5 degrees of freedom robot arm programming

Barbara Klojbert, Dominik Tál, Patrik Dósa

2018

University of Pannonia

Faculity of Information Technology

Department of Electrical Engineering and Information Systems

Engineering Information Technology Msc

Project Labor

5 degrees of freedom robot arm programming

Barbara Klojbert, Dominik Tál, Patrik Dósa

Supervisor: Dr. Attila Magyar

2018

Contents

List of Figures

1.
Introduction

Nowadays robots are not these futuristic and weird shaped machines as they were in the early 2000s American films. They are part of our life, and they make our days easier, more productive, and safer. It is pretty hard to define, what makes a robot, but we have some points which are true for all of them. First of all, we want to clarify, that our documentation will not say anything about software robots, we leave this topic to the IT security experts. The most important thing about machine robots is that they are programmable, and according to this program, it can make manipulations automatically. They can have external controller inputs, or they can have an embedded control. [2]

1.1 History of robots

Robots have their roots from the ancient ages, these cultures wanted to make automated devices for entertainment purposes, but mankind only had the proper materials and methods from the industrial ages, and in these ages they wanted to use robots in the industry to make the production cheaper and faster. As we reached the modern ages, engineers and inventors introduced automatic, remote controlled, and wirelessly remote controlled robots too. [1]

The word robot, comes from the Czech language, and it means “forced labor”. In its modern form it has been firstly used by Karel Capek’s play in 1920. The science of robotics is the engineering field which deals with the design, construction and operation of robots. The word of robotics has been introduced in Isaac Asimov’s Sci-Fi “Runaround” in 1942. In his books Asimov has written down the three laws of robotics. These are the following: [1]

1. ”A robot may not injure a human being, or, through inaction, allow a human being to come to harm.”

2. ”A robot must obey the orders given it by human beings except where such

orders would conflict with the First Law."

3. "A robot must protect its own existence as long as such protection does not conflict with the First or Second Law."

1.2 Modern robots

In every modern factory you can see robots. Robotic arms which assemble whole cars, robot trains which transport products to the warehouse, industry 4.0 is not the future, we live in it.

Why are robots good for mankind? They can make monotonous processes all day long without a lunch or a cigarette break. They can be used in hazardous areas without any health risks. To cut a long story short, they make everything more productive, safer, quicker. Their designers should be aware that they have to help humans work, not to take all of their work to make human workers useless.

According to industry 4.0 the separate robots and other machines are in a huge common network, and they synchronise their works, utilize the resources to reach the optimum. [2]

1.2.1 Types of modern robots

- Mobile robots
- Industrial (manipulator) robots
- Service robots
- Educational robots
- Modular robots
- Collaborative robots

1.2.1.1 Industrial (manipulator) robots

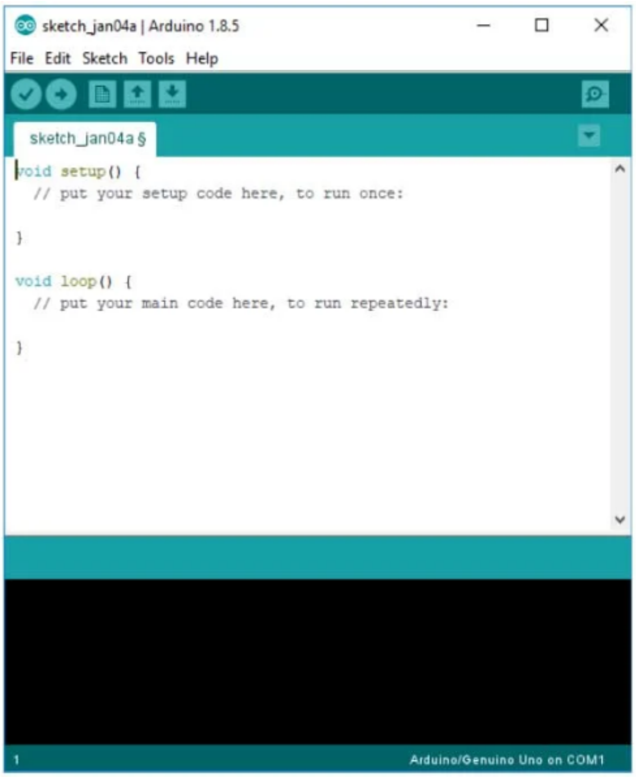
Industrial robots are used for manufacturing, they are programmable and automated. They can move on two or more axes, but they are not able to change their physical position.

