Mobile Floor-Marking Robot, utilizing Feedback from Laser Tracker

Lisa Klinghav

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6123
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2021

Abstract

Equipment installation at MAX IV, a state-of-the-art particle accelerator in Sweden, requires accuracy and precision in the range of micrometers. The measurement technology used is laser tracking, with which the position of a movable reflector is determined, by reflecting the laser beam emitted from a stationary tracker. Data describing the real-time position of the reflector is handled in a software called SpatialAnalyzer, managing it within the point cloud of statistically secured reference nodes, located all around the facility. The laser tracker is operated via a PC, whereas adjustment of the position of the reflector-equipped inventory is a manual, sometimes trying exercise, performed by measurement engineers. If this task were to be automated, it would free up time and resources, as well as improve the working condition for the employees.

The purpose of this master's thesis was to develop a mobile floor-marking robot to perform bluelining at MAX IV. Bluelining is the first step of equipment installation and it is a process of projecting reference points from a 3D-model to a drawing on the floor. In this step, for example, drill marks for bolts of a foundation of an optic table are to be drawn. A 1,5 mm radius marker-pen is used and a guiding tool, holding the reflector, is positioned by hand to mark the target.

The proposed design consists of a modified 3D-printer attached to a mobile robot equipped with omni wheels. It is controlled via a main program that runs on a Raspberry Pi and it is powered via rechargeable batteries. Performance was evaluated in case studies and the average deviation from target achieved was six millimeters. As the testing was brief, it is recommended to perform further activities to define accuracy, precision and repeatability. However, the indications are that it is possible to perform automatic bluelining with modified off-the-shelf products, receiving positional coordinates from a laser tracker system. The result could be further improved with a custom-made robot, preventing compromises in its automation and design.

Acknowledgements

This master's thesis was conducted in 2020 at the Department of Automatic Control, Lund University, and at MAX IV Laboratory.

I want to thank Anders Robertsson, academic supervisor, for his support and positive attitude. Anders Blomdell, research engineer, made an essential contribution by substantially helping out with the programming and communication protocol of the 3D-printer. The student group including Jacob Berntsson, Kim Haapamäki and Staffan Jeppsson, prepared the TurtleBot and were kind enough to let me build on their work [Berntsson et al., 2020]. Last, but not least, I thank Alina Andersson at MAX IV Laboratory for investing time in the thesis topic and for all the exciting moments together throughout the project.

Sincerely,

Lisa Klinghav
Malmö, March 13, 2021

Contents

1

Introduction

Accuracy in the range of micrometers is required when constructing a state-of-the-art particle accelerator like the one at MAX IV Laboratory in Lund, Sweden. This crucial requirement does not end when the walls and floor of the building are completed, but remains throughout all equipment installation; from drilling and molding of foundations, through to alignment of the optic tables and fixtures for holding samples in place during X-ray experiments. Misalignment causes a decrease in quality of the conducted experiments [Willmott, 2011], why continuously monitoring the installed base is a highly prioritized activity at the facility.

The more advanced technology applied, the higher the demand for accuracy becomes, and the measurement technology currently utilized at MAX IV is laser tracking [MAX IV Laboratory, 2020]. This technology offers a chance to find the exact position of an object, by equipping it with a reflector and beaming a laser onto it. Any motion of the object larger than 0,06 mm can be detected, and as long as the reflector is rotated roughly towards the tracker, contact is maintained throughout the motion. A PC, with a software, like SpatialAnalyzer (SA) [New River Kinematics Metrology Institute, 2020a], is connected to visualize the measurement.

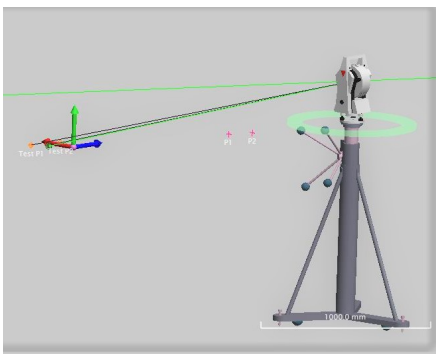


Figure 1.1 Visualisation in SA software.

Figure 1.2 Laser tracker.

1.1 Background

Alignment of all equipment at MAX IV is the responsibility of a small team of research engineers in the Survey, Alignment, and Mechanical Stability (SAM) team. The first stage in the process of equipment installation is called *bluelining*; an activity of projecting characteristic points of the three-dimensional model of the facility and equipment onto the floor. The accuracy required, to make possible micrometer accuracy in the later stages of installation, is relatively low: in the range of 1-2 mm.

