Implementation of the Arrowhead Framework in Autonomous Factory Robots

As a fulfilment to the course D7039E & E7032E. Lecturer: Jan van Deventer.

Martin Blaszczyk, Edward Cedgård, Niklas Dahlquist, Edward Källstedt, Albin Martinsson, Måns Norell
Computer Science, Electrical and Space Engineering Dept.
Luleå University of Technology
Luleå, Sweden
{marbla-6, edwced-4, nikdah-6, edwkll-7, mnsnor-5, albmar-6}@student.ltu.se

***Abstract*—The main objective of this work is to evaluate the integration of the Arrowhead framework with an autonomous factory robot designed to perform certain tasks. By utilizing Arrowhead it is possible to guarantee security, identifiability and safety to the factory floor. With colored lines and a robot implementing a line following algorithm together with QR code scanning, it is possible to have a flexible and cheap system of navigating a factory floor. On the robot itself the Robotic Operating System (ROS) was used while the communication with the mission planner utilizes the Arrowhead framework. The final product is able to autonomously navigate the factory floor and perform pick up and drop off tasks given by the mission planner.**

***Keywords*-Arrowhead, Machine Vision, Line Following, Differential Drive, Robotic Arm**

I. Introduction

A. Background and Motivation

Arrowhead is an initiative from Luleå University of Technology to create a unifying framework that can enable embedded devices to integrate and interoperate services in an open network environment. This framework and its approach will strongly contribute to the reduction of design and engineering efforts in the industry.

As part of the Engineering Project course for Master's students in Computer Science, Control and Electrical Engineering at Luleå University of Technology the goal was to implement and demonstrate how the Arrowhead Framework can be used in a factory setting using autonomous ground robots. This project report aims to provide an overview of the thought process, workflows, and results of the project.

The proposed solution is a robot which by implementing machine vision algorithms can navigate the factory floor while also being able to pick up objects using its arm and gripper. It utilizes the Arrowhead framework to integrate with the other parts of the model factory.

B. Contributions

Our proposed solution is an example of how the Arrowhead framework is utilized in a model factory together with a ground robot. The modular design of the robot makes it easy for future design improvements and implementations of third-party open-source solutions. Secondly, we propose a solution consisting of colored lines and QR codes and how the robot can follow a path using an RGB camera instead of the conventional sensors such as IR sensors. With this camera solution the possibilities of navigating the factory floor increase compared to the basic conventional line following.

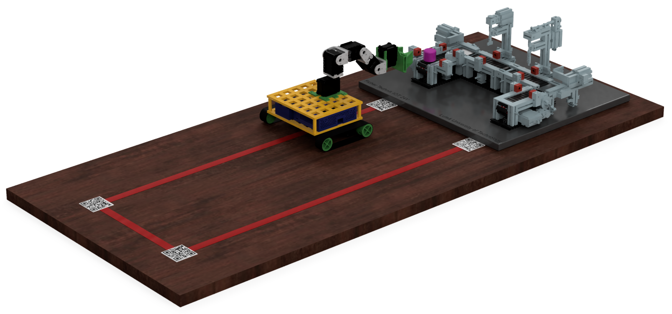


Fig. 1. Rendering of concept design and engineering challenge.

Structure

The rest of this report is structured as follows. In Section II, the overall challenge is described. In Section III covers the conceptual design of the robot followed by Section IV explaining the machine vision algorithms. Section V covers the theoretical modeling of the robotic arm as well as the modeling of the movable base. In Section VI the interested reader can find how to communicate with Smart Servo motors using Half Duplex circuitry. The last two sections V-O and VII explain the various motions and final conclusions respectively.

II. Engineering challenge

To evaluate the performance of the robot a test setup was assembled, shown in Fig. 1. The setup consists of the main factory model and red lines connecting the drop-off point to

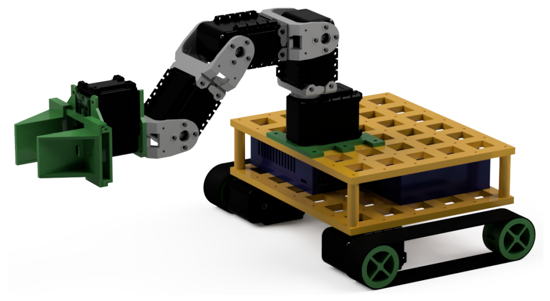


Fig. 2. Rendering of concept design.

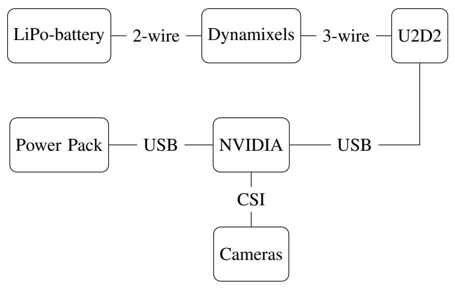


Fig. 3. Diagram of electrical connections

the left and the pickup point to the right. The red lines specify the path the robot can take. At each endpoint and corner of the colored path a QR code is placed which contains a coordinate of that point in space. The goal for the robot is to receive an instruction from Arrowhead, either to pick up the piece from the factory or drop it off at the drop off area. The robot would then navigate autonomously to the correct coordinate and perform the pickup or drop off action with the arm.

III. Concept

A. Mechanical structure

The design of the robot is based on the LEGO®Mindstorms®EV3 set. While the original set has a movable base without a manipulator arm, it also does not incorporate the Dynamixel AX-12A Smart Servos used in the final design. Thus the decision to redesign the robot with inspiration from the LEGO®EV3 design was made. By developing and designing a new platform it gave the possibility to adapt the dimensions and fastenings. The base was assembled using 3D-printed and laser cut parts. The final design is shown in Fig. 2 Consisting of seven Dynamixel®AX-12A Smart Servos for actuation; two on the base for locomotion, four on the arm and one for the gripping tool giving the robot seven degrees of freedom. The main advantage of using the AX-12As is the possibility of connecting them in series which enables parallel control of all the joints. Additionally with the built in sensors the motors return feedback of the joint angles, angular speed, current draw and temperature to name a few.

