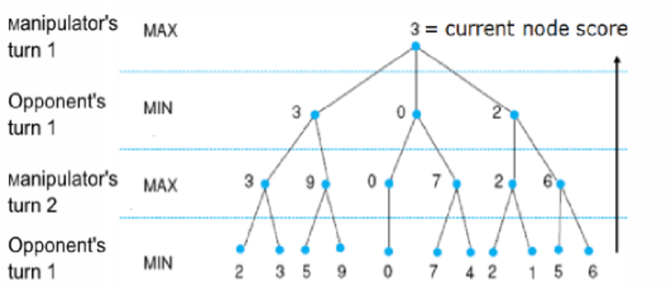


Fig. 3.28: Backpropagation in the Tic-tac-toe algorithm based on minimax

[14]the minimax method is based on maximizing wining probability of the player and minimizing wining probability of the opponent [4]

Previous algorithm is implemented into the Simulink model as shown below:

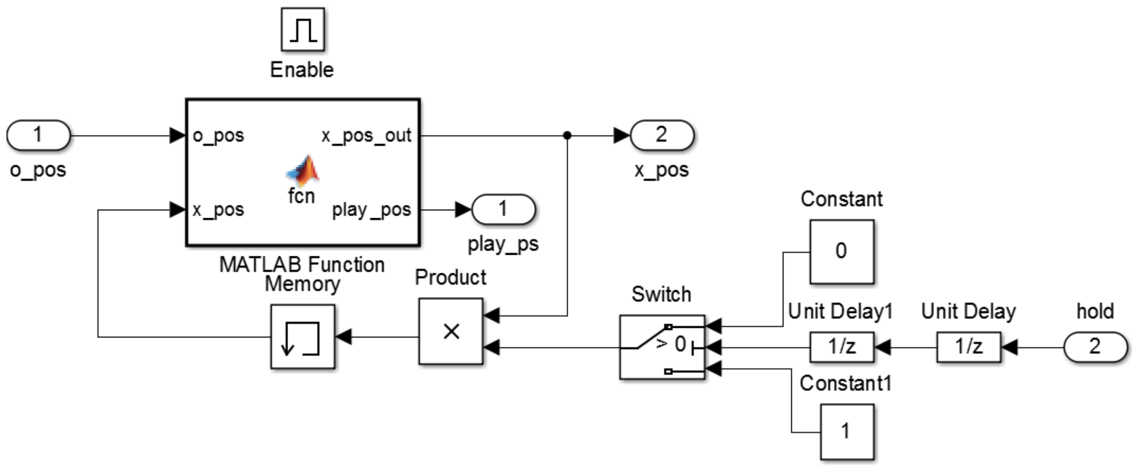


Fig. 3.29: Implementation of the Tic-tac-toe algorithm into Simulink

Blocks showed in fig 3.29 are implemented into an **enabled sub-system**. Therefore They are activated only when the sub-system is turned on.
The Matlab function block in fig 3.29 has two inputs:

- **o_pos:** the coordinates of the opponents moves sent by the real time camera .
- **x_pos:** The coordinates of the manipulator's moves that have been already calculated and remembered by the**Memory** block from the previous time step

Also it has two outputs:

- **play_ps:** A two element vector representing the coordinates of the manipulator's next move.
- **x_pos_out:** The coordinates of all the calculated moves by the algorithm.

The **hold** input resets the memory block elements to zero, when a game is over and the board is being erased, in order to enable the block to start a new game.
The code of the Matlab function in fig 3.29 [15] consitis of the following steps:

1. Combining both coordinate vectors (the opponent's and the manipulator's) into one matrix.
2. Checking if the entered vectors make the opponent win, if so abort the following steps, and output a coordinate that is out of range to refer to the opponent's win.