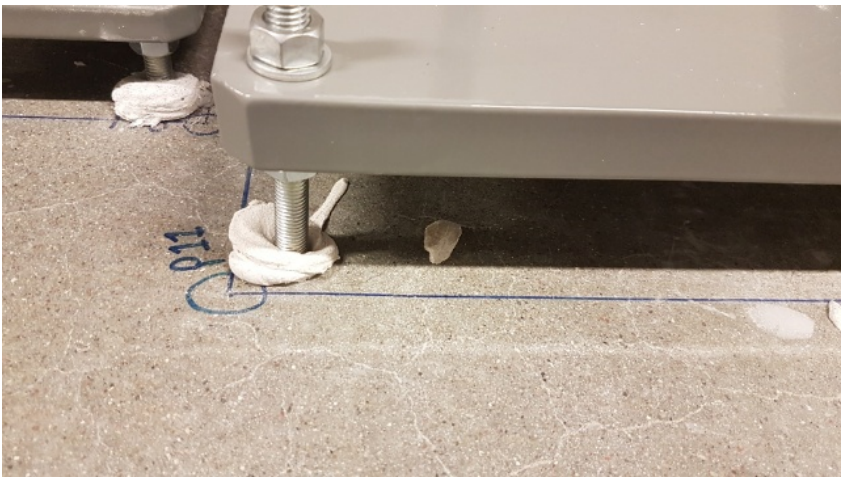


Figure 1.3 Drilled and installed equipment foot, with one corner projected as P11.

Figure 1.3 shows point P11, circled and connected to other points with blue lines. What is practically done during bluelining at MAX IV is to equip a manual tool, presented in Figure 1.4, with the reflector, facing the tracker. The position is fine-tuned by sliding the tool by hand over the floor, constantly checking the result on a computer screen. The reflector, seen in Figure 1.5, has a 60° field-of-vision, and when the target point is found the reflector is removed and a marker pen is used to draw a mark. This task is tedious, monotonous, and often done for many hours in a row, sitting uncomfortably on the concrete floor of a construction site [Robotics at Lund University, 2020]. This current, manual process has several disadvantages that need to be addressed:

- The process is time consuming as each point is to be positioned manually.
- The working position can be uncomfortable and ergonomically questionable.
- It takes two persons to do bluelining, one marking and one facilitating data.
- Pen precision is a limiting factor due to a radius of 1,5 mm.

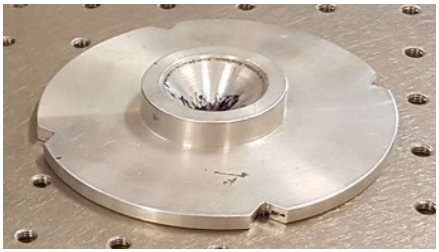


Figure 1.4 Manual tool with seat for reflector and center hole for pen.

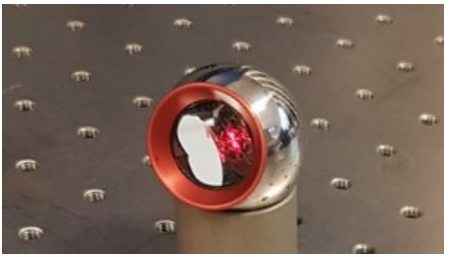


Figure 1.5 Reflector with 60° field-of-vision and in contact with red laser.

- At times the conditions at the site are hazardous (radioactive) or unpleasant (construction dust), which prevents the work from being completed timely.

1.2 Problem formulation

The manual and uncomfortable task of bluelining is well-motivated to be automated, and it is easy to imagine benefits like improved working conditions, higher productivity, and potential to free up time for more intellectually stimulating activities.

The fact that the driving force for starting such an initiative comes from within the SAM-team also implies that there is a chance that a successful implementation can lead to a more positive attitude towards the task in focus.

At European Spallation Source (ESS), currently under construction in Lund, bluelining is a frequent task and similar laser tracking technology is applied. On top of that, bluelining is performed on most construction sites, though possibly together with other position-tracking techniques, and with less strict requirement for accuracy. Hence, there is a clear potential in the concept of mobile robot assisted laser tracking, and useful learning to be made from designing and programming a technical means for high accuracy floor-marking.

1.3 Goal of the master's thesis

The purpose is to develop a mobile floor-marking robot to assist bluelining based on laser tracking. Off-the-shelf products are to be applied, modified in terms of design and programming, and the work to be documented in the form of a master's thesis.