B. Electrical components

The internals in the Dynamixel®AX-12A Smart Servo motors give feedback on the state of the motors, the 1000mAh LiPo battery supplies power to the motors and surrounding electronics. For the computations the NVIDIA®Jetson Nano is used as it runs Ubuntu natively which makes the implementation of the underlying Robotic Operating System (ROS) [1] faster. In order to power the NVIDIA Jetson Nano a separate power bank is mounted on the robot. A diagram of the electrical components can be seen in Fig. 3.

C. Robotic Operating System

Because the robot and the task consists of many different parts it is preferred to utilize an easy to use and well documented framework for internal communication on the robot. Due to its versatility and popularity within the research community this project implements the Robotic Operating System (ROS) [1] as its underlying framework. ROS utilizes an internal TCP communication protocol where different parts can communicate and be synchronized which enables real-time performance. The rosgraph describing the internal connections between the nodes is shown in Fig. 4.

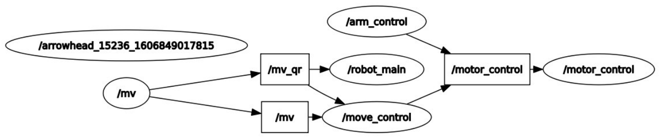


Fig. 4. Rosgraph of the robot.

IV. Machine Vision

In order for the robot to navigate the factory floor in a reliable and cost-effective manner a system using lines and QR codes is proposed. Ideally, the robot should be able to travel between points in an arbitrary grid. A grid is made up of multiple connected colored lines. At every intersection a QR code containing a coordinate is placed.

To facilitate this, a machine vision system using an RGB camera is used. The machine vision system consists of two major components. One component is a line following algorithm which can take an image of a colored line and produce an angle between the center of this line and the base of the robot. This angle is then sent to the motor controller. The second component is a QR code reader, this makes it possible for the robot to behave in different ways depending on the contents of the approached QR code. These components are used in conjunction to allow the robot to travel between the points on the grid.

A. Line Following Algorithm

The camera is placed in a top down configuration. This makes it possible for the camera output to be regarded as a 2D representation of the grid. The process of producing an angle from an input image of a colored line is divided into four different steps which can be summarized as follows:

1) Crop out a horizontal slice from the image.
2) Create a color mask to extract the colored line.
3) Determine the center point of line using image moment.
4) Calculate angle between center point and robot base.

First a horizontal slice of the captured image is cropped out at a fixed vertical offset, y_0. This slice is visualized by the two horizontal red lines in Fig. 5. All remaining steps are done on this slice.

The image is then converted to HSV color space and a predetermined color mask is applied to extract only the pixels with the same color as the followed line. This step produces a new binary image where each pixel is represented with either a 1 or 0, depending on if the original pixel was inside the predetermined color range or not. A visualization of this binary image is seen in the rightmost part of Fig. 5. This binary image is then used as the input for the next step.

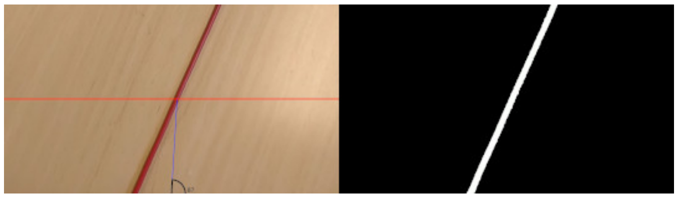


Fig. 5. Visualization of line following algorithm

To find the centroid of this binary image the *moment* is calculated. The image moment is in some sense analogous to the concept of *center of mass* used in physics. It is defined as

$$m_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q I(x,y)\mathrm{d}x\ \mathrm{d}y \tag{1}$$

where $I(x,y)$ denotes the intensities of the image [2]. This formula is discretized in order to be usable with an array of pixels

$$M_{pq} = \sum_x \sum_y x^p y^q I(x,y). \tag{2}$$

The center of the image in the horizontal direction is then given by

$$\bar{x} = \frac{m_{10}}{m_{00}}. \tag{3}$$

All that remains to do is to calculate the angle between the horizontal axis and the line drawn between the point $(\bar{x}, y_0)$ and the base of the robot, denoted as (x_r, y_r). The angle α is given by

$$\alpha = \tan^{-1}\left(\frac{y_0 - y_r}{\bar{x} - x_r}\right). \tag{4}$$

The angle output is visualized in Fig. 5. This process is repeated for every captured frame and will produce a continuous stream of angles which can then be used by other components of the robot.

B. QR Code Reader

In order to provide the robot with additional positional information QR codes are used. The basic QR code functionality is provided with the use of the *ZBar* [3] library.

Every frame is processed and scanned for QR codes. If a QR code is found in the input image the contents of the QR code, the position of the QR code and the cardinal direction from which the QR code is approached is sent as a message to be received by the other components of the robot.

C. Pathfinding

To autonomously navigate between different points a pathfinding system is used. Each QR code contains information of the form $\{x, y\}$, where x and y are the Cartesian coordinates of the QR code in the grid. An internal representation of the entire grid is stored in the form of an adjacency matrix. Dijkstra's algorithm [4] is used to find the shortest path between two different coordinates in the grid.

Calculating which direction the robot must turn to reach the next coordinate in the path is done using the relative position of the current and next QR code together with the direction from which the robot is approaching the current QR code. First the angle between the two adjacent QR codes relative the horizontal axis of the grid is calculated. Then the angle from which the robot is approaching the QR code is added to the angle between the QR codes. This sum then represents the angle which the robot must turn to align itself to the next QR code in the path.

Once the robot is aligned a new line following procedure is started until the robot reaches the next QR code. This process is then repeated until the destination is reached.

V. Modeling

Modeling means finding a mathematical representation of a physical system, kinematics and/or dynamical properties. This model is used in order to manipulate the actuators of the robot to their desired position or action.

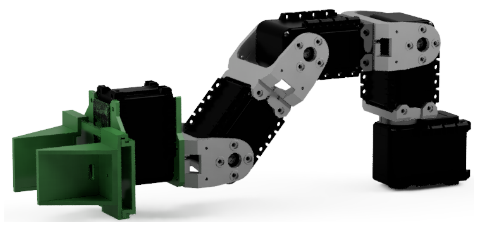


Fig. 6. The robotic arm used.