[15]Documented code used for the Tic-tac-toe algorithm is within the attachments

3. Check for an empty field and fill in with the manipulator's symbol. If there is not an empty field or, all of them have been used by this step, go to step 12.
4. Check if the move makes the manipulator win, and then:
    - If the move does make the manipulator win, score it as $+\infty$, and move back to step 3.
    - If the move does not make the manipulator win, move to next step.
5. For every new state generated by step number 3, check for an empty field and fill in with the opponent's symbol. If there is not an empty field or, all of them have been used by this step, go to step 11.
6. Check if the move makes the opponent win, and then:
    - If the move does make the opponent win, score it as $-\infty$, and move back to step 5.
    - If the move does not make the opponent win, move to next step.
7. For every new state generated by step number 5, Check for an empty field and fill in with the manipulator's symbol. If there is not an empty field or, all of them have been used by this step, go to step 10.
8. Check if the move makes the manipulator win, and then:
    - If the move does make the manipulator win, score it as $+\infty$, and move back to step 7.
    - If the move does not make the manipulator win, move to next step.
9. Calculate evaluation function for each state,

    $$f = n_1 - n_2 \tag{3.52}$$

    $n_1$: number of possibilities to win. $n_2$: number of possibilities to lose.
    then move back to step 7.
10. Preform backpropagation from **opponent's turn 2** to **manipulator's turn 2** based on minimax method, and move to step 5, see fig 3.28 .
11. Preform backpropagation from **manipulator's turn 2** to **opponents turn 1** based on minimax method, and move back to step 3, see fig 3.28.
12. Preform backpropagation from **opponent's turn 1** to **manipulator's turn 1** based on minimax method. Then move to step 13, see fig 3.28.
13. output the calculated coordinate for the manipulator's next move.

Previous Tic-tac-toe algorithm is activated by another Matalb function called **algorithm activator**. It has two inputs:

- **o_pos**: coordinates of the opponent's moves.
- **trigger**: a manual swith controlled by the user.

It has only one output connected to the enabling input of the Tic-tac-toe algorithm sub-system.

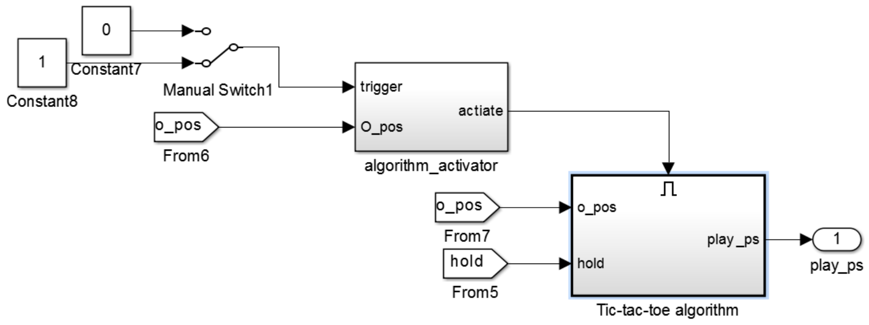


Fig. 3.30: Activision of the Tic-tac-toe algorithm

The **algorithm acitivator** compares the coordinates of the moves made by the opponent in the current time step to the same coordinates in the previous time step, and allows the user to activate the Tic-tac-toe algorithm using the manual switch only once as long as the coordinates are the same. Every time the coordinates change the user is allowoed to activate the algorithm one more time.

### 3.6.3 Real-time smart camera

Detecting the opponent's moves drawn on the board is an essential part of the Tic-tac-toe application. This is accomplished by using a real-time smart camera [16] manufactured by National Instruments. It helps to figure out the opponent's moves by sending data about the captured playing field to the Simulink model through the serial port of the Pc, see figures 3.22 3.21. The camera is equipped with its own processor, it can be programmed using Labview or Vision Builder, and the implemented program runs on the camera seperately from the Pc.
Figure below shows The used camera, attached to a stand that holds it above the drawing board.

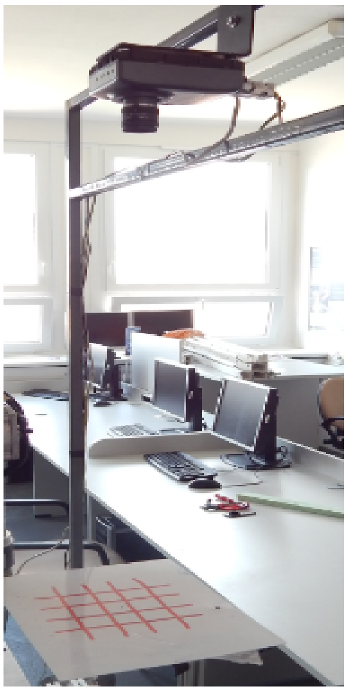


Fig. 3.31: Real-time smart camera

#### Programming the real-time camera

The Vision Builder environment was used to realize the detection algorithm, and implement it into the camera. Two detection algorithms were considered:

- **Symbol detection algorithm**: Vision Builder provides blocks that allow comparing the drawn symbols on the board to an already defined template that

[16]Describing a camera by being real-time smart refers to the fact that it has its own real time processor and can be programmed and operated independently [6] [7]

has the shape of an X or O according to the chosen symbol by the opponent to play with. Besides detection, these blocks output the coordinates of the detected symbols, which are sent to the Tic-tac-toe Simulink model, and then used to figure out the fields, where the symbols were played.
Although This method has the advantage of recognizing only X and O symbols among the drawn ones, and realizing where each of these symbols has been drawn, it has one disadvantage that can not be ignored, which is lack of robustness due to:

– Probable differene between drawn shapes by the opponent and the template. Although previous blocks have parameters that set invariance to rotation and size of the detected object, setting these parameters to a very low value makes the camera detect undesired shapes such as shapes drawn by the manipulator or even parts of the playing field, which would cause bad operation of the Tic-tac-toe application.
– Dependence of the detection quality on lighting circumstances. Difference in lighting while running the application form lighting used for capturing the template might cause missing symbols that should have been detected.

- **Color detection algorithm**: What was mentioned before confirmed the necessity of implementing a more robust algorithm, in order for the Tic-tac-toe application to run properly. By obtaining the percentage of pixels in one field that have a value bigger than a defined threshold(value of detecting a gray color by a pixel), the program can know for sure whether the field is clear, or filled with a symbol.
  The figure below shows the histogram of the pixels of one field, when filled with a symbol:

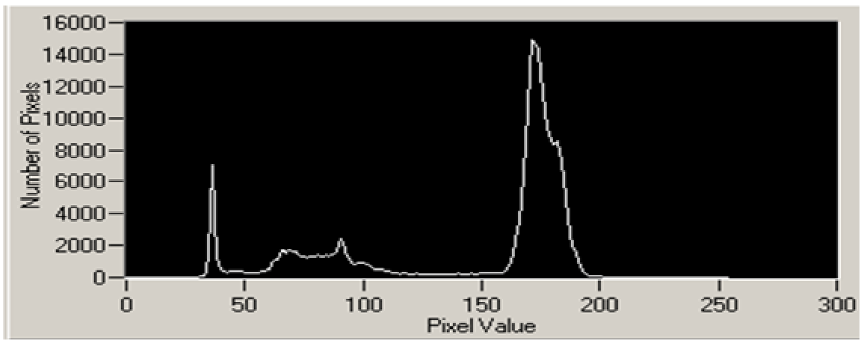


Fig. 3.32: Pixels' histogram of a field filled with a symbol.

If the threshold is set to 80, figure 3.32 is split into two parts:

- Above 80: has a peak that refers to the pixels capturing the clear area of the field.
- Below 80: has a peak that refers to the pixels capturing the drawn symbol. This peak indicates whether a symbol is drawn or not.

The color detection algorithm counts the number of pixels with a value less than the defined threshold. If they exceed a certain precentage of the field's pixels, then the field is designated as filled with a symbol.

For this algorithm to work propely the player should use a dark marker on the white board, to have the desired difference in pixels' values.
This algorithm has the following advantages:

- It is really robust when it comes to lighting circumstances. It depends on the difference in pixels values when exposed to a dark color form their values when exposed to a light color, without taking into consideration the change in lighting.
- It does not use any comparison to defined templates, which caused problems in the first algorithm.
- Its computational complexity comparing to the previous symbol detection algorithm is very low.

But still it has a few disadvantages :

- Not knowing anything about the shape drawn by the player. The player can draw any shape and still be detected as an X or O.
- Every checking area should be set to one field without intersecting with the field's borders. Therefore both fields and checking areas should be fixed. This is accomplished by using the manipulator to draw the playing field and attaching the camera firmly to the stand.