1.4 Delimitation

The work is to be performed within the range of a one-student degree project of 20 weeks, but there is a long term perspective on the concept and therefore a modular

concept is requested for the structure of the automation program. That way any artefact used as a prototype, may be replaced in future steps without loosing the chance to benefit from the work and effort done.

The long term vision is to have a robot that can perform all types of laser tracker based measurement and alignment activities at MAX IV, including drawing on floors and walls, adjusting optic tables and fixtures on machines up to several meters high. If, in addition, a future version of the concept can be remotely operated and cope with radiation, it could operate inside the accelerator ring during experiments, preventing costly stops in operation. However, this degree project is limited to address the specific task of bluelining and will only focus on changing the way positioning of the reflector is done. The activities concerning operation and setup of the laser tracker, how the bluelining coordinates are processed in the software or other related tasks will not be evaluated or changed.

Some of the work presented in this thesis, around programming in Robot Operating System (ROS) [Pyo et al., 2017], and modification of a mobile robot were done by a group of students in a separate project. The work performed in this thesis, related to that, included only to apply their solution as a transporter unit.

1.5 The structure of the report

Chapter 2 is a description of the supporting theory for this master's thesis and the methodologies used are presented in Chapter 3. The results are written in Chapter 4, and are divided into one section on the system design and another section on performance testing. Reflections on the achievements and results are found in Chapter 5, and in Chapter 6 the conclusions and future work are presented.

2

Theory

The theory that this master's thesis is based on is presented in this chapter. The most relevant information about mobile robots is presented first, followed by laser tracking and bluelining.

2.1 Mobile robots

In the ISO standard 8373:2012, Robots and robotic devices - Vocabulary, there is a definition of what a robot is. It is a technical means that may be either mobile or stationary and which fulfill certain characteristics:

1. Automatically controlled
2. Reprogrammable
3. Used for multiple purposes
4. Moves in three or more axes

Robots are commonly used in the industry, as they can be designed to move and perform tasks repetitively, and with accurate mathematical models, high precision of their motions can be achieved [Freidovich, 2017].

Automatic control

When a measured unit is fed back to a control system it provides what is called a closed loop [Glad and Ljung, 2006]. The opposite is open loop where you may run a process according to a recipe and cannot compensate the input signal based on the performance (output). Putting on cruise control while driving a car is a familiar application of automatic control. The level of automation can vary from full automation, like that when an auto pilot is flying an airplane, down to assisted driving of a car, that occasionally kicks in to prevent accidents. Automatic control of a self-positioning, mobile robot can be described using block charts as in Figure 2.1.

When the actual position is fed back and compared to a reference position, the loop is closed, making it possible for a robot to reach its target. Feedback based on laser tracking is based on information transfer via User Datagram Protocol (UDP), and data management is done in a software like for example SpatialAnalyzer, SA, from New River Kinematics [New River Kinematics Metrology Institute, 2020b].

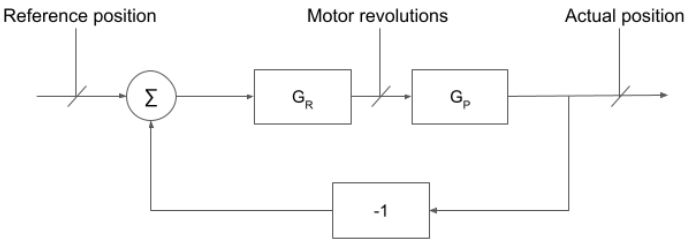


Figure 2.1 Block chart for a closed loop.

Motion calculation

In robotic kinematics it is common to work with several different coordinate systems, called frames. One method for denoting points and vectors, used for expressing a position or movement in relation to a certain coordinate system, is homogeneous transformation [Freidovich, 2017]. It provides one mathematical operation that simultaneously handles rigid body motion [Freidovich, 2017], in terms of rotation (see Figure 2.2) and shift of origin: translation (see Figure 2.3). Figure 2.4 shows the combined case of transformation.

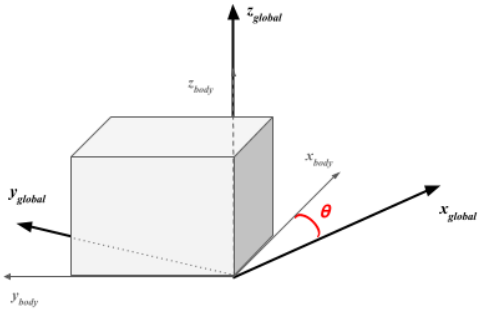


Figure 2.2 "Positive rotation of a body", around the z-axis of the global frame.

