Programming of robotic arm with computer vision based control

21. April 2022

Simon Goldhofer (s212252)

DANMARKS TEKNISKE UNIVERSITET

Table of contents

Introduction

KUKA iiwa lbr 14

Because this project was developed in part in collaboration with Novo Nordisk, one part of it involved development with the KUKA iiwa lbr 14 robot arm. The goal is to control the robot arm using the relative motion between your right hand and your right shoulder by using camera based pose estimation. Because this project involved proprietary hardware and software, the code is not public.

Custom robotic arm

The second part of this project is to develop a custom robotic arm solution, which can be further developed and used in later robotics classes at DTU. This involved development, manufacturing and testing of mechanical components, sensors and code. The goal is to produce a capable robot arm with a usable workspace size which can be later used in class to demonstrate the principles of coordinate transformation, forward and inverse kinematics. Furthermore, the gripper should be capable of picking up simple, lightweight objects to manipulate them. As this robot is open source, it is fully documented in this project paper and on the github page.

Documentation Practice

To improve usability and further development efforts of the software and the hardware developed, especially after this project, github has been used as extensive documentation platform. The github page[1] contains following information:

- Setup
- Theory (the content of this paper)
- Code Usage
- CAD (including stp files, drawings, stl files for 3D printing and a parts list)

Some parts of this paper are kept short and are further elaborated on the github page. Furthermore, better visualizations and videos are kept there to improve usability of the code and showcase its abilities. Direct references to the github repository are included as URL in the footnotes.

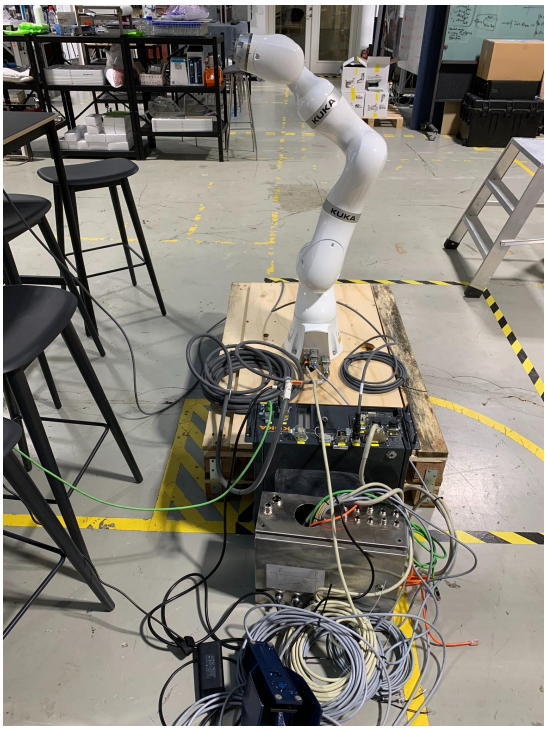
[1] `https://github.com/MonsisGit/MyRobot`

KUKA iiwa lbr 14 mechanical setup

Temporary Setup

In Figure 1a and 1b, a temporary setup with the arm mounted to two pallets can be seen. This was used because of the long lead time for the order of the table. Here, it was necessary to highly restrict the robots movements and accelerations to ensure a safe operation.

(a) closeup of robot arm with end-effector interface

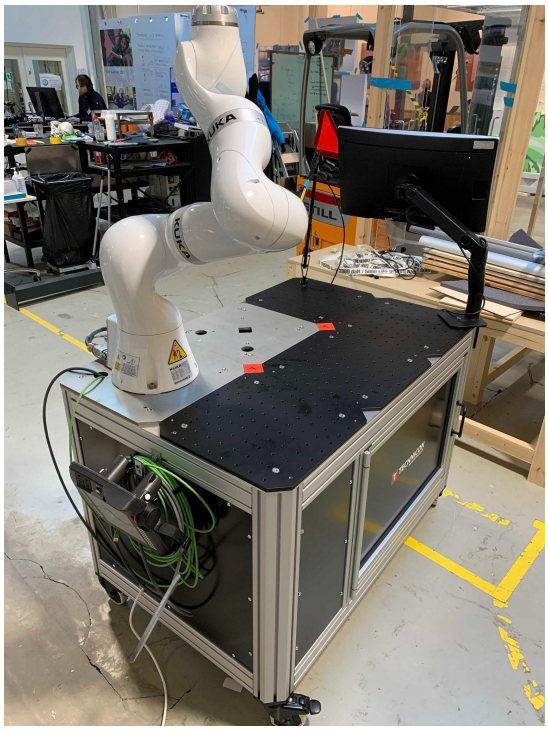


(b) side view

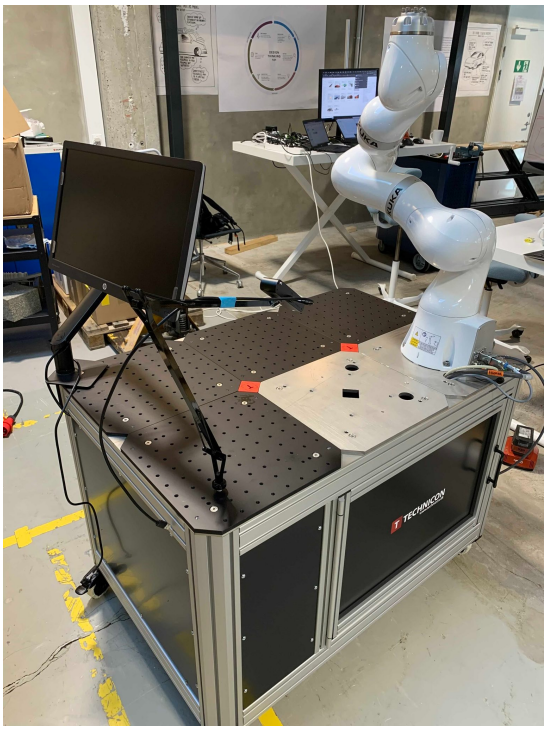
Figure 1: Temporary mechanical setup on two pallets

Final Setup

In Figure 2a and Figure 2b, the mechanical setup can be seen. The robot was setup without its clean room configuration. This means, a fan bracket mounted at the base of the robot, is missing. To provide stability and usability, the mounting platform is a robot table from technicon. An HMI has been mounted towards the front of the table, with the intention of displaying the pose detection results and user metrics. The ZED 2 Stereo camera[2] used was mounted next to the HMI, on a three-axis swiveling arm. The robots computer, power supply and connecting cables were stored inside the table. The green ethernet cable could be used to connect to the arms computer. Towards a later point in the project, a laptop and the nvidia jetson would also be stored inside the table and connected to the HMI and the arms computer.



(a) back view with teach pendant



(b) front view with HMI and ZED camera

Figure 2: Mechanical setup

[2] https://www.stereolabs.com/zed-2/

KUKA iiwa lbr 14 software setup

The software stack to control the robot arm consists of KUKA Sunrise proprietary software to run direct servoing, the KUKA iiwa stack, which runs a native ROSJava node on the robot as a Sunrise RoboticApplication and the usage of OpenPose software to run the pose detection.

KUKA Sunrise Software

The Sunrise software enables developers to install packages on the robot arm running on native Java. To run the iiwa stack, the direct servoing package had to be installed, which was kindly provided by KUKA Denmark as a 3 month trial version.