Their fundamental is the same, this structure has to be able to move a specific tool to the certain position, and hold it there until the manipulator finished its work. This manipulator can do welding, painting, screwing, lifting, etc. [2]

2.
Applied devices

2.1 Arduino IDE

We used Version 1.0.6 of the Arduino IDE to program the Arbotix-M microcontroller. The Arduino IDE is an open-source software which can be used to implement, compile and upload our codes to the microcontroller. The IDE can run on Linux and Mac OS as well as on Windows. Everyone can use the IDE to program any Arduino or Arduino based microcontroller like the Arbotix-M. The environment was created in Java and with other open-source and free software. Arduino has lots of advantages like: easy way to use, huge community who use Arduino, so they can help through online communication. [5] [6]


sketch_jan04a | Arduino 1.8.5
File Edit Sketch Tools Help
sketch_jan04a §
Arduino/Genuino Uno on COM1

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 2.1 Arduino IDE [7]

2.2 Arbotix-m Robocontroller

This is a versatile Arduino based robot controller method for systems which based in BIOLOID and DYNAMIXEL. Moreover this controller allows direct connection between the Dynamixel network and the board. ArbotiX-M Robocontroller has lots of advantages, as another Arduino compatible microcontroller, for example open source community of libraries and examples. The main difference between Arbotix and Arbotix M are the following:

- In name of Arbotix M m stand for mini, cause it's smaller than Arbotix.
- A smaller footprint(61mm X 61mm)
- 10 more general purpose I/O pins
- Barrel Jack for DC Power
- Separate Power Bus for 4 PWM Digital outputs for Hobby Servos
- No dedicated I2C header (I2C is still available on D16 = SCL D17=SDA)
- No DC Brushed Motor Controller
- Not compatible with the RX-Bridge

So the Arbotix-M has the following properties:

- 16MHz processor (ATMEG644p)
- 2 serial ports, these make easy to program using FTDI to USB connection.
- 3 TTL Dynamixel network interfacing ports (for direct motor connection)
- Analog and digital pins

Last but not least arbotixM robocontroller has a big disadvantage. Usually we said: Arbotixm is cheaper compare to the overall price of robots. Because of this, it's not got the fastest processors or the best memory.

2.3 Pincher Robotic Arm

The robot arm is a 5 degree-of-freedom robotic arm. This means the robot arm can can rotate along 5 different axes. [5]

Pincher Robotic Arm Specification	
Weight	550G
Vertical Reach	35CM
Horizontal Reach	31CM

2.3.1 Servo motor

We have used Dynamixel AX-12A Robot Actuator servos in our project. The are powerful, durable and they have diagnostic functions too. They can monitor their voltage, and it is very important in protecting our battery from the damages of a critical undervoltage situation.

These servos can measure their rotating angle, so they can be controlled very precisely. [3]

Servo motor specification	
Weight	53.5g (AX-12/AX-12+), 54.6g (AX-12A)
Dimension	32mm * 50mm * 40mm
Resolution	0.29°
Gear Reduction Ratio	254 : 1
Stall Torque	1.5N.m (at 12.0V, 1.5A)
No load speed	59rpm (at 12V)
Running Degree	0
Running Temperature	-5°C ∼+70°C
Voltage	9 ∼12V (Recommended Voltage 11.1V)
Command Signal	Digital Packet
Protocol Type	Half duplex Asynchronous Serial Communication (8bit,1stop,No Parity)
Link (Physical)	TTL Level Multi Drop (daisy chain type Connector)
ID	254 (0-253)
Communication Speed	7343bps ∼1 Mbps
Feedback	Position, Temperature, Load, Input Voltage, etc.
Material	Engineering Plastic