Previous disadvantages does not really affect the Tic-tac-toe application. Therfore the color detection algorithm was chosen, and implemented using Vision builder, which provides a block that can obtain the percentage of pixels, which have a value over a defined threshold in a certain area.
The implementation was done by using 25 of these blocks, and setting each block's detection zone to the area of a different field, and then sending the obtained percentages through serial port to the Pc.

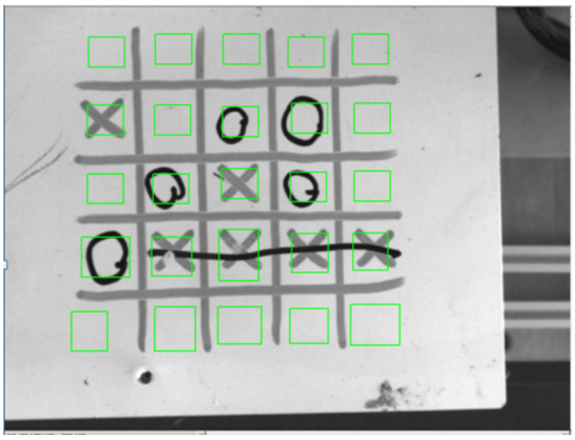
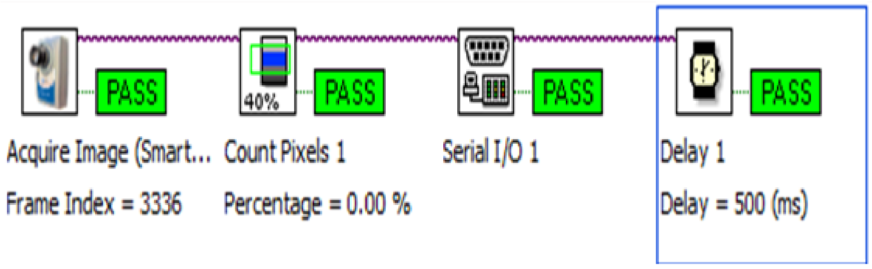


Fig. 3.33: Analyzed areas by the detection algorithm



Acquire Image (Smart... Frame Index = 3336
Count Pixels 1 Percentage = 0.00 %
Serial I/O 1
Delay 1 Delay = 500 (ms)

Fig. 3.34: Detection algorithm implementation using Vision Builder.

### Obtaining coordinates of the opponent's moves

Previous percentages are received by the Simulink model of the Tic-tac-toe application using the **Stream input** real time block. A Matlab function called **Precent-process** that has two inputs:

- Precentages sent by the camera.
- Coordinates of the moves made by the manipulator.

  Outputs the coordinates of the opponent's moves. It checks which of the precentages exceeds a certain value, and then excludes the coordinates of the manipulator's moves to get the coordinates of the opponent's[17].

[17]The simulink-model of the Tic-tac-toe application is in the attachments

## 3.7 Designing user interfaces

To make Tic-tac-toe application, and interacting with the manipulator more user friendly two control panels have been designed using the Graphical User Interface (GUI) environment, which Matlab provides.

### Tic-tac-toe GUI

This GUI enables the user to play Tic-tac-toe agianst the manipulator, while displaying the game's progress taking place on the board. It enables the user to choose the playing symbol for the manipulator, and command it to play, when it is its turn.