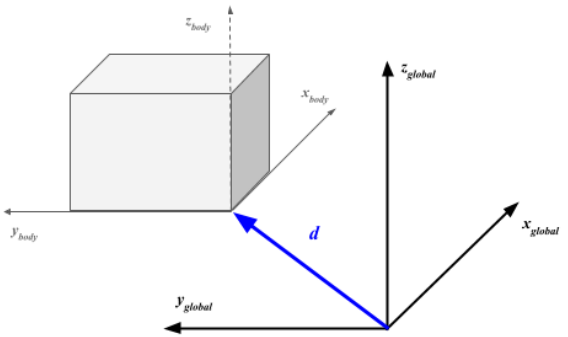


Figure 2.3 "Translation a vector *d*", relative to the origin of the global frame.

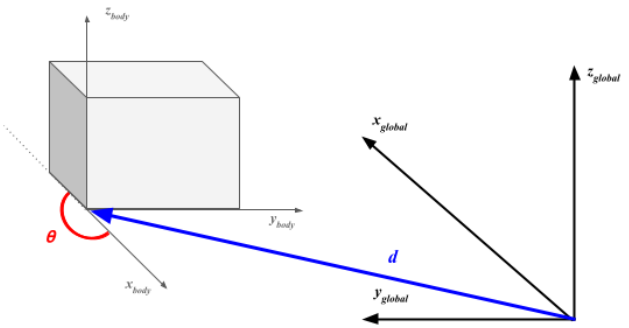


Figure 2.4 Transformation is rotation and translation combined.

The rotation of a body's frame, in this case an angle θ around the z-axis, can be expressed in global coordinates, using the corresponding rotation matrix.

$$R_{body}^{global} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \overset{\theta=0^\circ}{=} \begin{bmatrix} \cos 0^\circ & -\sin 0^\circ & 0 \\ \sin 0^\circ & \cos 0^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The word global is elevated to represent which coordinate system is used for reference, and can be read as *body expressed in global*. As can be seen, a diagonal matrix with zeros and ones is the result of a 0° rotation. Equation 2.1 shows how a point in space can be transformed between coordinate systems using matrix calculation.

$$P^{global} = T_{body}^{global} \cdot P^{body} \tag{2.1}$$

where the transformation matrix, T_{body}^{global}, is:

$$T_{body}^{global} = \begin{bmatrix} [& R_{body}^{global} &]_{3x3} & [-(R_{body}^{global} \cdot d)]_{3x1} \\ 0 & 0 & 0 & 1 \end{bmatrix}_{4x4}$$

The one is added as a dummy and d, the translation vector, is:

$$d = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix}^{global}$$

A point expressed in the body frame is represented as a vector, P^{body},

$$P^{body} = \begin{bmatrix} x_b \\ y_b \\ z_b \\ 1 \end{bmatrix}^{body}$$

with the 1 being a dummy. A point expressed in the global frame is represented by another vector, P^{global}:

$$P^{global} = \begin{bmatrix} x_g \\ y_g \\ z_g \\ 1 \end{bmatrix}^{global}$$

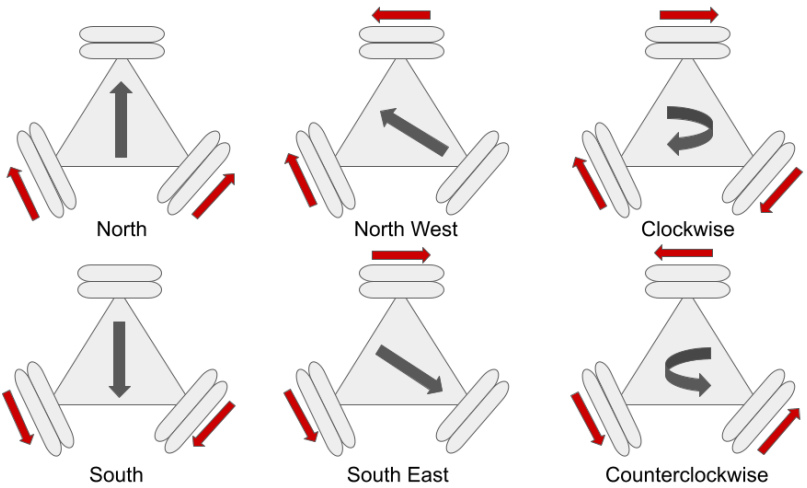
Equation 2.1, with variables inserted and written in vector form, becomes:

$$\begin{bmatrix} x_g \\ y_g \\ z_g \\ 1 \end{bmatrix}^{global} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & -(\cos\theta \cdot \Delta x - \sin\theta \cdot \Delta y) \\ \sin\theta & \cos\theta & 0 & -(\sin\theta \cdot \Delta x + \cos\theta \cdot \Delta y) \\ 0 & 0 & 1 & -(\Delta z) \\ 0 & 0 & 0 & 1 \end{bmatrix}_{body}^{global} \begin{bmatrix} x_b \\ y_b \\ z_b \\ 1 \end{bmatrix}^{body}$$