A. Rigid body transformation

Kinematics is what describes the motion of rigid bodies and points in space. In general any position can be described by translation along three axes and rotation along these same axes. These translations/rotations can be described by a matrix

$$T = \begin{bmatrix} R & d \\ \mathbf{0} & 1 \end{bmatrix} \tag{5}$$

where R is a 3 x 3 rotation matrix and d a 3 x 1 translation matrix. This implies that any transformation could be characterized by six parameters, three for the translation and three for the rotation of each joint. [5]

B. Denavit-Hartenberg Convention

A common approach for selecting the coordinate frames of reference for each joint in a robotic arm is the Denavit-Hartenberg convention. This allows the transformation matrix for each joint to be expressed as

$$A_i = Rot_z(\theta_i) \cdot Trans_z(d_i) \cdot Trans_x(a_i) \cdot Rot_z(\alpha_i) \quad (6)$$

that consists of the four basic transformations

$$Rot_z(\theta_i) = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & 0 \\ sin(\theta_i) & cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

$$Trans_z(d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8)$$

$$Trans_x(\alpha_i) = \begin{bmatrix} 1 & 0 & 0 & \alpha_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (9)$$

$$Rot_z(\theta_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha_i) & -sin(\alpha_i) & 0 \\ 0 & sin(\alpha_i) & cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (10)$$

The parameters θ_i, a_i, d_i and α_i, known as DH-parameters, characterize each joint. To allow the transformation for each joint to be represented by only four parameters, compared to the six parameters that is required in general, there are some restrictions on how the coordinate frames can be chosen. To comply with the DH-convention, the following must be satisfied.

- The axis x_1 is perpendicular to the axis z_0
- The axis x_1 intersects the axis z_0

where it is assumed that two frames are given, frame 0 and frame 1, and the transformation from equation (6) transforms a coordinate from frame 1 into a coordinate in frame 0. [5]

In Table I the DH-parameters that was used for the robotic manipulator, seen in Fig. 6, are listed.

TABLE I
THE DENAVIT-HARTENBERG PARAMETERS USED FOR THE ROBOTIC MANIPULATOR.

i	θ_i [rad]	d_i [mm]	a_i [mm]	α_i [rad]
1	θ_1	75	0	$\pi/2$
2	θ_2	0	67.5	0
3	θ_3	0	67.5	0
4	θ_4	0	65	0

C. Kinematic Chain

A robotic manipulator can be described by a set of joints with links between them where a homogeneous transformation matrix A_i describes the transformation with respect to the previous joint. This means that a transformation that describes the position and orientation of joint j with respect to a joint i can be found by a transformation matrix [5]

$$\begin{cases} T^i_j = A_{i+1}A_{i+2}...A_{j-1}A_j, \text{ if } i < j \\ T^i_j = I, \text{ if } i = j \\ T^i_j = (T^j_i)^{-1}, \text{ if } i > j \;. \end{cases} \quad (11)$$

D. Forward Kinematics

Forward kinematics is the procedure of finding the position and orientation of the end effector given the joint states.

Since a manipulator can be seen as a kinematic chain the problem of finding the position and orientation of the end effector can be solved by finding the transformation matrices in equation (11). The transformation matrices from equation (6) for each joint can be multiplied together

$$T_{end} = A_1 \cdot A_2 \cdot A_3 \cdot A_4 \quad (12)$$

where A_1 - A_4 are calculated from equation (6) to find the pose T_{end} of the end effector and thereby solving the forward kinematic problem.

E. Inverse Kinematics

The procedure of finding the joint states required for achieving a desired pose is called inverse kinematics. For some simple kinematic chains an analytical solution exists, but in general a numerical approach is required. To solve the inverse kinematics for the position of a three joint manipulator a geometric method can be used to find an analytical solution. The required angles to achieve a desired position (x_d, y_d, z_d) in an elbow-up configuration are [6]

$$\begin{cases} \theta_1 = tan^{-1}(x, y) \\ \theta_3 = tan^{-1}(D, +\sqrt{1-D^2}) \\ \theta_2 = tan^{-1}(\sqrt{x_d^2 + y_d^2}, z_d - d_1) \\ -tan^{-1}(a_2 + a_3 \cdot D, a_3 \cdot \sqrt{1-D^2}) \end{cases} \quad (13)$$

where $tan^{-1}(x, y)$ is the two argument arctangent function [7] and

$$D = \frac{x_d^2 + y_d^2 + (z_d - d_1)^2 - a_2^2 - a_3^2}{2a_2a_3} \quad (14)$$

where a_2, a_3, and d_1 are the corresponding DH-parameters.

F. Workspace

A robotic manipulator will not be able to reach all points in space since it has a fixed size. It will not even be able to reach all mathematically reachable points since each joint (usually) have some restrictions on how it can rotate/translate. The physically reachable space is defined as the workspace of the manipulator.

The robotic manipulator seen in Fig. 6 has the following limitations for the first three joints

$$\begin{cases} -100^\circ < \theta_1 < 100^\circ \\ -14^\circ < \theta_2 < 194^\circ \\ -104^\circ < \theta_3 < 104^\circ \end{cases} \tag{15}$$

due to limitations on how much each joint can rotate and limitations such as cables and safety concerns. A simulated 2D slice of the workspace can be seen in Fig. 7. The full workspace can be formed by rotating this 2D slice around the z-axis according to the limitations on θ_1 from equation (15).

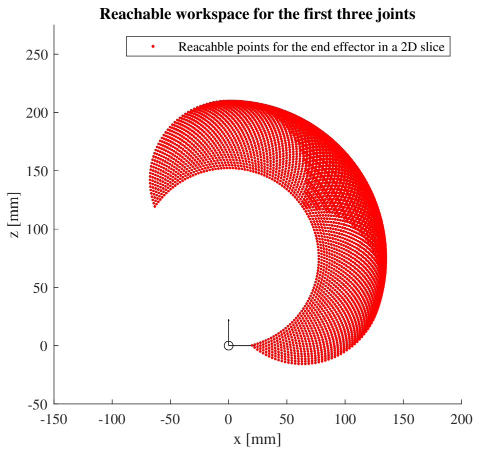


Fig. 7. A simulated 2D slice of the workspace for the three jointed manipulator.