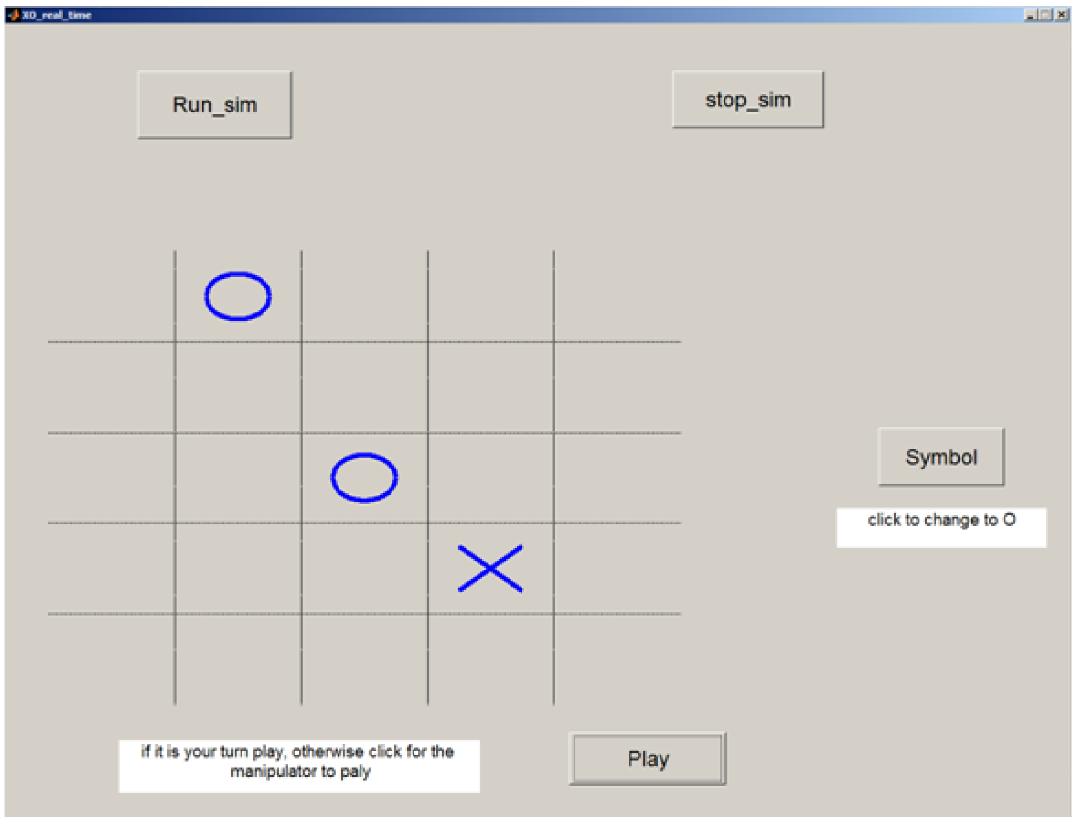

XD_real_time
Run_sim
stop_sim
Symbol
click to change to O
if it is your turn play, otherwise click for the manipulator to paly
Play

Fig. 3.35: Tic-tac-toe GUI

The figure above shows the desigend GUI, which consits of the following components:

- **Run_sim** button: Enables the user to compile the Simulink model of the Tic Tac Toe game, connect to the Mf624 card and run the real time process.
- **Symbol** button: Enables the user to choose the manipulator's playing symbol (whether it is X or O).
- **Play** button: Clicking this button commands the manipulator to carry out its next move.
- **Stop_sim** button: Enables the user to stop the Tic-tac-toe Simulink model. The manipulator is returned to its starting position avoiding the drawing board, and then the model is stopped.
- **Text box1**: Displays a massage that explains whether the user can change the playing symbol of the manipulator to an (X) or to an (O).
- **Text box2**: It displays the following massages:
  - **Wait for the model to be compiled**.
  - **The compilation is done wait for the manipulator to be initialized.**
  - **If it is your turn to play draw your next move, otherwise click for the manipulator to play**
  - **The manipulator is drawing.**
  - **The manipulator is erasing.**
  - **The manipulator is being turned off.**

This GUI is based on using functions set and get to exchange data with the Simulink model. The visualization of the Tic-tac-toe game is done by getting the coordinates of the manipulator's moves and the moves of the player form the Simulink model, and using the **Plot** Command within a while loop[18].

[18]The source code of the Tic-tac-toe GUI is in the attachments.

## Operational GUI

This GUI enbales generating a few defined movements by the manipulator such as drawing certain shapes ...... , and displays data acquired by the sensors to give the user an idea about the quality and safety of the operation. The user can change the shapes' sizes, in addition to controlling the speed of drawing.