2.3.2 Pincher tool

Our robot arm uses a so called pincher tool. Its maximum width is <> centimeters. It can produce a <> N holding force when closed. This tool is controlled by the fifth servo of the system, and it is made from industrial plastic too. [3] [4]

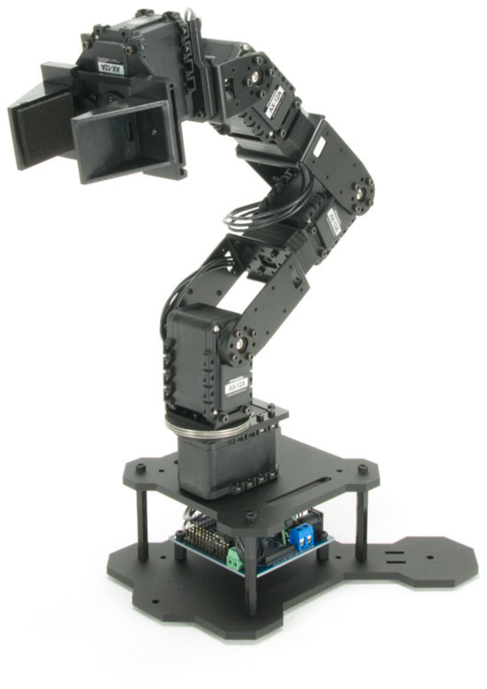


Figure 2.2 Pincher [4]

2.4 Programming language

Arduino doesn't run either C or C++. It runs machine code that compiled from C, C++ or any other language. The Arduino IDE accepts C and C++ as it is. Most of the libraries are written in C++, but it is very different from most C++ varieties. The Arduino C++ language has a lot of abstraction built in.

Most of the embedded systems in the microcontroller world are using C++ language. Arduino code's core functions are simply a set of the C++ classes and libraries. Arduino language simplifies a lot of things, but we can still write our functions as we want and where we need more specialization.

3.
System design

First of all see the design in picture:

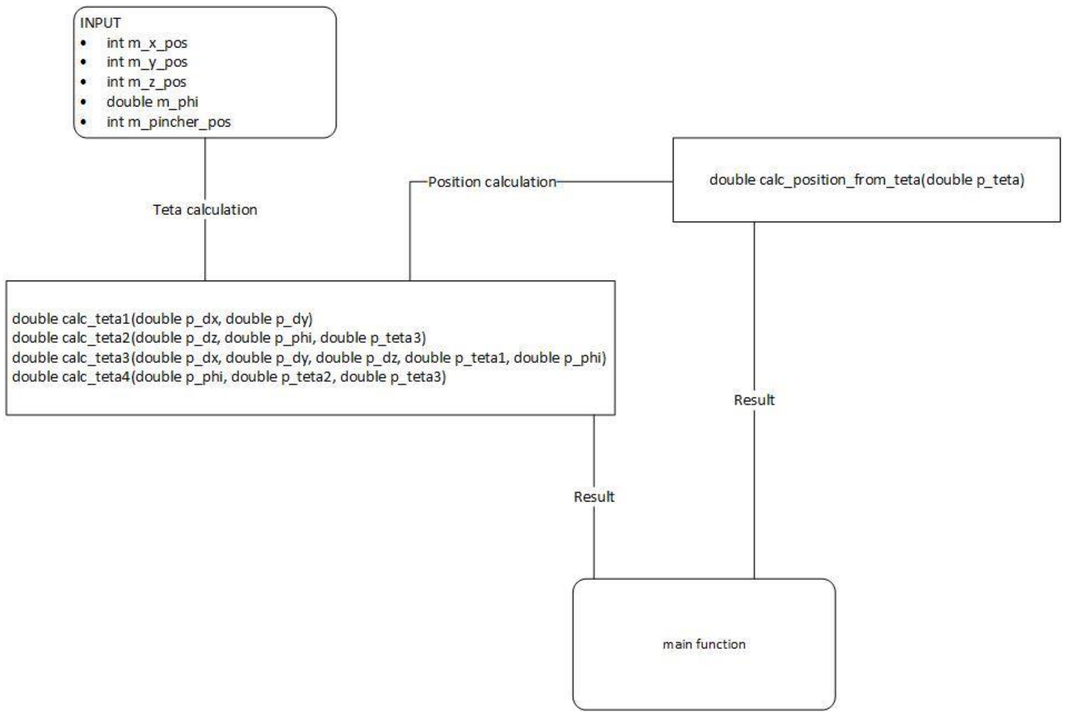

INPUT
int m_x_pos
int m_y_pos
int m_z_pos
double m_phi
int m_pincher_pos
Teta calculation
Position calculation
double calc_position_from_teta(double p_teta)
double calc_teta1(double p_dx, double p_dy)
double calc_teta2(double p_dz, double p_phi, double p_teta3)
double calc_teta3(double p_dx, double p_dy, double p_dz, double p_teta1, double p_phi)
double calc_teta4(double p_phi, double p_teta2, double p_teta3)
Result
Result
main function

Figure 3.1 System design

Generally about the Denavit-Hartenber parameters For general description of robot geometry exist a method called Denavit-Hartenberger transformation matrix. With this a coordinate system can be transmitted in any coordinate system, if two offsets and two rotations are used in a corresponding order. In this case the two distance marked with d and a, and the rotation parameters are the following: θ, q.

General steps:

1. Define the coordinate system for the wrists of the robot.

(a) For the i and i+1 wrists take a rectangular coordinate system, so the origin of the coordinate system fits into the wrist centers. The z axes of the coordinate system points to the direction of wrists.

2. Define the conventional parameters

(a) First parameter: q_i in case of rotation wrist this is the size of the angle between x_i and $x_(i-1)$ axes.

(b) Second parameter: d_i:the distance of the normal transverse of the wrists.

(c) Third parameter: a_i: length of the distance of the i-th and (i+1)-th wrist axes.

(d) Fourth parameter: α_i: angle between $z_(i-1)$ axis of i-th wirst and z_i axis of (i+1)-th wirst.

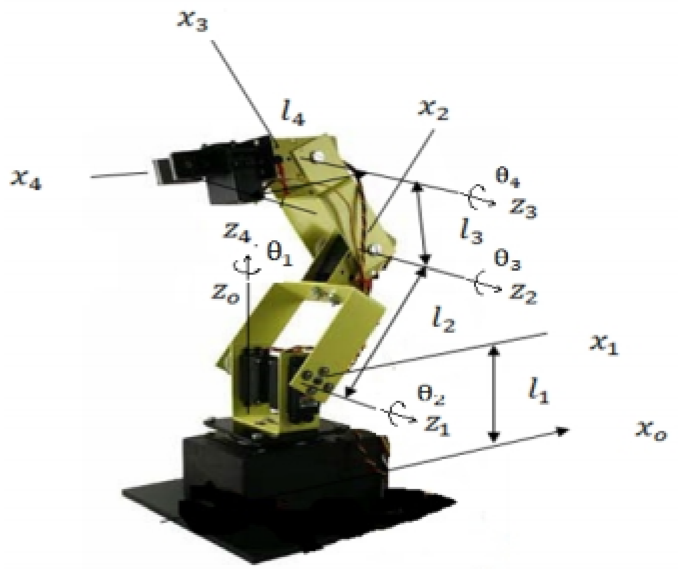


Figure 3.2 a. Denavit-Harzenbergeer parameters of the robot

3 The steps of matrix transformation:

$$\mathbf{D}(q_i)=\begin{bmatrix}\cos(q_i) & -\sin(q_i) & 0 & 0\\ \sin(q_i) & \cos(q_i) & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}.$$

Figure 3.3 a. Matrix transformation

$$\mathbf{D}(d_i)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & d_i\\ 0 & 0 & 0 & 1\end{bmatrix}.$$

Figure 3.4 b. Matrix transformation

$$\mathbf{D}(a_i)=\begin{bmatrix}1 & 0 & 0 & a_i\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Figure 3.5 c. Matrix transformation

$$\mathbf{D}(\alpha_i)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos(\alpha_i) & -\sin(\alpha_i) & 0\\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Figure 3.6 d. Matrix transformation

$${}^{i-1}\mathbf{D}_i=\begin{bmatrix}\cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & 0\\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & 0\\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & q_i\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Figure 3.7 e. Matrix transformation