G. Base

The motors which are connected to the tracks of the robot can be modeled as two wheels connected with a rod, as seen in Fig. 8.

Deriving a mathematical model is straight forward if the encoders from the motors can be read. From the encoders both the length the robot has traveled and more importantly the rotational velocity of each motor can be calculated. The speed for the individual motors is given by

$$v_m = \frac{2\pi r_{wh}/N_{enc}}{\Delta t} \tag{16}$$

where N_{enc} is the number of encoder readings per rotation, r_{wh} is the radius of the driving wheel and thickness of the track and Δt is the time between the previous encoder reading and the most recent one. By doing this for both the left and right motor the robots velocity can be calculated by

$$\overline{v} = \frac{v_L + v_R}{2}(-\cos\theta\hat{i} + \sin\theta\hat{j}) \tag{17}$$

with the y-axis parallel to the line. In a similar fashion the angular velocity of the base can be calculated using

$$\dot{\theta} = \frac{v_R - v_L}{2r_b} \tag{18}$$

where r_b is the distance from the center of the base to the wheels. From (18) and Fig. 8 it is possible to see that the change of angle is given by

$$\theta = sin^{-1}\left(\frac{V_L - V_R}{2r_b}t\right). \tag{19}$$

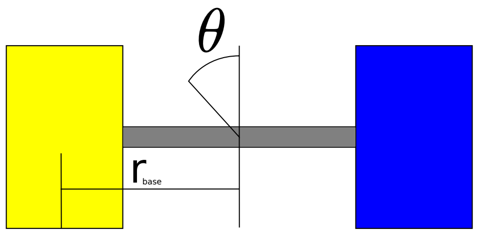


Fig. 8. Modeling of the base as two wheels connected with a rod, where θ is the angle to the line.

Given the use of small-angles approximation[1], the angular velocity can be written as

$$\theta = \frac{V_L - V_R}{2r_b}t. \tag{20}$$

Using equations (16)-(18) a system can be built for simulations in MATLAB, using rate of change and max/min value limiters for simulations of the motors physical restrictions.

H. Line Follower

The robot utilize a line follower controller in order to navigate the factory floor. This in combination with QR codes and a path-planning algorithm will give a huge variety in places that the robot can be instructed to go, as long as it is on the grid and denoted with a QR code.

I. Choice of Controller

In order to move the robot to a specified point the use of a line follower controller was decided on. The robot will read values from the camera that publishes the angle that the PD-controller uses as an input for calculating the signal it sends to the motors IV-A.

The choice of a PD-controller was made for the improved smoothness over a P-controller and the fact that the integrating part in a PID-controller would have undesired effects, as illustrated in Fig.9.

[1]Which will be assumed since the robot will be operating on a grid with hard coded functions for turning at QR-codes

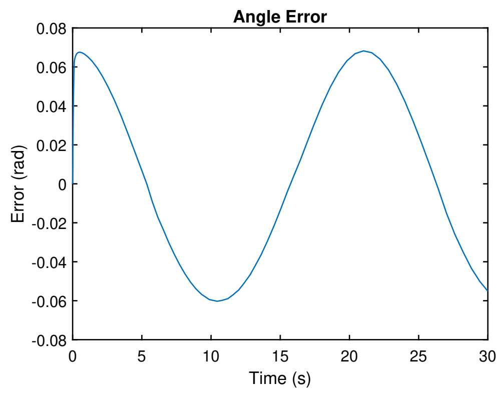


Fig. 10. The error between the robots pose and the line.

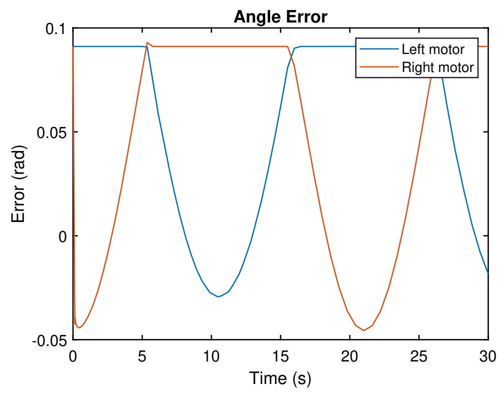


Fig. 11. The motor output for the results in Fig.10.

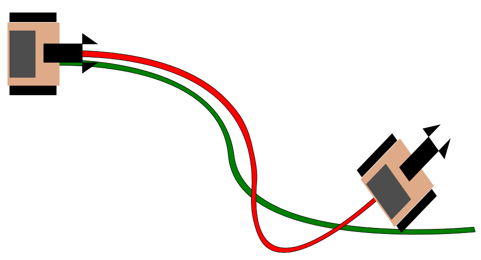


Fig. 9. The robot will keep acting on the integrated error even when reaching the line, this might eventually cause overshoots that makes the robot lose the line. (Green: correct path, red: actual path)

J. Simulation

To verify the validity of using a PD-controller for the line following task, simulations were run in MATLAB. The simulation set the motors to a common speed, so that the controller always makes the robot move forward when $\theta = 0$ and lower the speed of a motor on one side to turn. This is the same way as the controller is implemented on the robot. Both motors speed can therefore be set into one variable that is defined as the difference between the motor speeds ($\Delta v = v_L - v_R$).

The simulation used a sinusoid as input to the controller, simulating that the robot travelled along a line that consistently turn from left to right and back again. The results can be seen in Fig. 10 and 11

The results presented above is evidence that a PD-controller can be used for a line follower.

K. Stability

As touched on in section V-I, the controller needs to be stable for the range in which the robot operates. The stability analysis was preformed on the kinematic equation shown in (18). Since the robot has two controllers, one for turning left and one for turning right, the kinematic equation (18) can be rewritten as.

$$\dot{\theta} = \frac{\Delta v}{2r_b} \tag{21}$$

1) Continuous Time: The Δv will be seen as a positive constant and will be ignored in future equations. Using (21), the open-loop and closed-loop equations becomes

$$G_{ol}(s) = \frac{1}{2r_b s}, \tag{22}$$

$$G_{cl}(s) = \frac{G_{ol}(s)}{1 + G_{ol}(s)} = \frac{(2r_b s)}{(2r_b s)\cdot(2r_b s + 1)}. \tag{23}$$

The function (23) has a pole and zero cancellation at the origin. In Fig.12 the step response, poles and zeros are plotted for the closed loop system. Since all the poles appear on the negative real plane, the system is stable.