To run Sunrise and install software on the robot, these steps were followed:

- Use Laptop running Ubuntu (16 or higher)
- Change Ubuntu network ipv4 settings to IP: 172.31.1.151 with Netmask: 255.0.0.0
- Launch virtualbox, run Sunrise application
- In virtualbox network settings: / Network, set to Bridged Adapter enp0s31f6. Then go to Devices/ Network/ Connect Network Adapter
- In Windows Settings/Network Connections, set ipv4 settings to IP: 172.31.1.150 with Netmask: 255.0.0.0
- Plug Ethernet cable into robots 99X port
- Ping 172.31.1.147, should show reply
- Sunrise should be able to update robots software using StationSetup.cat/ Installation /Install. For relevant packages, see iiwa stack [3]
- For safety configuration see iiwa stack[4]. Additionally, a low end-effector force limit was included to increase the robots safety.
- Synchronize Sunrise Project

[3] https://github.com/IFL-CAMP/iiwa_stack/wiki/sunrise_project_setup
[4] https://github.com/IFL-CAMP/iiwa_stack/wiki#safety-configuration

IFL-CAMP iiwa stack

The iiwa stack enables IP/TCP communication to the robot using ROS. The robotic application installed on the robot is called ROSSmartServo. This application requires having a ROS Master running on a ROS machine connected to the robot.

Once both are started, the robot should connect to the ROS Master.The following topics are available to read the robots state:

- Position of the robot joints on /iiwa/state/JointPosition
- Torque values of the robot joints on /iiwa/state/JointTorque
- External torques applied to the robot on /iiwa/state/ExternalJointTorque
- Current velocities values of the robot joints on /iiwa/state/JointVelocity
- A combination of joints' positions and velocities on /iiwa/state/JointPositionVelocity
- ROS's JointState of the robot on /iiwa/state/jointstates
- Cartesian Pose of robot (or tool) TCP on /iiwa/state/CartesianPose
- Cartesian Wrench of robot (or tool) TCP on /iiwa/state/CartesianWrench
- Signal that the robot reached its commanded destination on /iiwa/state/DestinationReached

ROSSmartServo subscribes to the following topics for the following command modes:

- Position control in joint space on /iiwa/command/JointPosition
- Position control in cartesian space on /iiwa/command/CartesianPose
- Velocity control in joint space on /iiwa/command/JointVelocity
- Joint/Velocity commands on /iiwa/command/JointPositionVelocity the robot will reach the given destination at the given velocity

For further information, see the iiwa stack wiki[5].

[5] `https://github.com/IFL-CAMP/iiwa_stack/wiki`

OpenPose

OpenPose[6] has represented the first real-time multi-person system to jointly detect human body, hand, facial, and foot keypoints (in total 135 keypoints) on single images. It is used here, to detect the relative motion between to right arms wrist and its shoulder using a 2D image and its depth value using a stereo camera.

Even tough the software was successfully run and some simple algorithms were implemented, which could calculate the vector between shoulder and wrist, it was never integrated into the rest of the system and always run on a standalone computer. It proved too computationally intensive to run without optimizations on the nvidia jetson gpu.

Integration

The integration of all systems would be handled using a network switch, Figure 3 shows the layout and which machine runs which software.

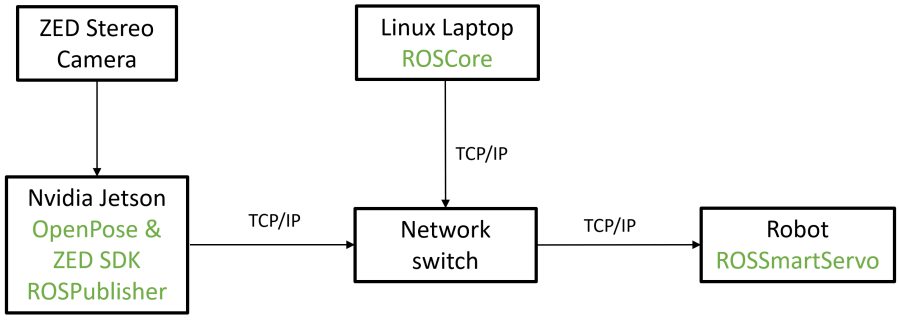


Figure 3: Communication between components

[6] https://github.com/CMU-Perceptual-Computing-Lab/openpose

Custom robotic arm mechanical design

CAD

The CAD model can be found on github [7]. This folder includes:

- The original CATIA V5 assembly
- The assembly in STEP format
- An overview drawing of the robot arm
- A complete parts list with assembly structure
- stl files for 3D printing

The parts list [8] (see table 1) includes part numbers referencing the CAD parts, their quantities and a comment on manufacturing or ISO numbers. The parts list is structured as assembly using subassemblies and the order in which to mount the parts. Using the drawings, CAD model and the structured parts list, it is possible to fully assembly a copy of the robot arm. The parts list furthermore contains comments on the parts such as manufacturing method.

Dynamixel AX-12A Motors

In the Robot arm, four identical Dynamixel AX-12A motors are used. Relevent specifications of the motors are:

Input Voltage (min,max) [V]	9-12
Stall Torque [Nm]	1.5
No Load Speed [rpm]	59
Resolution [deg/pulse]	0.2930
Weight [g]	55
Feedback	Position, Temperature, Load, Input Voltage
Dimensions (W,H,D) [mm]	32 X 50 X 40
Gear Ratio	22:31:00

The technical drawings in Figures 14, 15, 16 show relevant details needed for the modeling of the links and base in combination of the motors. More extensive details can be found on the manufacturers website. Robotis provides the Dynamixel SDK 4 to control the motors using various programming languages.

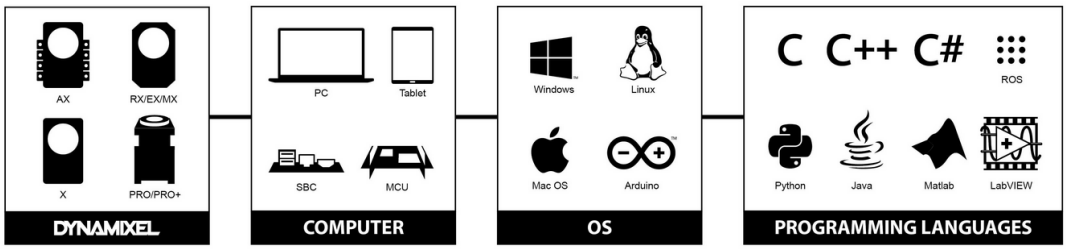


Figure 4: Dynamixel SDK [4]

[7] https://github.com/MonsisGit/MyRobot/tree/master/CAD
[8] https://github.com/MonsisGit/MyRobot/blob/master/CAD/parts_list.pdf

Workspace and strength capabilities

Figure 5 shows a 90° cutout of the current hardware configuration of the workspace of the robot. (This means the first link is still the old one with a3 = 96 mm). Because of the joint limitations[9], the robot cant reach close to its base, thus highly limiting the workspace.