The Denavit-Hartenberg parameters can be listed in a table as the follows:
Our robots parameters is the following:

Frame (i)	a_i	α_i	d_i	θ_i
1	0	90	I_1	θ_1
2	I_2	0	0	θ_2
3	I_3	0	0	θ_3
4	I_4	0	0	θ_4

Table 3.1 Table of Denavit-Hartenberg parameters

Parameters	Robot parameters in mm
I_1	8
I_2	101
I_3	101
I_3	101

Table 3.2 Table of robot parameters

In the end T matrix will specify the position and orientation of the robot in the standing coordinate system. After we have to using for this trigonometric formulas:

$$dx = \cos\theta_1[l_2 \times \cos\theta_2 + l_3 \times \cos(\theta_2+\theta_3)] + l_4 \times \cos\theta_1 \times \cos(\theta_2+\theta_3+\theta_4)$$

$$dy = \sin\theta_1[l_2 \times \cos\theta_2 + l_3 \times \cos(\theta_2+\theta_3)] + l_4 \times \sin\theta_1 \times \cos(\theta_2+\theta_3+\theta_4)$$

$$dz = [l_1 + l_2 \times \sin\theta_2 + l_3 \times \sin(\theta_2+\theta_3)] + l_4 \times \sin(\theta_2+\theta_3+\theta_4)$$

With the help of this parameters we can set the inverse kinematics on our robot as in the 4.4.

4. Implementation

4.1 Setup function

The setup function in the Arduino is basically to create those things which is only one-run things. In our code in the program start we would like to open the Serial connection to communicate with the Arbotix-M microcontroller. After that if we are currently using the Arbotix library then we define the center position (Which was already defined earlier). On the start `set_position_slowly` function will set the position of the robot to center. This function will be discussed later. If relax is needed then the Relax function wil relax all the servo motors one-by-one.

```
void setup() {

  // Open serial connection
  Serial.begin(9600);
  delay (500);
  Serial.println("###########################");
  Serial.println("Serial Communication Established.");

  // If 'set all servo to center position' is required
  // Set everything to center
  Serial.println("###########################");

  Serial.println("Set all servo motor to center (512).");
  #ifdef Arbotix

    const PROGMEM unsigned int m_init_pos[] = {5, 512, 512, 512, 512,
        512};
    set_position_slowly(m_init_pos);

   delay(1000);

```

```
    #ifdef RELAX
      for (int i = 1; i<= 5; i++){
        Relax(i);
      }
   #endif

  #endif
}
```

4.2 Paramteres

Our parameter list is not too much. We need to define the distances of the axes in milimeter.

```
// Length in mm
#define L1 82.0
#define L2 101.0
#define L3 101.0
#define L4 101.0
```

4.3 Acquire the required position values

All the other required values will be given by the user. Serial communication ensures to read more than one character on the serial in the following way. We read the characters one-by-one (with the negative sign too.)

4.3.1 X, Y and Z position

First we need to get the X, Y and Z values. These values will be our 3D space values. (X - width, Y - depth, Z - height). For example (X, Y, Z) = (0, 0, 385) is the center position, because the Z position need to be the maximum length of the robot.

```
    // Read the required X value
    Serial.println("###########################");
    Serial.print("X position: ");

    // Wait til the serial input
    while(!Serial.available()){}
    delay(100);

```

```
  // Something came across serial
  if (Serial.available() > 0) {
    // Read the required x position
    while(Serial.available() > 0) {
      incomingByte = Serial.read();

      if(incomingByte == '-'){
        negative = 1;
      }else{
        // Shift left 1 decimal place
        m_x_pos *= 10;

        // Convert ASCII to integer, add, and shift left 1 decimal
            place
        m_x_pos = ( (incomingByte - '0') + m_x_pos);
      }
    }

    if(negative){
      // Set X position as a negative value
      m_x_pos = 0 - m_x_pos;

      // Init negative again
      negative = 0;
    }

    Serial.println(m_x_pos);
  }
```

Reading of the Y and Z positions is the same. We don't want to show these, because there is only variable change in the method.