Omnidirectional motion

The prefix omni- means “all-“, and has its roots in Latin language. Hence, omnidirectional motion implies that movement in all directions is possible. The, so called, omni wheels provide a motion pattern where turns are not necessary for directional change. Supply a car with omni wheels and you end up with a vehicle that can slide sideways into a tight parking pocket on the side of a road, known as parallel parking.

Within the field of robotics, the term holonomic is used to describe this ability of instantaneous change of direction. There are several configurations possible for omni wheels and one, based on three wheels in triangular pattern is presented in Figure 2.5. The red arrows by the wheels show which of the three are active in each case.



[Parmar J., Savant C., 2014]

Figure 2.5 Possible motion using omni wheels (figure modified from source).

Movement using omni wheels, work on the principle of the parallelogram law, and will always be in the direction of the resultant vector between two, or more, forces acting at an angle to each other [Parmar J., Savant C., 2014].

2.2 Laser tracking

Laser trackers are portable metrology instruments used for short- and long-range, high-accuracy measurements of scanned and discrete point data [New River Kinematics Metrology Institute, 2020a]. High accuracy applications, like aircraft assembly, nuclear plant or particle accelerators installation are often facilitated using measurements made by laser tracking [Biao M, Guorui Y, Weidong Z, Yinglin K, 2014], which is a relatively expensive technology.

A Leica laser tracker consists of a unit emitting a laser beam (see Figure 1.2) and a reflector (see Figure 1.5). The Leica AT40x connects via a wired Ethernet connection or a wireless connection. In either case, the IP configuration must be correct before the PC and instrument can communicate. The computer must be set to the same network with a unique IP address.

The data stream feature in SA enables the instrument interface to broadcast real-time reflector position over the UDP. This allows the interface to send this data over a network without requiring a specific data connection to be established beforehand. This means that other clients on the network can be set up to receive this data. For example, you might set up UDP data streaming to broadcast real-time positional information to a custom client application that is listening on the proper port for UDP data packets. The client application can receive and process information, to perform a task, such as using the information to control a motorized pan/tilt head camera mount [New River Kinematics Metrology Institute, 2020a].

2.3 Bluelining

Bluelining is a term used for drawing on the floor at construction sites. There is not one single, formal definition of the word, in fact it is used for many different things;

A blueprint is a reproduction of a technical- or engineering-drawing, using a contact print process on light-sensitive sheets, used for reproduction of specification drawings, in construction and industry [Wikipedia, 2021]. The process is today obsolete, but the word is still used by engineers and architects to refer to a floor plan.

A blue-line is defined in a glossary for printing plates as;

> "A photographic proof made from flats for checking accuracy, layout and imposition before plates are made." [The Pressroom Inc., 2021]

In this thesis the term *bluelining* is defined as;

> *An activity of projecting characteristic points of the three-dimensional model of the facility and equipment onto the floor.*

3

Methodology

The methods applied in this degree project were observation, semi-structured interview with users, prototyping, and performance validating experiments in the form of case studies.

3.1 Understanding the task

After taking part in practical bluelining at MAX IV and getting familiar with the manual process it was possible to describe the process in detail, from the perspective that it was to be performed by a mobile robot. Automatic bluelining requires a point list, derived from a virtual model of the building and equipment, and system to perform localization, positioning and printing.

Localization

The localization task concerned knowing where the reflector was in the room and keeping track of the position relative to a set target. It involved keeping track of targets and decoding of UDP, with positional coordinates in three dimensions, sent from the laser tracker.

Positioning

The positioning task was defined as the action in which the reflector-equipped vehicle moved from starting position, traversed each target defined in the point list and stopped when the list was completed. The fact that the laser beam could only be reflected back to the laser tracker while kept rotated within +/- 30° from neutral position (parallel to laser tracker x-axis) imposed a restriction to how the vehicle had to navigate. It could be described as a drone-like, or floating motion across the floor.