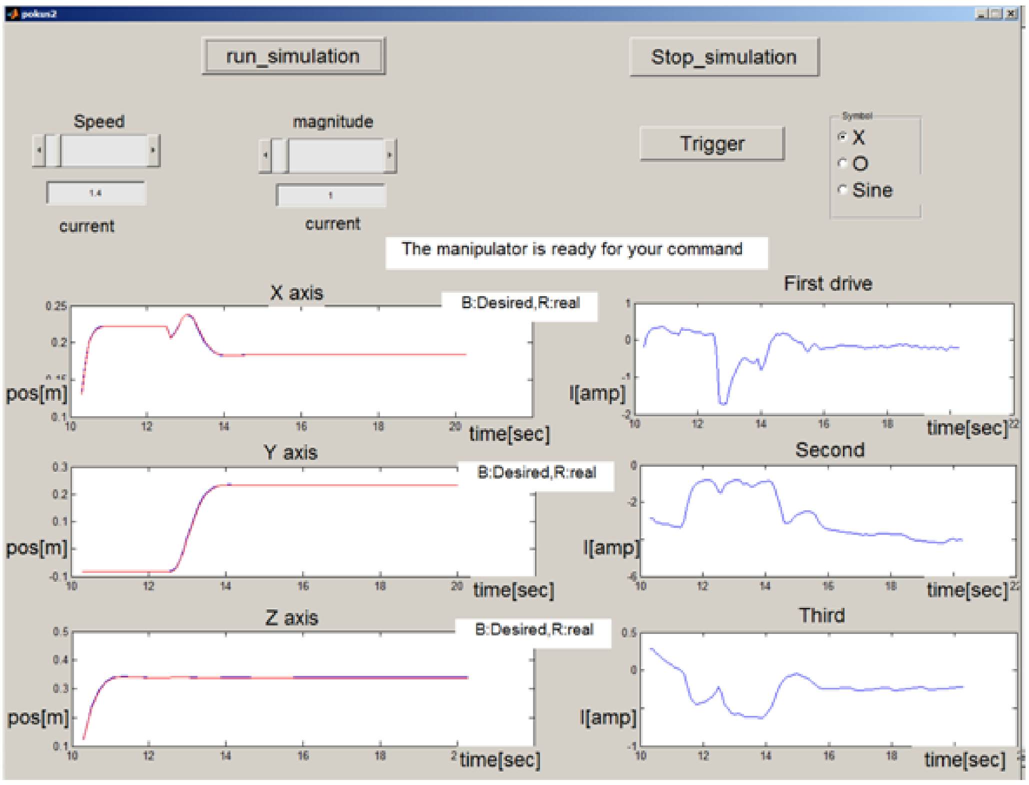


Fig. 3.36: Operational GUI

The figure3.36 shows the designed GUI, which consists of the following parts:

- **Run_simulation** button: Enables the user to compile the Simulink model of the manipulator, connect to the Mf624 card, and run the real time process.
- **Symbol** list Box: Enables choosing the symbol desired to be drawn. The user can choose an X, O or a sine wave for the manipulator to draw.
- **Trigger** button: Enables the user to command the manipulator to draw, when it is in the waiting position near the board.

- **Magnitude** slider gain: Enables the user to choose the amplitude of the sine wave, the diameter of the circle, or how big the X symbol is.It is connected to a box that displays its value. The slider Gain's value could be changed either by typing the desired value into the previous box or by manual sliding.
- **Speed** slider gain: Enables the user to set the speed of drawing. It is connected to a similar box that also displays its value. The slider gain's value can be changed either by typing the desired value into the previous box or by manual sliding.
  **Note**: The range of the **Magnitude** slider gain is from 0 to 10 cm. In case of a sine wave, the value of the magnitude is the peak to peak distance of the sine wave. The value of the Speed slider gain ranges from 0 to 5. 0 is minimum speed, and 5 is maximum.
- Static text that displays a massage, which corresponds to the manipulator status. The following massages are displayed:
  - **The manipulator is being initialized**: This massage is displayed, while the manipulator is performing the initialization process for the encoders.
  - **The manipulator is ready for your command**: This massage is displayed while the manipulator is in the waiting position near the drawing board.
  - **The manipulator is drawing**: It is displayed while the manipulator is drawing the chosen symbol.
  - **The manipulator is being turned off**: It is displayed while the manipulator is being moved to its starting position and stopped.
- Graphs:
  - **X axis, Y axis, Z axis**: Each graph is a comparison of the desired value in the given axis, and the real value calculated by using the Block of forward kinematics on the rotation angles of the encoders.
  - **First drive, Second drive, Third drive**: Each graph shows the drawn current by the corresponding motor.

**Note**: This GUI is mainly based on using functions set and get[19] to transfer data back and forth between the Simulink model and the Matlab Script. A while loop that runs with a frequency of (10Hz) enables drawing the previous graphs while the real time process is running[20].