4.3.2 Phi angle

Reading of the phi angle is almost the same as the position reading from the serial. The difference is we read the angle, but our calculation use radian values, so a conversion needed as you can see in the code below.

```
  // Read the required Phi value
  Serial.println("###########################");
  Serial.print("Phi degree: ");

  // Wait til the serial input
  while(!Serial.available()){}
  delay(100);

  // Something came across serial
  if (Serial.available() > 0) {
    // Read the required phi
    while(Serial.available() > 0) {
      incomingByte = Serial.read();

      if(incomingByte == '-'){
        negative = 1;
      }else{
        m_phi *= 10; // shift left 1 decimal place

        // Convert ASCII to integer, add, and shift left 1 decimal
            place
        m_phi = ( (incomingByte - '0') + m_phi);
      }
    }

    if(negative){
      // Set phi as a negative value
      m_phi = 0 - m_phi;

      // Init negative again
      negative = 0;
    }

   m_phi = m_phi * (M_PI / 180.0);

    Serial.println(m_phi);
```

4.3.3 Pincher position

Reading the pincher position from the Serial is require only a 0 or 1 value. 0 for the closed position, 1 for the open position of the pincher. It can be changed if we want to use a more exact pincher position, but in our project work we don't want to define it. If the read pincher position value is differ from the boolean value then the default value will be the closed position (0).

```
    // Read the required pincher value
    Serial.println("###########################");
    Serial.print("Pincher (0/1): ");

    // Wait til the serial input
    while(!Serial.available()){}
    delay(100);

    // Something came across serial
    if (Serial.available() > 0) {
      // Read the required pincher position
      while(Serial.available() > 0) {
        incomingByte = Serial.read();

      m_pincher_pos = incomingByte - '0';

      }

     //Set pincher to close if the value incorrect
     if( (m_pincher_pos != 1) || (m_pincher_pos != 0) )
      m_pincher_pos = 0;


      Serial.println(m_pincher_pos);

     // Open pincher
     if(m_pincher_pos)
       m_pincher_pos = 512;

    }
```

4.3.4 Default values

If we don't want to acquire the positions and the angle then we use the default center position values (0,0,385), 180° and closed pincher.

4.4 Teta calculations

Each teta calculation formula will be defined in their own section based on the inverse kinematics. All the calculations are using the following calculated values:

$$a = l_3 \times sin\theta_3$$

$$b = l_2 + (l_3 \times cos\theta_3)$$

$$c = dz - l_1 - (l_4 \times sin\phi)$$

$$r = \sqrt{a^2 + b^2}$$

$$A = dx - (l_4 \times cos\theta_1 \times cos\phi)$$

$$B = dy - (l_4 \times sin\theta_1 \times cos\phi)$$

$$C = dz - l_1 - (l_4 \times sin\phi)$$

4.4.1 Teta1

$$\theta_1 = \arctan(\frac{dy}{dx})$$

```
double calc_teta1(double p_dx, double p_dy){
  //Teta1 calculation
  double m_teta1 = atan( p_dy / p_dx );

  return m_teta1;
}
```

4.4.2 Teta2

$$\theta_2 = \arctan(c, \pm\sqrt{r^2 - c^2}) - \arctan(a, b)$$

```
double calc_teta2(double p_dz, double p_phi, double p_teta3){
  // Basic calculation for Teta2
  double m_a = L3 * sin(p_teta3);
  double m_b = L2 + ( L3 * cos(p_teta3) );
  double m_c = p_dz - L1 - ( L4 * sin(p_phi) );

  // R calculation based on m_a and m_b for Teta2
  double m_r = sqrt( (m_a*m_a)+(m_b*m_b) );

  // Teta2 calculation based on the other values
  #ifdef LEFT_HANDED // If left handed arm
    double m_teta2 = atan2(m_c, sqrt((m_r*m_r) - (m_c*m_c)) ) -
        atan2(m_a, m_b);
  #else              // If right handed arm
    double m_teta2 = atan2(m_c, (-sqrt((m_r*m_r) - (m_c*m_c)) )) -
        atan2(m_a, m_b);
  #endif

  return m_teta2;
}
```