Printing

The printing task concerned transferring reference points from a virtual model to real space, as marks on the floor with minimum error. It also involved a procedure

to find the rotation of the 3D-printer relative to the global coordinate system, shown in Figure 3.1.

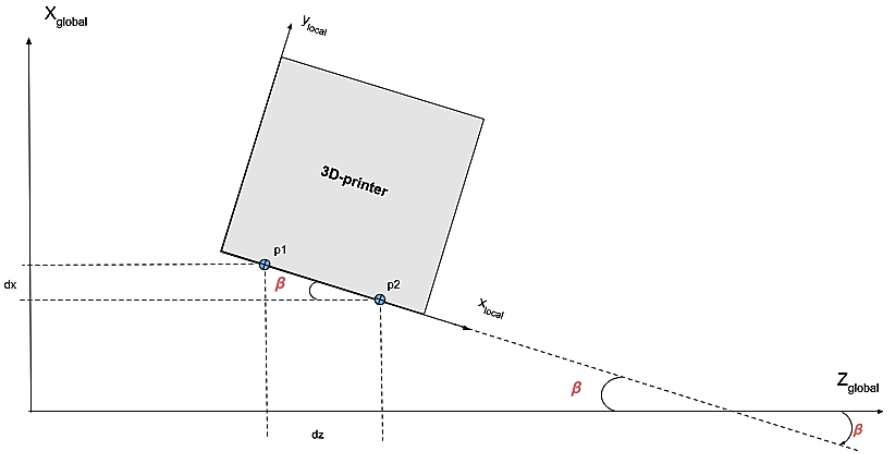


Figure 3.1 Procedure to find rotation, presenting a case with negative rotation.

The procedure was to register the coordinates of a starting point, p1, moving along the local x-axis and register coordinates again, in position p2. The angle β ranges from -30° to +30° and can be calculated as:

$$\beta = \arctan(dx/dz) \tag{3.1}$$

where:

$$dx = p2x - p1x$$
$$dz = p2z - p1z$$

The pen to be used for printing was of the same type as used for manual bluelining: a 1,5 mm radius Artline70 marker.

3.2 Setting the requirements

When the project was launched there was a lack of specific requirements, and the first step of the work was to evaluate the need and make a specification.

Interview

The members of the SAM team were interviewed in a semi-structured way [Höst et al., 2006] to get to know the future users and better understand the need. The questions covered were:

1. What steps during bluelining, if any, are easy to perform?
2. What steps during bluelining, if any, are difficult to perform?
3. Explain the weakness, if any, of the existing manual process.
4. What is your view on how the process could be improved?
5. If the process would be made less manual, what effect do you foresee?
6. What features/functions do you wish for in a potential automated process?
7. What risks are related to an automation of bluelining?
8. What is your overall view of the current process of bluelining?

The major insight was that the improvement in working posture and conditions really was the strongest drive for automation of bluelining. Also, it became clear that the accuracy of the manual positioning was enough for bluelining and that it would be sufficient if the same level of accuracy were achieved with the proposed design. Even a slightly lower level of accuracy could be accepted.

Initial requirements

The first set of requirements was put together while considering a system design based on a combination of a TurtleBot 3 Waffle Pi and a M3D micro+ 3D-printer (to be explained in the section on System design). Some points may occur too subjective or limited to the specific combination. However, it was agreed that there could be other combinations possible, for example exchanging the printer for a Delta-type instead of a cartesian. One reason to write them down was to, better late than never, make sure that the expectations were kept on a reasonable level, and that it was worth it to spend the money on the M3D. The TurtleBot was already purchased, and used for other applications. As many conditions were unfamiliar during this early innovation project, the requirement setting was to be iterated throughout the work. The following list of requirements is sorted with descending importance:

1. Pen-positioning; Pen/reflector-axis mounted completely vertical, with maximum distance between pen tip and reflector: 100 mm.
2. Accuracy; maximum deviation from target: 2 mm radius.
3. Range; Possible to operate 2-10 m from the laser tracker. Positioning in the range of 100 mm deviation from target to be handled by M3D, above that handled by TurtleBot.
4. Operation; moving the robot by hand only to its starting position, then automatically perform a job according to a point list.

5. Communication; Ethernet between laser tracker and laptop, wireless for the rest.
6. Speed; Perform a job of about 30 points in 4 hours including setup time.
7. Power supply; battery for TurtleBot and plug in the wall for M3D.
8. Cost; 5,500 SEK (excluding TurtleBot).
9. Weight; Maximum payload for the TurtleBot: 30 kg.
10. Other; Cope with construction dust and work well at a temperature of 30°C.