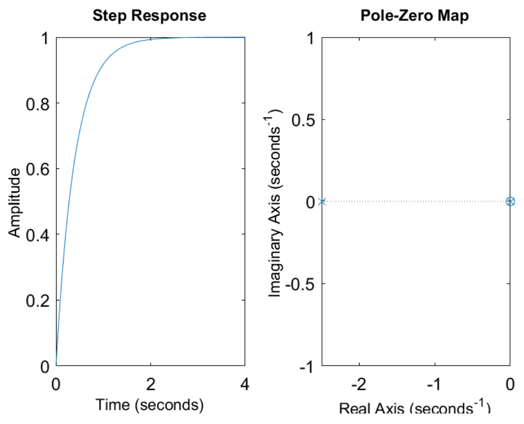


Fig. 12. The closed loop system without a controller.

The PD-controller discussed in previous sections only need to improve the system response, since the overall system is already stable. Introducing the controller to the system gives us the closed loop function

$$G_{clPD} = \frac{2r_b s(k_p + k_d s)}{2r_b s(s(2r_b + k_d) + k_p)}. \tag{24}$$

From equation (24) it is clear that the system will remain stable for $k_d > -2r_b$ and that the value of k_d will not have any effect on the stability of the system in continuous time. The values for k_p and k_d is therefore chosen in such a manner that $0 > k_d > -2r_b$ and k_p so that the largest error input from the MV-node will not saturate the actuators. The result of these are shown in Fig.13 and a comparison was made, represented in Table II.

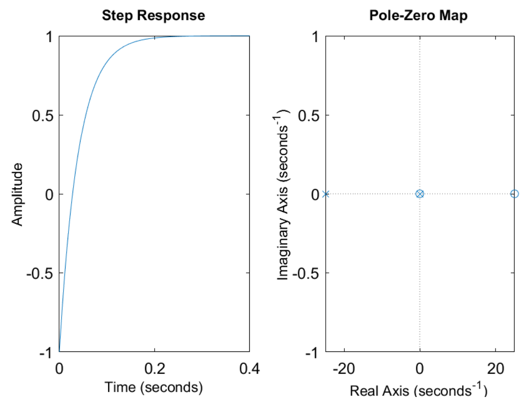


Fig. 13. The closed loop system with a controller.

TABLE II
COMPARISON BETWEEN THE CLOSED LOOP SYSTEMS.

	No controller	PD-controller	Discrete PD-controller
Rise Time	0.8788s	0.0879s	0.1000s
Settling Time	1.5648s	0.1565s	0.2500s

2) Discrete Time: Using MATLAB to move from continuous time to discrete time with a update frequency of 20 Hz[2] and an input delay of $1/30s$[3]. In Fig. 14 it is clear that the pole is within the unit circle on the discrete time plane and that the response follows the step input. In conclusion, the system is stable using the k_p and k_d values proposed above, even in discrete time [8].

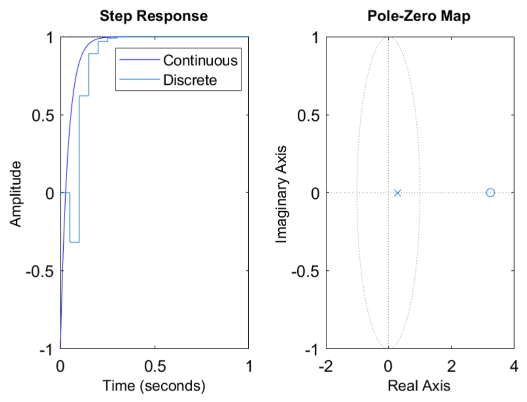


Fig. 14. The continuous step response, discrete step response and the poles and zeros of the discrete transfer function.

[2]The update frequency of the camera
[3]Estimated input delay to the actuators

L. Robot Arm Path Planning

If a robot manipulator is required to visit specific discrete points, known as waypoints, path planning is the process of deciding how to move between these points. Planning the path is also important to ensure the movements are smooth, meaning that, since we control the speed of each joint, the desired positions and velocities should be continuous.

M. Interpolation

One method of interpolating between points is to use cubic spline interpolation. A spline [9] is a piecewise polynomial function and consequently a cubic spline is a function consisting of piecewise cubic polynomials. A cubic polynomial will ensure a smooth movement since its first derivative, the velocity, will be continuous. To ensure that each required point is visited the bounding values for each cubic function are set to zero [10].

N. Joint Space interpolation or Cartesian Motion

The interpolation can take place in joint space or in Cartesian space. Joint space interpolation means that the waypoints are expressed as joint values, angle of rotation if the joint is a revolute joint and distance if the joint is a prismatic joint. The interpolation is then calculated between these joint values. Cartesian motion means that the waypoint are expressed as Cartesian coordinates and that the interpolation takes place in Cartesian space.

O. Implementation

The robotic manipulator, shown in Fig. 6, has a number of waypoints that it is required to visit. The desired waypoints are expressed in joint values, and a cubic spline interpolation is used to interpolate between these joint space waypoints. The path for picking up and placing the object can be seen in Fig. 15 and 16 respectively.

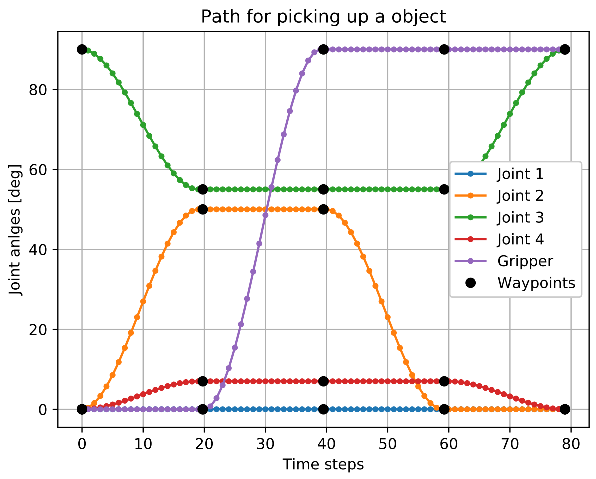


Fig. 15. Path showing the joint values required for picking up the object.