4.4.3 Teta3

$$\theta_3 = \arccos(\frac{A^2+B^2+C^2-l_2^2-l_3^2}{2\times l_2 \times l_3})$$

```
double calc_teta3(double p_dx, double p_dy, double p_dz, double
    p_teta1, double p_phi){
  // R calculation based on m_a and m_b for Teta3
  double m_A = p_dx - (L4*cos(p_teta1)*cos(p_phi));
  double m_B = p_dy - (L4*sin(p_teta1)*cos(p_phi));
  double m_C = p_dz - L1 - (L4*sin(p_phi));

  // Teta3 calculation based on the other values
  double m_teta3 = acos( ( (m_A*m_A) + (m_B*m_B) + (m_C*m_C) -
      (L2*L2) - (L3*L3) ) / (2*L2*L3) );

  return m_teta3;
}
```

4.4.4 Teta4

$$\theta_4 = \phi - \theta_2 - \theta_3$$

```
double calc_teta4(double p_phi, double p_teta2, double p_teta3){
  // Teta4 calculation based on the parameters
  double m_teta4 = p_phi - p_teta2 - p_teta3;

  return m_teta4;
}
```

4.5 Calculating position value from teta

It is not enough if we know the teta values because the servo motors need a position input to be in the right angle. The following code part is calculate the right position from the calculated radians. In this function we need to calculate the angles back from the radian. After that it is just an easy matemathical calculation to acquire the right position. If we know that the $0°$ is the position value 0 while the $360°$ is the position value 1023.

```
int calc_position_from_teta(double p_teta){
  if(isnan(p_teta))
    p_teta = 0.0;

  double m_degrees = 0.0;
  m_degrees = p_teta * (180.0/M_PI);

  m_degrees = m_degrees + 180.0;

  int m_pos = 0; // Possible positions: 0 - 1023
  double m_one_degree;

  // If it is in degree
  // Else -> degree shall be calculated first from the radian value
  m_one_degree = 1024.0 / 360.0; // Around 2.844...


  m_pos = (int)round(m_degrees*m_one_degree);

  return m_pos;
}
```

4.6 Basic logic

In the loop function we calculate all the tetas from the acquired values.

```
  // Teta calculation
  double m_teta1 = calc_teta1(m_x_pos, m_y_pos);
  double m_teta3 = calc_teta3(m_x_pos, m_y_pos, m_z_pos, m_teta1,
      m_phi);
  double m_teta2 = calc_teta2(m_z_pos, m_phi, m_teta3);
  double m_teta4 = calc_teta4(m_phi, m_teta2, m_teta3);
```

After that the software calculate the positions from the teta values.

```
  // Teta -> Position
  int m_pos1 = calc_position_from_teta(m_teta1);
  int m_pos2 = calc_position_from_teta(m_teta2);
  int m_pos3 = calc_position_from_teta(m_teta3);
  int m_pos4 = calc_position_from_teta(m_teta4);
```

If we get the right positions we need to set the servo motors to this position.

```
  #ifdef Arbotix
    // TODO -> linear movement with threads
    //unsigned int m_position[] = {4, m_pos1, m_pos2, m_pos3, m_pos4};
    //SetPosition(5, m_pincher_pos);
    //set_position_slowly(m_position);

    SetPosition(1, m_pos1);
    SetPosition(2, m_pos2);
    SetPosition(3, m_pos3);
    SetPosition(4, m_pos4);
   SetPosition(5, m_pincher_pos);
  #endif
```

Right now the robot in the required position so there is no further step if we don't want to run the program again. The last few row in the software is for the new run option.

```
// New run?
Serial.println("Send a random character if you want to set another
    position");
// Wait til the serial input
while(!Serial.available()){}
incomingByte = Serial.read();
Serial.flush();
delay(1000);
```

5. Test

Our first plan was to test all of the servos and the robot's possibilities. We reached that because we implemented a test software which can communicate with the laptop on serial.