Revised requirements

After working with the M3D micro+ for a few weeks it was decided that its x- and y-motions, when tested for reflector positioning only, were precise enough for the purpose and that it could cope with the added weight. The object oriented programming language Python could be well suited for controlling the printer, and it was considered possible to develop the type of application that would be required. However, there were concerns that the common axis for the pen and reflector would not be rigid enough and that the distance between pen tip and reflector was too long, risking poor precision and accuracy when used to mark the floor. The market was scouted for a replacement but there was no good alternative found. Instead of changing the type of printer and loosing time it was decided to follow through with the M3D in this project, and to focus on developing the conceptual algorithm for automatic bluelining.

Long term, positioning and point accuracy were still the most important requirements, but it was decided to start a separate initiative to develop a device that would be custom fit for the task, rather than working with off-the-shelf products and modify them, as in this project.

Even a robust solution, where the pen-axis could be adjusted, would on some level experience a tilt, as seen in Figure 3.2, and it was decided to program the robot with a possibility to compensate for that, rather than redesigning the hardware to prevent it.

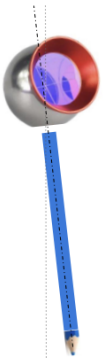


Figure 3.2 Tilted pen-axis, causing the pen tip location to be off reflector center.

Figure 3.3 shows how the tilt was to be dealt with in the program.

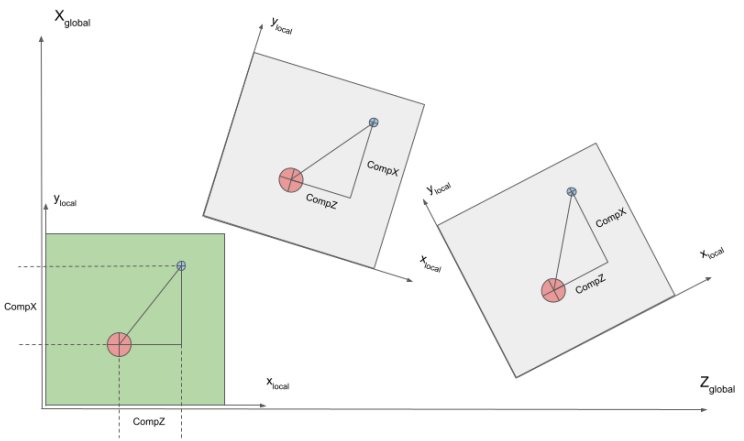


Figure 3.3 3D-printer orientation in global coordinate system.

The green box in Figure 3.3 represents what is done during setup of the system; The 3D-printer is aligned with the global coordinate system and a target is bluelined, with deviation laser tracked and registered. The grey boxes symbolize scenarios during bluelining, where the printer has moved and rotated. Throughout the job, the pen tip (blue circle) will, regardless the translation or rotation, need to be move the distances CompZ and CompX to draw right below the reflector (red circle). In an ideal case, with no tilt, the red and blue circles would overlap.

Another concern raised was that the full area of the M3D was not reachable with the current design. Requirements 1-3 were revised, taking the before mentioned concerns into account. It was also added that the robot needs to operate closer to the laser tracker than earlier stated, whereas all other requirements were left unchanged.

1. Pen-positioning;

   Pen/reflector-axis mounted as close to vertical as possible, with least possible distance between them and add possibility to compensate for tilt in the code.

2. Accuracy;

   Evaluate and minimize deviation from target.

3. Range;

   Possible to operate 0,8-10 m from the laser tracker. Positioning in the range of 25 mm deviation from target to be handled by M3D, above that handled by TurtleBot.

Automatic bluelining

To successfully perform automatic bluelining the mobile robot must:

1. Be compatible with localization using laser tracking, via UDP.
2. Read a text file specifying targets, and keep track of them throughout.
3. Move with rotation relative to the tracker maintained within +/- 30°, position a 3 mm diameter marker pen and draw dots on a concrete floor.
4. Complete the job in one, autonomous, sequence.

Bluelining case studies

When the robot was ready its performance was evaluated in case studies, further described in Section 4.2 Evaluation of performance. Targets were communicated to the bluelining system via a point list, saved as a .txt-file. When the list was completed, the manual tool was placed on each mark, and the coordinates registered using the laser tracker. Documentation was made throughout using a camera and the results were analyzed and visualized in MATLAB.