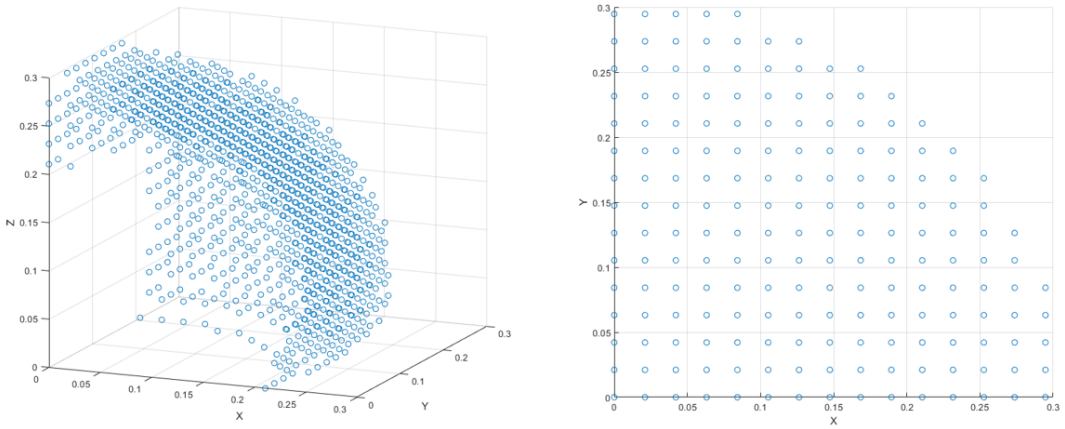


Figure 5: Cut out of the robots workspace (current hardware config)

In figure 6, the same workspace can be seen using the D.H. parameters from the theoretical configuration shown in CAD. Due to time constraints, this was not implemented in hardware. An increase in reach can be seen.

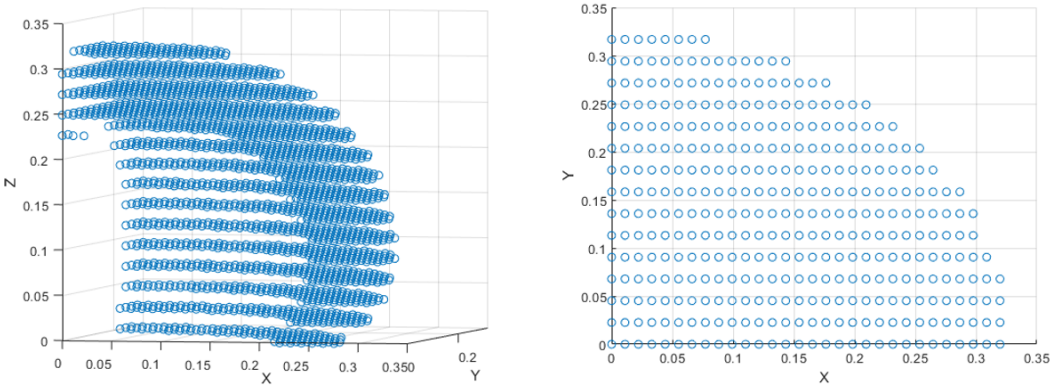


Figure 6: Cut out of the robots workspace

Due to these hardware constraints, the robots reachable space is quite limited and inverse kinematics tend to fail often for arbitrary values. The robots strength is determined using the stall torque of the second motor and the robot arms length at full extension. Here, dynamic forces and the robots own weight are neglected.

[9] In matlab, joint limits are currently set to [-130, 130; -180, 0; -100, 100; -100, 100];

$$\begin{aligned} L_{arm} &= 0.3821m \\ Torque_{stall} &= 1.5Nm \\ Force_{max} &= \frac{1.5Nm}{0.3821m} = 3.9257 \end{aligned} \tag{1}$$

This results in a lift capability of 0.4 kg at maximum extension.

Iterations

The robotic arm went through several iterations, which have not been fully documented. For fun, here are some pictures of legacy versions. The current one is not included, because it is fully documented on github.

Figure 7: Evolution of the robot arm

Custom robotic arm theory

For effective usage of the robot arm, the Denavit Hartenberg (D.H.) parameters need to be defined to figure out forward and inverse kinematics. The D.H. parameters are defined using following coordinate systems. The base is drawn in black. Joints are numbered 1-4. The base corresponds to the black coordinate system.

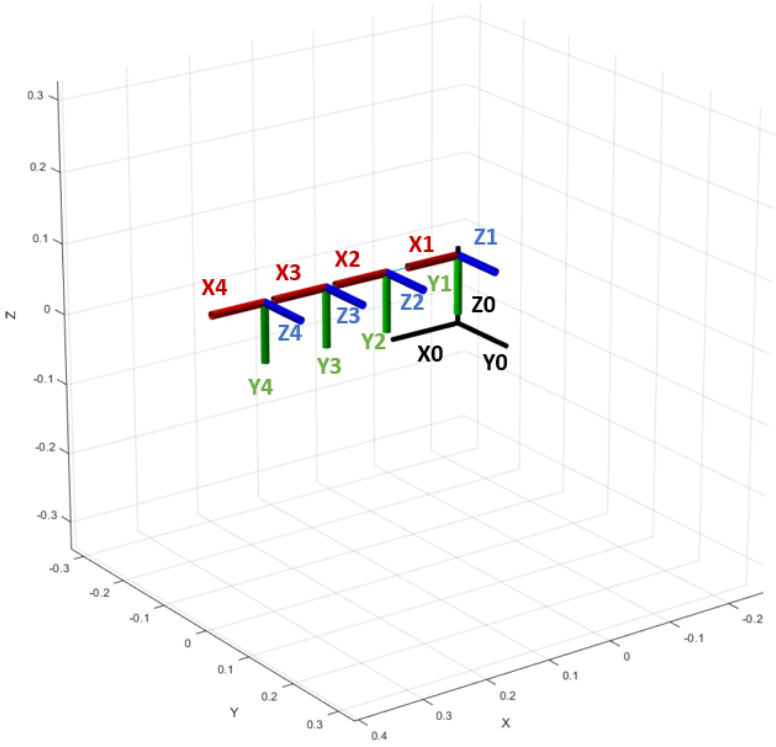


Figure 8: Configuration of the robot in home position

This results in following D.H. parameters:

$$D.H. = \begin{bmatrix} 0 & -pi/2 & 0.0955 & 0 \\ 0.116 & 0 & 0 & 0 \\ 0.096 & 0 & 0 & 0 \\ 0.09611 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

In matlab, the D.H. parameters are defined as:

```
% Denavit Hartenberg Parameters for Robot (a (link length),
% alpha (link twist), d (link offset), theta(joint angle))
dh = [0,-pi/2,0.0955,0;
    0.116,0,0,0;
    0.096,0,0,0;
    0.09611,0,0,0];
```

Remark: a_3 not being the same then a_2 is due to the current hardware configuration realized while writing this report. In the CAD model, a_3 is the same as a_2.