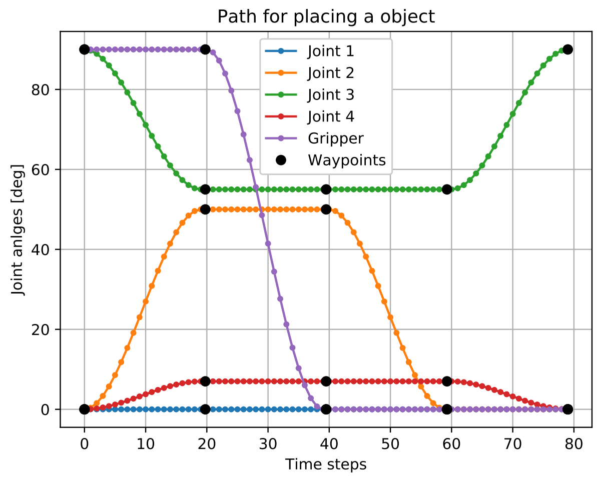


Fig. 16. Path showing the joint values required for placing the object.

VI. Communication

Unlike most servo motors Dynamixel uses digital packet communication instead of pulse-width modulation (PWM) signals. By grouping data into different packages and transmitting it as a bundle. This in turn opens up for many possibilities since the motors responds to commands, have built in control functions together with memory allocation on the motor itself. ROBOTIS recommends two different setups for communication, either by using a Tri-state-buffer to construct your own communication bridge based on Universal asynchronous receiver-transmitter setup (UART). Or by using their own solution called U2D2. In essence, the U2D2 uses the same protocol but also enables for Serial communication between the controller and Dynamixels [11].

A. UART full duplex to half duplex

UART communication scheme works by connecting the transmitter pin (Tx) of the Jetson Nvidia Nano microcontroller (MCU) directly to the receiver pin (Rx) of the motors and vice versa. UART communication can be setup into three different ways of communication. Full duplex, half duplex and simplex. Full duplex allows for data communication both ways simultaneously. Half duplex only allows for one way communication at any given time, by either sending or receiving. Simplex communication is static, meaning communication is always one direction.
Full duplex and half duplex are not directly compatible. If the MCU and Dynamixel motors are connected directly, since the MCU sends and receive data simultaneously it will occupy the transmission line indefinitely and the Dynamixel will not be able to tell when the messages stop. This phenomena is known as chatter. The solution for this is to implement a tri-state-buffer circuit converting the full system to half duplex.

B. Tri-state-buffer

A tri-state-buffer dictates when the MCU is allowed to communicate with the motors. This is explained by Fig. VI-B. By controlling the direction pin (GPIO pin), we can direct the flow of data. If the direction pin is pulled high we simultaneously allow for transmission from the MCU (Tx) to the motors and disable the MCU receiver pin (Rx). A crucial part is timing, the direction pin must be pulled high long enough for the data to transfer through the buffer otherwise information is lost. A logic analyzer will give this information and is recommended to use in conjugation with the tri-state-buffer for tuning.

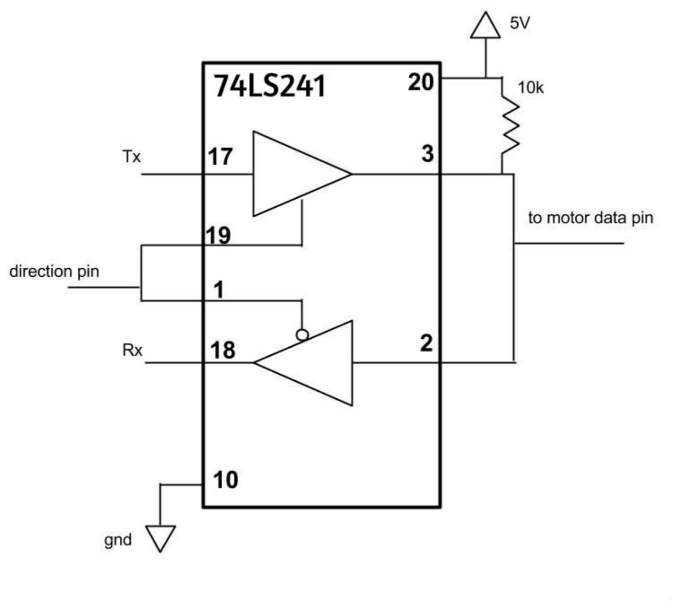


Fig. 17. Tri-state-buffer schematic.

C. Data package - structure of message

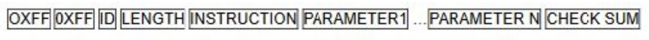
The protocol allows for a size between 5 to 255 bytes per message between the MCU and motors. The two types message structures is represented by the Fig. VI-D and Fig. VI-E.

D. Instruction package

Instruction package carries commands from the controller to the motor. An abundance of support is included, from changing statics parameters like torque, baud-rate or id of each individual unit or different actions, for example move to this location with this speed. The structure of the message is represented in Fig. VI-D and Table III describes the structure from left to right.

TABLE III
Explanation of parameters in instruction packadge

$OXFF$	Indicates start of incoming packet
ID	Specific ID of each unit
$Length$	Length of message
$Param$	Used for additional information
$Checksum$	Calculate length of packet

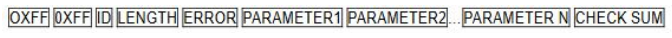


OXFF	0XFF	ID	LENGTH	INSTRUCTION	PARAMETER1	... PARAMETER N	CHECK SUM

Fig. 18. Structure of instruction message.

E. Status package

Status packet contains the response from the Dynamixel to the controller after receiving a instruction. The structure is almost the same as the instruction package. However, one difference is the the message confirms the status of sent instructions including an error if one occurs. The structure is represented in Fig. VI-E and Dynamixels support page [12] provides more information how to read the error messages.



OXFF	0XFF	ID	LENGTH	ERROR	PARAMETER1	PARAMETER2	... PARAMETER N	CHECK SUM

Fig. 19. Structure of return message.