[19] See Matlab help for more information about functions set and get
[20] the Documented source code of the Operational GUI is in the attachments

# 4 Conclusions

This thesis aimed to develop the 3dof manipulator created last year in order to perform more complex applications such as playing Tic-tac-toe against a human being. First part of the thesis deals with enhancing previous software developed for the manipulator by my colleagues. In addition to improving the initialization process of the manipulator to be more robust by using State Flow, which is necessary for the relative encoders, the simple PID controllers have been replaced with a better cascade controllers which has increased the precision and efficiency of operation.
The Second part focused on expanding the manipulator by one more degree of freedom to become a 4dof manipulator. This allows controlling the rotation of the end effecter, which is essential in writing applications. A servo motor is used as a fourth drive, and the end effector attached to it was designed in Solid works, and realized using a 3D printer.
In order to control the expanded manipulator, the third part of the thesis analyzes its kinematics and chooses the least damped squares algorithm to solve the inverse kinematics problem. The solution is implemented into a matlab function used in the Simulink environment.
Previous parts are necessary foundations for the Tic-tac-toe application, which is explained in the fourth part of this thesis. The designed solution for the application includes a **Control** block responsible for generating Cartesian coordinates that correspond to the desired shape or movement. This block recieves the coordinates of the manipulator's moves from the **Tic-tac-toe algorithm** that uses basics of artificial intelligence to calculate and output the most proper move. The algorithm is implemented into a Matlab function that receives the coordinates of the opponents moves form a real-time smart camera. A proper and a simple detection algorithm was chosen and implemented into the camera using Vision Builder environment.
Fifth part is about making previous application and the operation of the manipulator more user friendly, which was done by the GUI environment, which Matlab provides for designing user interfaces.
This thesis managed to enhance the previously built 3dof educational platform and realize an interesting application( playing Tic tac toe against a human being ) that could be a real inspiration for potential students of mechatronics.

# 5 References

[1] Klimeš, D.: *Hardwarové a softwarové řešení diagnostiky a bezpečnosti provzu robotického manipulátoru.*, [ Master's Thesis.] Brno: VUT, Faculty of mechanical engineering, 80 s., 2013

[2] Grepl, R.: *Kinematika a dynamika mechatronických systémů*, Vysoké učení technické v Brně, 2007

[3] Samuel R. Buss : *Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods*, Department of Mathematics University of California, San Diego, 2009

[4] Jones,M.Tim : *Artificial Intelligence* , Library of Congress Cataloging in-publication Data, 2009

[5] Grepl, R.: *Modelování mechatronických systémů v Matlabu/SimMechanics*, BEN, 2007

[6] NI 1742 Smart Camera :, `http://sine.ni.com/nips/cds/view/p/lang/cs/nid/204079`, [online], 2014-04

[7] NI 17xx Smart Camera User Manual: *Austin*, 2008,

[8] National Instruments: *NI Vision Builder for Automated Inspection Tutorial*, Austin, 2012

[9] Šuranský, M.: *Modelování, identifikace a řízení robotického manipulátoru.*, [Master's thesis] Brno: VUT, Faculty of mechanical engineering, 2013

[10] A.S. Deo, I.D. Walker: *Adaptive Non-linear Least Squares for Inverse Kinematics*, Electrical Engineering Rice University Houston, 1992

[11] A.S. Deo, I.D. Walker: *Robot Subtask Performance with Singularity Robustness using Optimal Damped Least-Squares*, Electrical Engineering Rice University Houston, 1992

[12] Kareem Abdel-Malek, Harn Yeh: *Analytical Boundary Of The workspace For General 3-Dof Mechanism* , Department of Mechanical Engineering The university of Lowa, 1997

# 6 Appendix

1. **Tic-tac-toe Game** - Simulink model for playing Tic-tac-toe (manipulator vs humna being).
2. **Camera program**- detection program for the Tic-tac-toe application built in Vision Builder.
3. **Operational GUI** - Simulink model, M-file and GUI panel for the operational interface.
4. **Tic-tac-toe GUI** - Simulink model, M-file and GUI panel for the user interface of the Tic-tac-toe application.