Forward Kinematics

Forward kinematics using the D.H. convention are calculated using following equations:

$$A_i = \text{Rot}_{z,\theta_i} \text{Trans}_{z,d_i} \text{Trans}_{x,a_i} \text{Rot}_{x,\alpha_i}$$

$$A_i = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$A_{14} = A_1 * A_2 * A_3 * A_4 \tag{4}$$

where:

θ_i : joint angle
d_i : link offset
a_i : link length
α_i : link twist

Forward kinematics are calculated in matlab using following code:

```
function ee_cartesian_coords = forward(self, j_a)

    %forward function for the MyRobot Class.
    %   Calculates forward transformation for all joint positions
    %   and end effector coordinates
    %
    %Inputs:
    %   j_a : a vector of four joint angles in [rad]
    %Outputs:
    %   ee_cartesian_coords : returns cartesian coordinates of end
    %   effector in the base coordinate system in [m]

    self.forward_transform = [cosd(j_a(1)) -sind(j_a(1))*cos(self.dh(1,2))  sind(j_a(1))*sin(self.dh
        sind(j_a(1)) cosd(j_a(1))*cos(self.dh(1,2)) -cosd(j_a(1))*sin(self.dh(1,2)) self.dh(1,1)*sin
        0 sin(self.dh(1,2)) cos(self.dh(1,2)) self.dh(1,3);
        0 0 0 1];

    self.joint_pos(:,1) = self.forward_transform * [0 0 0 1]' ;
    self.joint_pos(:,1) = self.joint_pos(:,1) / self.joint_pos(4,1);

    for i=2:length(j_a)
        self.forward_transform = self.forward_transform * [cosd(j_a(i)) -sind(j_a(i))*cos(self.dh(i,
            sind(j_a(i)) cosd(j_a(i))*cos(self.dh(i,2)) -cosd(j_a(i))*sin(self.dh(i,2)) self.dh(i,1)
            0 sin(self.dh(i,2)) cos(self.dh(i,2)) self.dh(i,3);
            0 0 0 1];
        self.joint_pos(:,i) = self.forward_transform * [0 0 0 1]' ;
        self.joint_pos(:,i) = self.joint_pos(:,i) / self.joint_pos(4,i);
```

```
    end

    ee_cartesian_coords = self.joint_pos(:,4);
end
```

For example, the forward transformation for the home position (joint angles all zero degree) results in:

$$f_{transform} = \begin{bmatrix} 1.0000 & 0 & 0 & 0.3081 \\ 0 & 0.0000 & 1.0000 & 0 \\ 0 & -1.0000 & 0.0000 & 0.0955 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} \tag{5}$$

with joint positions (colored coordinate systems 1-4 in figure 8) :

$$j_{position} = \begin{bmatrix} 0 & 0.1160 & 0.2120 & 0.3081 \\ 0 & 0 & 0 & 0 \\ 0.0955 & 0.0955 & 0.0955 & 0.0955 \\ 1.0000 & 1.0000 & 1.0000 & 1.0000 \end{bmatrix} \tag{6}$$

For example, we can see that the end-effector is in position x: 308,1mm, y: 0mm, z: 0.0995 mm.

Inverse Kinematics

The inverse kinematics can be numerically solved using matlabs robot toolbox and the following script:

```
if self.rbt == 0
    self.create_rbt();
end

j_a = zeros(4,1);
initialguess = self.rbt.homeConfiguration;
tform = [ 1 0 0 x;
    0 0 1 y;
    0 -1 0 z;
    0 0 0 1];

[configSoln,solnInfo] = self.ik('link4',tform,self.ik_weights,initialguess);
if strcmp(solnInfo.Status,'success')
    for i=1:4
        j_a(i) = configSoln(i).JointPosition*180/pi;
    end
elseif strcmp(solnInfo.Status,'best available')
    fprintf("Status: '%s', using Angles:\n", solnInfo.Status);
    for i=1:4
        j_a(i) = configSoln(i).JointPosition*180/pi;
        fprintf("%f°  ",configSoln(i).JointPosition*180/pi);
    end
else
    fprintf("Unsuccesfull IK, with status: '%s'", solnInfo.Status);
end



function create_rbt(self)
    self.rbt = rigidBodyTree;
    bodies = cell(4,1);
    joints = cell(4,1);
    for i = 1:4
        bodies{i} = rigidBody(['link' num2str(i)]);
        joints{i} = rigidBodyJoint(['jnt' num2str(i)],"revolute");
        joints{i}.PositionLimits = [self.joint_limits(i,1)*pi/180,self.joint_limits(i,2)*pi/180];
        setFixedTransform(joints{i},self.dh(i,:),"dh");
        self.ik = inverseKinematics('RigidBodyTree',self.rbt);
    end
end
```

Since this produces inconsistent results and is a slower, less transparent approach, an analytical approach was used, using geometrical interpretations of the robot arms movements. For joint 1, figure 9 shows the geometric interpretation of the robot arms position. The equation for θ_1 can be derived:

$$\theta_1 = \arctan(\frac{x_{current}}{y_{current}}) \tag{7}$$

for $(x_{current}, y_{current} \neq (0, 0)$ which is implemented in matlab with:

```
j1 = atan2(y,x);
```

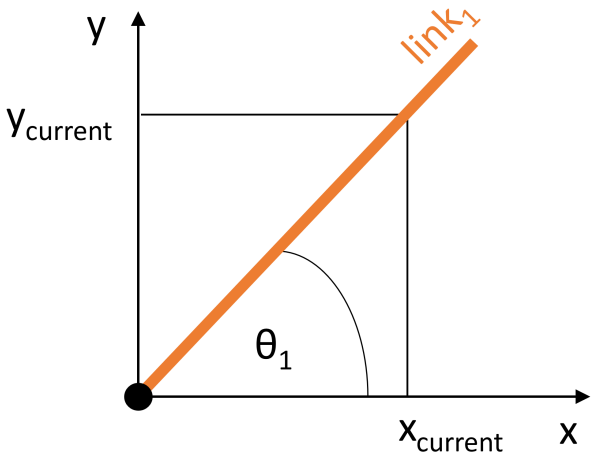


Figure 9: Geometric interpretation for joint 1

To figure out θ_2 and θ_3, figure 10 shows the geometric interpretation. Following equations can be used to derive the joint angles:

$$L_{arm} = \sqrt{x_{current}^2 + y_{current}^2} \tag{8}$$

$$s_z = z_{current} - d_1 \tag{9}$$

from the law of cosines:

$$\cos\theta_3 = \frac{L_{arm} + s_z - a_2^2 - a_3^2}{2 * a_2 * a_3} \tag{10}$$

$$\theta_3 = \arccos\frac{L_{arm} + s_z - a_2^2 - a_3^2}{2 * a_2 * a_3} \tag{11}$$

We can find two solutions for elbow-up/ elbow-down:

$$\theta_3 = \text{Atan2}\left(\cos\theta_3, \pm\sqrt{1 - \cos\theta_3{}^2}\right) \tag{12}$$

Now θ_2 can be found with:

$$\theta_2 = \text{Atan2}(L_{arm}, s_z) - \text{Atan2}\left(a_2 + a_3 * \cos\theta_3, a_3\sin\theta_3\right) \tag{13}$$

which is implemented in matlab with:

```
j3 = acos( ((sqrt(x^2+y^2)-self.dh(4,1)*cos(pitch))^2 + (self.dh(1,3)-z)^2 - self.dh(2,1)^2 ...
- self.dh(3,1)^2) / (2*self.dh(2,1)*self.dh(3,1)) );
```

```
j2 = -atan2(z-self.dh(1,3)-self.dh(4,1)*sin(pitch),sqrt(x^2+y^2)-self.dh(4,1)*cos(pitch)) ...
- atan2(self.dh(3,1)+self.dh(2,1)*cos(j3),self.dh(2,1)*sin(j3)) + (pi/2-j3);
```