F. Motor communication

Dynamixel provides a lot of support for lower level programming using instructions and reading return messages with documentation support, as stated above . However, there is also support for higher level and more user friendly programming languages like Python, C and third-party libraries by using Dynamixels own software development kit called DYNAMIXEL SDK [13].

VII. Conclusions

A. Engineering challenge

Performance of the robot during navigation on the setup shown in Fig. 1 was evaluated with the following criteria

- Autonomy of the robot
- Speed and navigation
- Pickup / drop-off performance

The test was performed at Luleå University of Technology at EISLAB with a local Arrowhead cloud and private network. The robot was able to navigate the factory floor and perform the action but moved slowly. This is because the Dynamixel motors used to move the robot are slow as they are mostly used for joint control, not continuous rotation. Communicating with Arrowhead was working and the test was proved successful. A video with the whole run can be seen here: https://www.youtube.com/watch?v=YoAFspEC2no. We can't do the evaluation without mentioning some limitations. The speed of the robot is mentioned above but also because of the line following algorithm the robot can get lost if it ever loses the line. This could be prevented with an external navigation system such as GPS, Ultra Wide Band (UWB) or similar.

B. Factory setup

In this project we also propose how a factory test bed can be designed. In this solution the main components are the colored lines and QR codes. The advantage of this setup is that it is highly flexible. First of all the path can have any shape and the QR codes can contain any information. Further, more colors can be utilized to identify different parts of the factory, stop certain robots from following them or use lines with multiple colors to determine the orientation of the robot while following the path. While flexible, this solution may get complicated and hard to alter for very big factories. Also a streamlined system to register QR codes and information would need to be implemented for efficiency and high flexibility.

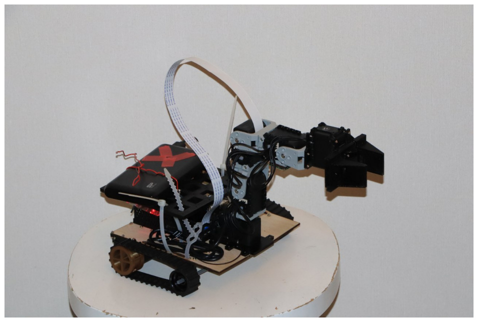


Fig. 20. Final robot design.

C. Design

While the design was initially modular the final design of the robot does not fully follow the concept design. In the final solution a power bank had to be added because the LiPo battery drained too quickly when powering the motors and the NVIDIA Jetson Nano board. Also the design lacked integrated fastenings so zip-ties and tape was used. Because this project focuses more on the control of the robot and Arrowhead implementation the design was not further improved. The final design of the robot in shown in Fig. 20.

Machine vision and line following evaluation

The line following algorithm has generally worked as expected. During testing the robot was able to follow the lines and correctly identify the QR codes. The robot is thus able to travel autonomously between points on the grid in a sufficient manner.

A limitation of the line following algorithm is its sensitivity to noise. Bad lighting conditions and a poorly calibrated color mask can negatively affect the ability to reliably track the line. A possible future optimization to mitigate this effect would be to keep track of the center point of the line in the previous frame. The image moment calculation could then be done in a smaller subsection of the horizontal slice around the previous center. Currently, the algorithm uses the entire width of the frame to calculate the center point. This leads to a heightened sensitivity to random noise in the outer parts of the frame. If the general position of the line is already known only pixels close to this point should be considered, since it is assumed that the line will not move considerably between frames.

Another limitation is the fact that the machine vision system is dependent on sufficiently fast hardware. Some lag will always be introduced while the current image is being processed. If the hardware on which the system runs is slow then this lag introduces the risk of being too large. This will result in the angle sent to the base controller being out of date. However, when running on a Raspberry Pi 4 or the Nvidia Jetson Nano this delay is negligible. This can also be tuned by changing the image resolution used to capture frames with the camera. Preferably, the resolution should be low to increase performance but not so low that the ability to recognize QR codes is impaired. During testing a resolution 360x240 pixels was found to be a good middle ground between performance and image quality.

D. Robotic arm

In the beginning many different arm choices were discussed. One solution was using a simpler forklift design that just moves up and forward in a rigid motion without rotation. This solution would suffice. However, the decision was made to go with an arm design with multiple links connected with motors. This is a more complex solution that opens up for translation and rotation in 3D-space. The design allows the arm to move to any spot in the defined work space and pick up the piece at an angle from the factory. A limitation is the fact that the last arm joint is kept static and parallel to the ground, so the end effector cannot rotate. This can be adjusted by adding one more motor. However, for the task at hand it was deemed unnecessary.

E. End effector

In order to pick up the object a gripping mechanism was needed, different choices were discussed in the early stages. However the simple solution using a claw would suffice. By rotating the shaft of the motor, connected to the claw, the claw either retracts or expands. As stated above this design is easy to implement and works well. One downfall is the fact that the connecting rods that move the claw are fragile and can break for higher torques. This can be improved by using stronger materials for the rods connected between the motor and the claw.

F. Acknowledgements

We'd like to thank out supervisor Jan van Deventer for the constructive feedback during this project. Also Aparajita Tripathy for all the help with Arrowhead and being patient with fusing our Arrowhead clouds. A special thank you to the other student group in the same course, without the rivalry and mutual feedback between the two groups, none of this would've been achieved.

REFERENCES

[1] Stanford Artificial Intelligence Laboratory et al. Robotic operating system. [Online]. Available: https://www.ros.org

[2] J. Flusser, “Moment invariants in image analysis,” *Proceedings of World Academy of Science, Engineering and Technology*, vol. 11, 2006. [Online]. Available: https://perso.telecom-paristech.fr/bloch/ANIM/flusser.waset.06.pdf

[3] J. Brown. (2010) Zbar bar code reader. [Online]. Available: http://zbar.sourceforge.net/

[4] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[5] M. V. ”Mark W. Spong”, ”Seth Hutchinson”, *Robot Dynamics and Control*, 2nd ed. John Wiley & Sons, Inc., 2004.

[6] D. Rus. Robotics systems and science lecture 14: Forward and inverse kinematics. [Online]. Available: https://courses.csail.mit.edu/6.141/spring2011/pub/lectures/Lec14-Manipulation-II.pdf

[7] E. W. Weisstein. ”inverse tangent.” from mathworld–a wolfram web resource. [Online]. Available: https://mathworld.wolfram.com/InverseTangent.html

[8] L. L. Torkel Glad, *Reglerteori, Flervariabla och olinjära metoder*, 2nd ed. Studentlitteratur, 2003.