The program firstly set all the servos to center and after that relax all of them.The software's input is the servo number and the requested position (from 0 to 1023) and after that the robot's servo motor set that parameter to the selected servo. With this we could test all the servos. The basic code is the following:

```
#include <Commander.h>
#include <ax12.h>
#include <BioloidController.h>

void setup() {

  //open serial port
   Serial.begin(9600);
   delay (500);
   Serial.println("###########################");
   Serial.println("Serial Communication Established.");

  // set everything to center
    Serial.println("###########################");
    Serial.println("Set all servo motor to center (512).");
    for (int i = 1; i <= 5; i++){
      SetPosition(i, 512);
      delay(100);
    }

    Serial.println("###########################");
    Serial.println("Relax all servo motor.");
    for (int i = 1; i <= 5; i++){
      Relax(i);
    }

}

void loop() {

  // read the servo motor number
    Serial.println("###########################");
    Serial.println("Which servo motor?");

    while(!Serial.available()){} // wait til the serial input
    char servoInByte = Serial.read();
    int servoNumber = servoInByte - '0';

    unsigned int positionValue=0; // Max value is 65535
    char incomingByte;

  // read position value
    Serial.println("###########################");
    Serial.println("Which position?");


    while(!Serial.available()){}
    delay(100);
    if (Serial.available() > 0) { // something came across serial
      positionValue = 0;       // throw away previous integerValue
      while(Serial.available() > 0) {
        incomingByte = Serial.read();

        // Serial.println(incomingByte); // just for test

        positionValue *= 10; // shift left 1 decimal place

        // convert ASCII to integer, add, and shift left 1 decimal
            place
        positionValue = ( (incomingByte - '0') + positionValue);
      }

      SetPosition(servoNumber,positionValue);
```

```
        Relax(servoNumber);

        Serial.print("Servo");
        Serial.print(servoNumber);
        Serial.print(" in Position: ");
        Serial.println(positionValue);
    }
}
```

6.
Summary

The first milestone of the project has been reached successfully, our main goal was getting to know this very complicated system and design the plans of its advanced control software. The biggest challenge is implementing the mathematical model of a three-dimension coordinate system on this manipulator. Fortunately, Arduino IDE is one of the simplest and easily usable development tools.

We think that our method is very effective with a very early constructed test for the software. This is not a pure example of TDD (Test Driven Development), but it has its own advantages too, and it is efficient enough for this project.

We started to discover a very interesting and very popular field of robotics. Robot manipulators are the most common robots all over the world. They are used in ever modern factory. They have the oldest origins, but with Industry 4.0 they have developed rapidly, and they have many more opportunities in the future. Robot manipulators ca be integrated into other types of robotic system, for example on the board of a mobile robot, or a manipulator can be part of a co-robot system.

6.1 Future plans

In the second part of the semester our goal will be to finish our planned work, implement the solution of the mathematical model, write the documentation, and specify system tests. Hopefully, after the execution of the tests we will report that everything is passed, and the robot manipulator will work safely.

We have further plans for the system and for other semesters, the movements of the robotic arm might be logged by other Arduino add-on devices, and this log files could be used for developing the software of the arm.

Bibliography

[1] Robotics,
https://www.techrepublic.com/article/
robots-of-death-robots-of-love-the-reality-of-android-soldiers-and-why-la

[2] Tom Harris
Ḧow Robots WorkḦow Stuff Works.

[3] Ax actuator,
http://support.robotis.com/en/techsupport_eng.htm#product/
dynamixel/ax_series/dxl_ax_actuator.htm

[4] Pincher Robot Arm,
https://www.trossenrobotics.com/p/PhantomX-Pincher-Robot-Arm.
aspx

[5] Daniel Jacobs,
Introduction to dynamixel Motor Control Using the ArbotiX-M Robocontroller

[6] Arduino IDE
https://en.wikipedia.org/wiki/Arduino_IDE

[7] Arduino IDE,
https://www.digikey.com/en/maker/blogs/2018/
introduction-to-the-arduino-ide

[8] Pincher,
https://www.trossenrobotics.com/Shared/Images/Product/
PhantomX-Pincher-Robot-Arm-Kit-Mark-II/pincher.jpg