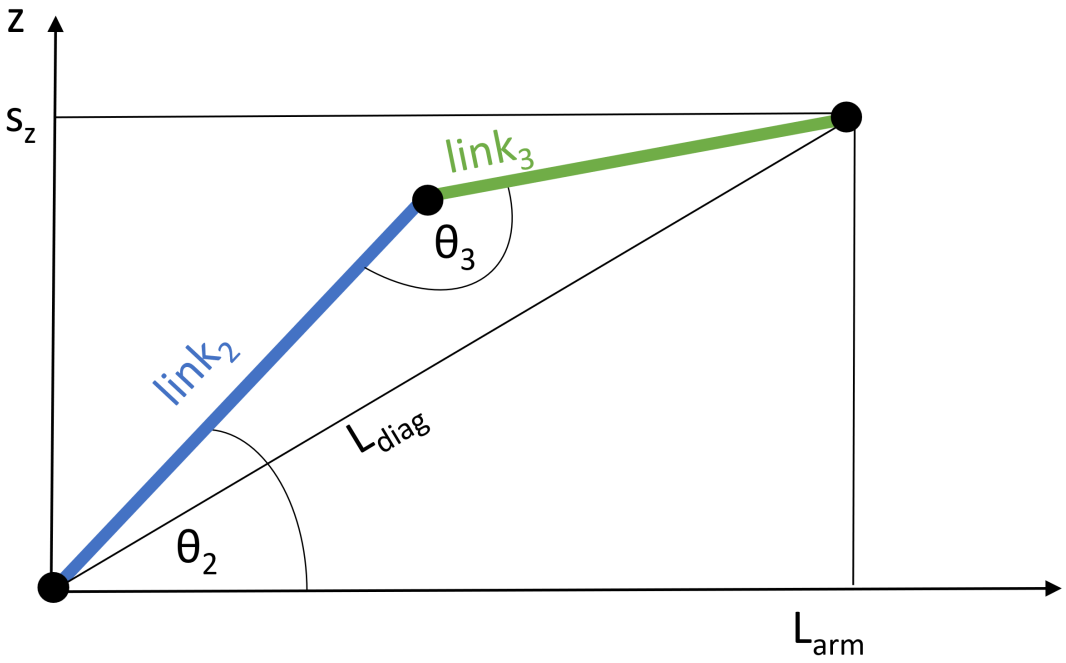


Figure 10: Geometric interpretation for joint 2 and joint 3

To figure out θ_4, we need to take the pitch of the end-effector into consideration, which is its angle to the horizontal plane, as can be seen in figure 11. θ_4 is calculated in dependence of the pitch angle, which is a user input:

$$\theta_4 = pitch - \theta_2 - \theta_3 \tag{14}$$

which is implemented in matlab with:

```
j4 = pitch - j2 - j3;
```

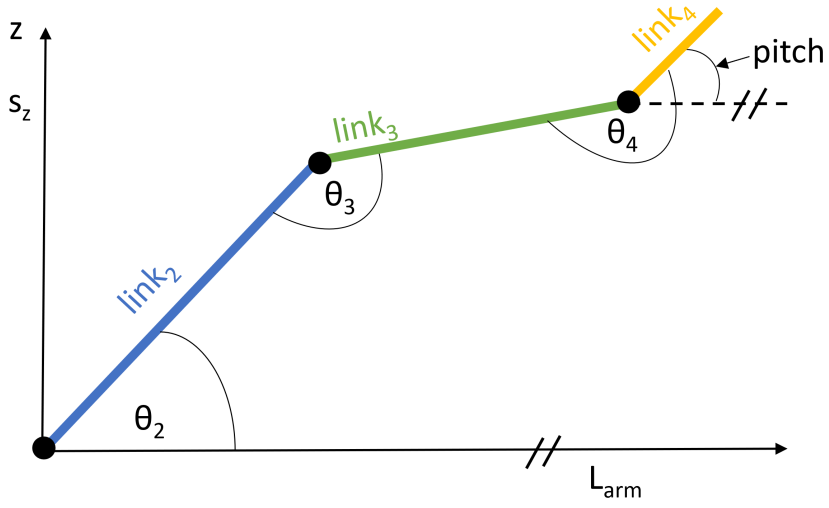


Figure 11: Geometric interpretation for joint 4

A check if the joint angles contain complex numbers can catch impossible configurations of the robot arm:

```
assert(isreal(j_a),"Configuration Impossible");
```

Custom robotic arm software design

MyRobot Class

The MyRobot[10] class in matlab was created to control the robots functionalities and also to track the robots current parameters like joint angles, joint positions, speed, torque etc. For setup and documentation, see the github[11]. More detailed documentation for the individual methods can be found by typing into the matlab command window

```
doc MyRobot
```

which will open the matlab html documentation. Example usage of the class could look like:

```
% Initialize the robot, it should move to home configuration
% (0°,0°,0°,0°)
robot = MyRobot();
%
% Set movements speed of each individual joint, update interal joint
% speeds for later commands
robot.set_speed([0.1,0.1,0.1,0.2],true);
%
% Set all motors to maximum torque
robot.set_torque_limit([1,1,1,1]);
%
% Draw the current configuration of the robot
robot.draw_robot();
%
% Move the robots joints
robot.move_j(20,-90,0,50);
%
% Actuate the gripper. If the gripper is currently closed, it will open
robot.actuate_gripper();
%
% Get the robots current joint positions
current_joint_positions = robot.joint_pos
%
% Disable all motors. This is necessary to free up the com port. If you
% forgot to do this and clear the robot object, it will fail at
% reinitialization. To fix this unplug the robots USB cable and clear
% the workspace
robot.disable_motors();
```

[10] https://github.com/MonsisGit/MyRobot/blob/master/matlab/MyRobot.m
[11] https://github.com/MonsisGit/MyRobot/blob/master/README.md#Basic-Usage-of-MyRobot

Graphical User Interface

A graphical user interface has been created to ease the interaction with the robot. It can be installed using the mlappinstall[12] file on github or simply run using the mlapp[13] file. The interface is shown in figure 12 and further detailed on github[14]. The GUI accesses the internal state of the robot, thus updating its status like joint angles or end-effector position with every robot interaction. Buttons like *Open Gripper* change to *Close Gripper* once the gripper is open or *Show Configuration* to *Hide Configuration*. The status button turns green once communication with the arm is sucesfully established. A whole robot program can be created by using the *Record Configuration, Del Last Configuration, Play Configuration History* buttons. Using this, the robot records all internal states like speeds, positions, gripper status or torque and plays it back step by step.