[9] E. W. Weisstein. ”spline.” from mathworld–a wolfram web resource. [Online]. Available: https://mathworld.wolfram.com/Spline.html

[10] Weisstein, Eric W. Cubic spline. from mathworld–a wolfram web resource. [Online]. Available: https://mathworld.wolfram.com/CubicSpline.html

[11] ROBOTIS TEAM. e-manual from robotis. [Online]. Available: https://emanual.robotis.com/docs/en/dxl/ax/ax-12a/

[12] ——. Protocol from robotis. [Online]. Available: https://emanual.robotis.com/docs/en/dxl/protocol1/

[13] ——. Dynamixel sdk lib. [Online]. Available: https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/overview/

[14] Students, “D7039e,” https://github.com/kottz/D7039E, 2020.

VIII. Appendix

Martin Blaszczyk

Martin is a 5th year Y-student with interest in Control och Mechatronics. In this course he'll take the role of the Project Leader where the main objectives are to keep focus on the goal, hold meetings and an overall oversight of the project. As for the technical part the main interest will be in machine vision together with Edward K. to use cameras or other sensors to localize external objects for the robot to grip, avoid or approach.

Edward Cedgård

Master in electronic systems and control engineering. Edward's main task is to design the robotic arm and gripper mechanism together with Niklas. Tasks as deriving the kinematic equations, and implementation using forward and inverse kinematics. Choise of motors, armdesign, communication with motors using serial communication.

Niklas Dahlquist

Niklas is studying his fifth year at the Engineering physics and electrical engineering student program.

His main focus will be to work with Edward Cedgård to evaluate the gripping mechanism and if necessary design new components and model the corresponding control system to be able to lift up and hold the target object.

Edward Källstedt

Currently studying his fourth year in the Master Programme in Computer Science and Engineering. A fan of making things secure, fast, scalable, and well-documented. Primarily interested in low level software development. Will initially work on the machine vision implementation together with Martin. In addition to machine vision specifically this work will also consist of robot localization and collision avoidance. As the project progresses he will take on more general software problems that might arise.

Albin Martinsson

Albin is a 5th year computer science student specializing in industrial computersystem. In this project he will be focusing on the arrowhead integration and bein charge of the Github repository. This will entail connecting all the services toeach other and making sure they are authenticated and secure. Being in chargeof the git repository will entail merging pull requests and sorting out conflicts, making sure that the version control part of this project runs smoothly.

Måns Norell

Studying for a master in electronic systems and control engineering. Måns main task is to design the base and line-following controller for moving the robot along a line. Tasks include designing the base, printing the specialized parts, simulating and testing the base. Communication between controller, motor and camera will be worked on in collaboration with those in charge of these tasks.

A. *Working together*

Project structure

To keep the project going and have an organized structure the project is divided in different parts, or subprojects. Each group member is either alone or in group responsible for each part of the project which coincides with their interests.

- Arrowhead
- Machine vision and localization of external objects
- Gripping tool
- Movable base

Meetings

Every week the group met on Mondays and Thursdays to catch up and support each other. This structure gave the students a great deal of responsibility to do work for each meeting while still maintaining a good structure of the project and encouraged discussions. The agenda of the monday meeting was:

- Status of work done the previous week by each member
- Preparation for the seminar
  - Discussion of the previous seminar meeting
  - How the weeks work has been coinciding with the seminar feedback
  - Questions to ask the teachers
  - Questions to ask the other group
  - Who does what during the seminar
- Other

The Thursday meeting was to collect and reflect over the feedback from the teachers and the other group from the seminar. Also a status on the work planned to be done the following week was discussed so that each member had a good understanding of what the other members were doing. Those meetings had had the following agenda

- What feedback did the teachers give
- What feedback did the other group give
- Feedback to each other withing the group
- Work to be done the following week
- Other

B. *Project planning*

The projects course started in August 2020 and continued until mid January 2020 with the project deadline in the middle of December. To keep a good structure and to synchronize all the parts of the project a plan was made, shown in Table IV. This plan enabled the group to be flexible and maintain a good long-term structure of the project. Every month a more detailed time plan was prepared to facilitate the small changes, delays etc.

For the group members to what task were to be done, the built in function of Issues on the souce control platform Github® was utilized. A Milestone was created for each week and populated with issues. When an issue was finished it could simply be closed. If the issue was delayed it showed clearly in the Issue overview which tasks had to be prioritized.

TABLE IV
OVERALL TIME PLAN

Sep	Oct	Nov	Dec
Concept generation	Evaluation	Evaluation	
Theory	Prototyping	Evaluation	Finishing up
Simulation	Evaluation	Evaluation	
Prototyping	Final Design	Evaluation	

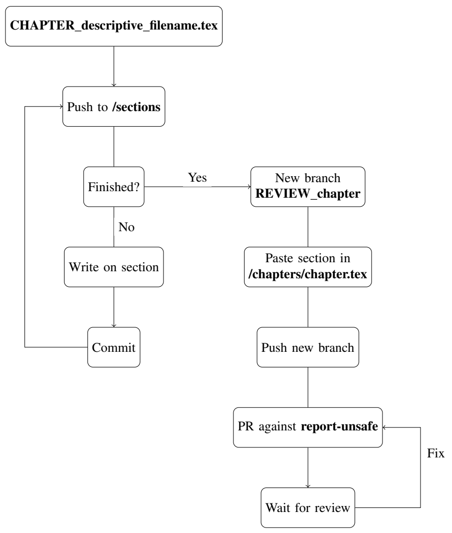


Fig. 21. Report writing flowchart

C. Source control

To keep track of the different software implementation the projects source control implements Git in one common repository [14]. The repository is where all source code and relevand 3D-files are located. This report is written in LaTeX with Git as source control. To make sure it's easy for all members to write their designated sections a workflow was designed to to minimiza merge conflicts while writing drafts as show in Fig. 21. Without this flow the group members would experience merge conflicts after every push which would make it more complex and time consuming.