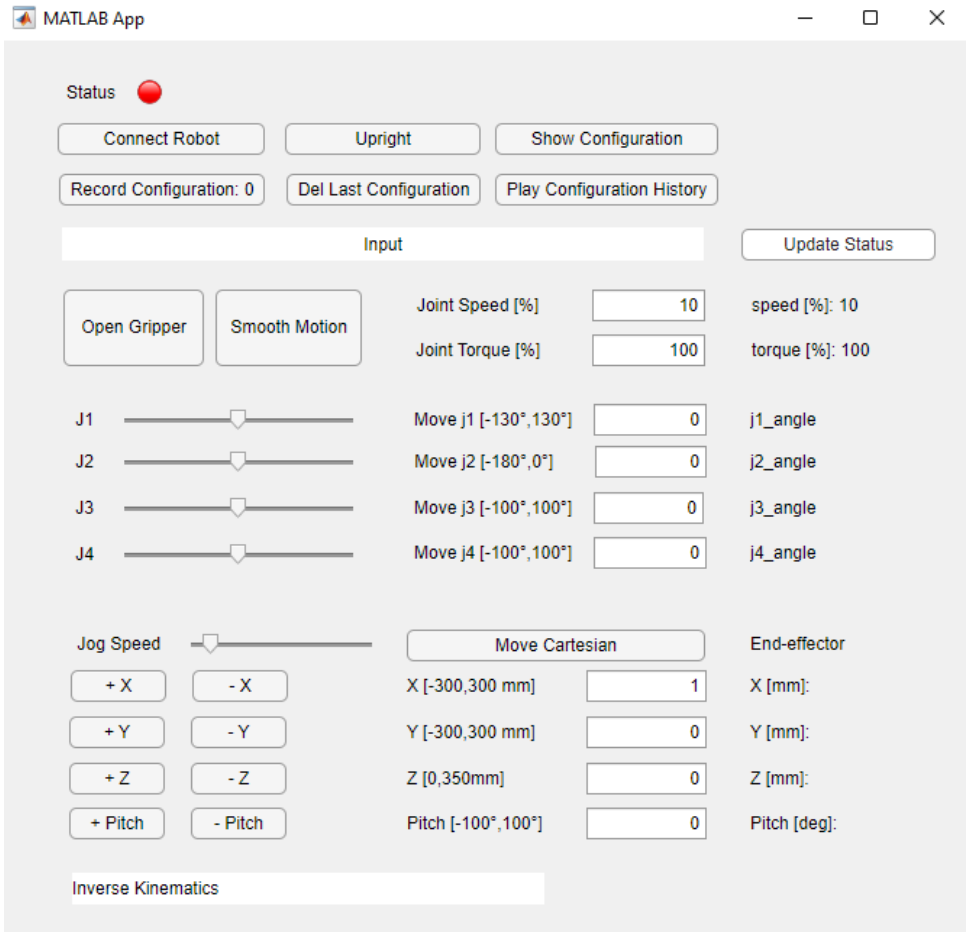


Figure 12: GUI for the MyRobot class

[12] https://github.com/MonsisGit/MyRobot/blob/master/matlab/MyRobot_Studio.mlappinstall
[13] https://github.com/MonsisGit/MyRobot/blob/master/matlab/MyRobot_Studio.mlapp
[14] https://github.com/MonsisGit/MyRobot/blob/master/README.md#Using-the-GUI

Custom robotic arm visual servoing

A simple visual servoing demo[15] was implemented as a showcase of the MyRobot class. A logitech C270 camera was mounted on the robots end effector with the purpose to detect a persons face using MATLABs inbuilt cascade detector. Some safety features are implemented, such as a region of interest, in which the face needs to be located and a minimum size for the detected region of the face. Once a face is detected, the middle of the resulting bounding box is calculated. The robot adjusts its joints to keep the persons face in the center of the camera, thus following your movements, as seen in figure 13. The red line visualizes the offset between the persons face and the center of the camera.

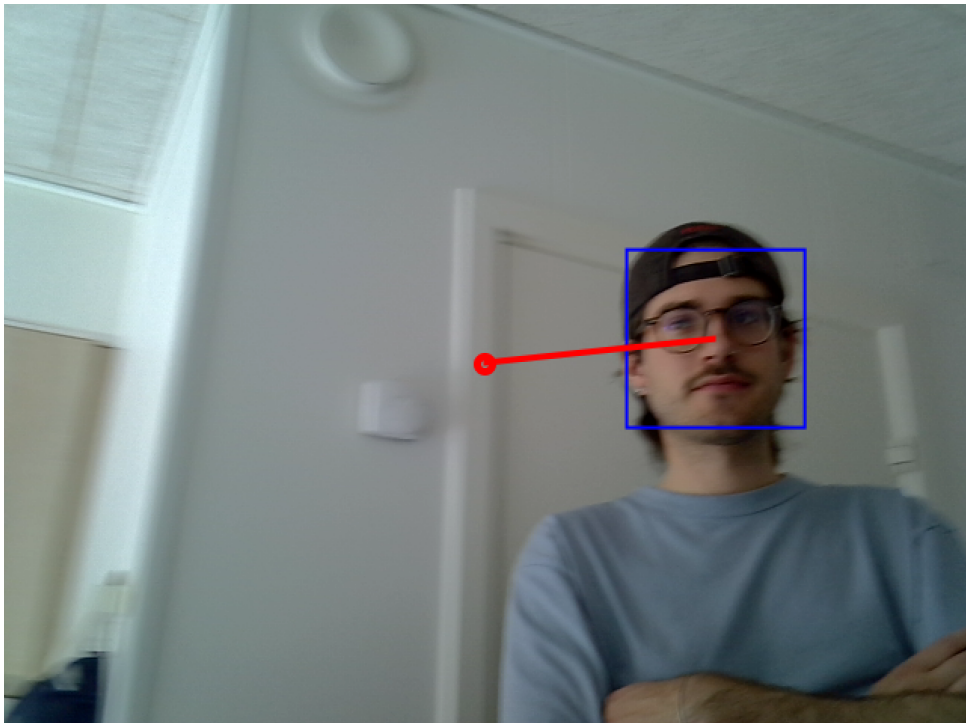


Figure 13: Example for face tracking

[15] https://github.com/MonsisGit/MyRobot/blob/master/README.md#Visual-Servoing

Evaluation of KUKA iiwa and custom robotic arm projects

Custom robotic arm

Developing a robotic arm poses numerous mechanical, electro-mechanical and software challenges. The scope of the project was thus reduced to developing a four-axis arm with a simplified mechanical setup, which compromised on various things such as mechanical factors like strength of the arm, precision or service-life. Software compromises included the necessity to use matlab to provide accessibility for students which slows down code execution or to write simplified software, which relies on the motors inbuilt motor controllers and communication protocols rather then writing custom, more tune able software.

The mechanical setup is mostly 3D-printed and uses M2 and M3 screws, which provide the possibility for fast iterations, but are not suitable to built a lot of these arms. Using standard robotis parts as links and using the same nuts and bolts everywhere could be a future improvement. Using the motors attached gear box as joint axis puts cyclic strain on their plastic gearbox, which drastically reduces service-life. Due to the usage of the same motors everywhere, the strength to weight ratio and speed of the arm is limited. The gripper uses a bowden-style setup for weight saving, which needs to be further iterated on, as its repeatability and strength varies. The gripper opening relies on tensioned rubber bands, which tend to wear and do not produce repeatable motion. The base needs to be mounted to a heavier base plate to avoid tipping over the arm during high acceleration movements. The motor cables are currently not protected and hanging on the outside of the arm, prone to tangling and damage. Due to vibrations and jerking movements, all the robots screw connections tend to loosen up.

The robots software provides a class to control the robot and monitor its state. Due to it being implemented in matlab, accessibility is limited. Ports to Python3 and C++ and ROS compatibility could improve upon that. The robots GUI provides a nice starting point in using and programming the robot. The GUIs inverse kinematic jog functions need to be improved, they work unreliably.

KUKA iiwa robotic arm

Using the KUKA iiwa 14 proved to be harder then expected. Due to the scope of the project beeing too large, the part of actually connecting the KUKA via ROS to the NVIDIA Jetson NANO and controlling it via pose estimation could not be finished.

The mechanical setup of the robot required a stable base, for which a robot table needed to be purchased. Due to the strength and speed of the robot, the user needs to configure safety precautions around the robot, which can be done using the KUKA sunrise software. Integrating the robot into ROS required the installation of licensed KUKA software, which was at the time only available as a free trail and thus makes the project obsolete once the license expires. Setting up the robot via the sunrise software was over all more difficult and worse documented then expected. The robots target group are professional industry applications, not hackers trying to use the robots for research purposes. Thus the overall accessibility of the software, drivers, documentation and help is limited.

Strong points of the robot are its extreme precision and speed, its strength, it being waterproof and its clean-room capability. The 7-axis configuration provides excellent workspace coverage, and its wrist and adapter flexibility for the end-effector. The teach pendant makes operation and supervision of the robot easy. All this comes with a high price to purchase and expensive licensing costs.

Possible improvements custom robotic arm

Software

- Improve the calculation of the inverse kinematics by catching certain cases which cause non-valid results or singularities. This has been only partly done.
- In case of elbow-up/ elbow-down solutions in the inverse kinematics, finding the angle with the least residual rotation compared to the current position.
- Solve the blocked port problem when robot is not disconnected properly.
- Implement manipulator jacobian and trajectory planning.

Hardware

- Install a bearing in the robots base above the first motor.
- Improve gripper by mechanically linking the two fingers, for example with gears. The current gripper has trouble with smooth, simultaneous closing and opening. Once the tensions on the bowden cable is alleviated by closing one side of the gripper, the other side sometimes only closes partially.
- Select motors based on torque requirements, thus saving weight, money and space. Consider using two motors to actuate the second joint.
- Select and install robot on a stable base.
- Further iterations on the robot arm could include moving the motors for joint 3 and joint 4 one link length down to joint 2 and joint 3. These motors would work using pulleys or links which reach through the link to actuate the actual joint, thus decreasing the leverage on the motors weight. A good showcase of this is the ABB irb 460 robot[16].

Conclusion

Even tough the KUKA project could not be finished successfully, the hardware integration was fully finished. Pose estimation was successfully implemented and some ROS functionality tested. The robot arm could be remote controlled through ROS messages and its current state could be read out. Because the robot arm is not intended to be used this way, a lot of work went into setup and integration issues, which could have been avoided with a different choice of hardware and software components.

The custom robotic arm project also had a large scope. The hardware underwent multiple iterations. Even after considerable effort, it still shows plenty of points for improvement, most notably the end effector. The software could be developed further and is in a usable, well documented state. The graphical user interface works very well. The inverse kinematics can be at times tricky, since the workspace is so limited. The visual servoing shows a nice use-case for the MyRobot class using MATLAB.

[16] https://new.abb.com/products/robotics/industrial-robots/irb-460

Bibliography

[1] URL: http://en.robotis.com/shop_en/item.php?it_id=902-0003-001.

[2] *FP04-F2 10pcs gt; frames - ROBOTIS premium*. URL: http://en.robotis.com/shop_en/item.php?it_id=903-0036-001.

[3] *FP04-F3 10pcs gt; frames - ROBOTIS premium*. URL: http://en.robotis.com/shop_en/item.php?it_id=903-0037-001.

[4] Robotis. *Dynamixel SDK*. URL: https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/overview/.

Appendix

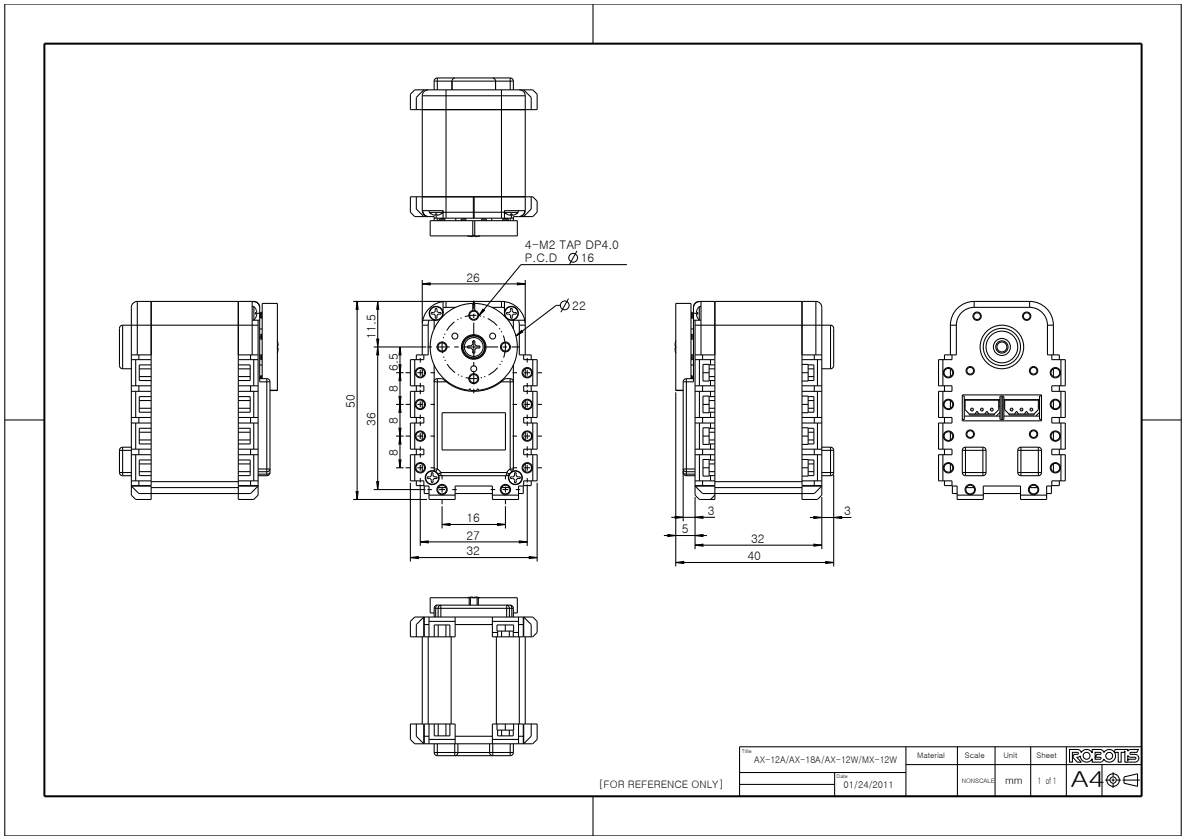


Figure 14: Dynamixel AX-12A motor [1]

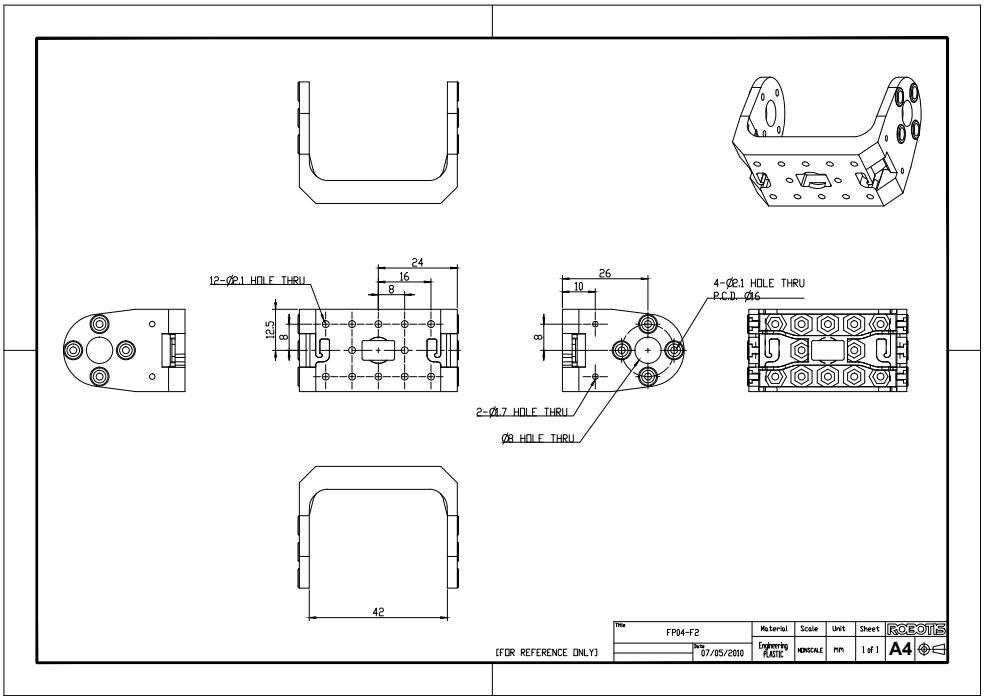


Figure 15: Flange 1 (FP04-F2) [2]

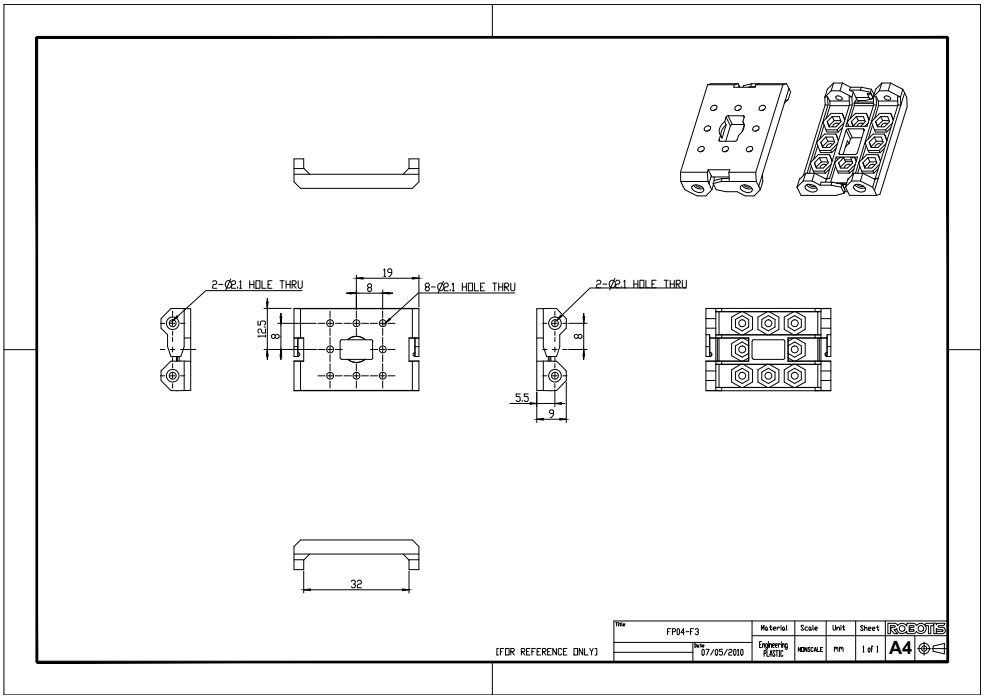


Figure 16: Flange 2 (FP04-F3) [3]

Table 1: Parts List

Part Number			Quantity	Name	Comment
0			1	Robot Arm	
	010		1	Base Assembly	
		011	1	Base Wall	FDM 3D-Print
		012	1	Base Cover	FDM 3D-Print
		013	1	Base Bracket	FDM 3D-Print
		014	1	Bowden Pulley	SLA 3D-Print
		015	6	ISO 10642 M4x16	Countersunk
		016	12	ISO 7046 M2x6	Robotis Part
		017	2	ISO 4762 M3x16	
		018	2	ISO 4032 NUT M3	
		019	8	ISO 4032 NUT M2	Robotis Part
		040	2	Dynamixel AX-12A	Robotis Part
		070	1	Zip Tie	
		071	2	Robot Cable-3P 200mm	Robotis Part
		072	1	Usb2dynamixel	Robotis Part
		073	1	Power Supply 12V, 2A	
		074	1	USB Extension Cable	
		075	5ml	Loctite	Apply to 016,024
	020		1	Bottom Link Assembly	
		021	1	Link	SLA 3D-Print
		022	1	Washer	SLA 3D-Print
		023	1	FP04-F3	Robotis Part
		024	1	ISO 4762 M3x12	Robotis Part
		016	16	ISO 7046 M2x6	Robotis Part
		071	2	Robot Cable-3P 200mm	Robotis Part
		075	5ml	Loctite	Apply to 016,024
		040	1	Dynamixel AX-12A	Robotis Part
	030		1	Top Link Assembly	
		021	1	Link	SLA 3D-Print
		022	1	Washer	SLA 3D-Print
		040	2	Dynamixel AX-12A	Robotis Part
		016	8	ISO 7046 M2x6	
		024	1	ISO 4762 M3x12	
		071	1	Robot Cable-3P 200mm	Robotis Part
		075	5ml	Loctite	Apply to 016,024
	050		1	Gripper Assembly	
		051	1	Bowden Tube	length ca. 500m, d:2,4mm
		052	1	Gripper Base Plate	SLA 3D-Print
		053	1	Bowden Tube Clamp	SLA 3D-Print
		054	2	Gripper Fingers	SLA 3D-Print
		055	2	Gripper Tips	SLA 3D-Print
		056	1	Bowden Roller	SLA 3D-Print
		057	1	FP04-F2	Robotis Part
		058	1	AX123_Bushing-F (BPF-BU)	Robotis Part
		059	2	ISO 4762 M3x20	
		060	1	ISO 10642 M3x25	
		061	2	ISO 10642 M3x30	
		062	7	NUT M3 Thread Locking	
		063	1	Bowden Cable	length ca. 500m, outside d:2mm
		064	2	Rubber Bands	length ca. 200mm
		065	3	Cable Crimp	endstop for bowden cable

bibliography.bib