4

Results

The most important result of this master's thesis is the system design and a detailed description of it is made in this chapter. The practical testing of the system was rather brief due to time limitation and external factors. Hence, the results presented from these case studies, are to be considered a framework for future testing rather than a quantitative analysis.

4.1 System design

To automate the task of bluelining, an off-the-shelf mobile robot and a simple 3D-printer were modified, connected, and integrated via a custom made, main program. Designing the program was a major part of the work and it made possible to handle communication with the laser tracker and between the modules and to perform the task. The overview of the logic was; read the target coordinates from a point list, initiate TurtleBot positioning, hand over to the printer for fine tuning of position and drawing of the point. One important challenge was to combine the modules to one system that performed each subtask in the right order and with sufficient result.

The bluelining robot

The bluelining robot applied in this project can be seen in Figure 4.1, and it was designed to be modular, so that the logic could be applied even if the type of printer or mobile robot would be changed in the future.

The main parts of the system were; 1) TurtleBot 3 Waffle Pi mobile robot – for long-range positioning, 2) Raspberry Pi 3+ – for running the main Python-program and 3) M3D micro+ 3D printer – for short-range positioning of the pen holder (3a) and reflector (3b). A dlink router (not shown in the picture) was used for wifi communication.

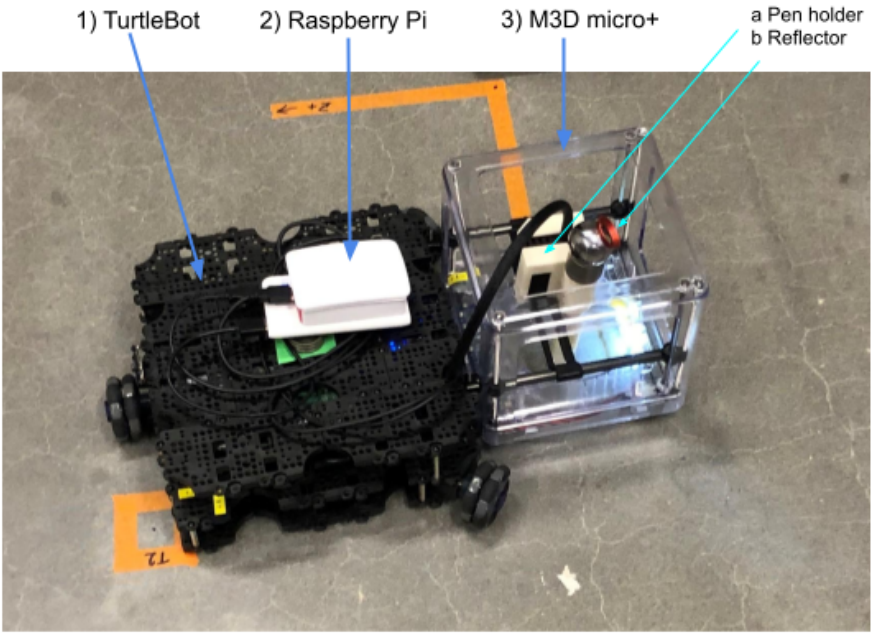


Figure 4.1 Bluelining robot system.

Positioning using a TurtleBot is a common setup for mobile applications [Kamat et al., 2016]. The subtask of positioning the pen and drawing was considered well suited to be performed by a 3D-printer. The printer and mobile robot were modified according to below descriptions.

Modified TurtleBot 3 Waffle Pi mobile robot

This type of mobile robot operates with a with Raspberry Pi and input-output for motors is conducted via a single board computer. It comes with an inertial magnetometer unit, IMU, reacting on the earth's magnetic field. Its IMU is an integrated triple-axis gyroscope, triple-axis accelerometer, and triple-axis magnetometer sensor in one chip, making possible various applications without adding a sensor [Pyo et al., 2017].

The standard wheels would not qualify for the required movement pattern, but a group of students at LTH made the characteristic motion possible by reprogramming it to operate with omni wheels and utilizing the internal IMU for maintaining orientation. If launched in a certain angle relative to the laser tracker, the bot will strive to retain it throughout the job, also in case it is displaced due to, for example, hitting an obstacle. It works like a compass and if pushed out of course it will automatically return to its starting rotation relative to the global frame. The necessary

customization was to supply it with omni wheels and Dynamixel motors. Dynamixel are common in robotics because of their various functions [Pyo et al., 2017]. A motion algorithm was developed in ROS, using a remote PC with Ubuntu operating system. As the laser tracker follows the reflector motion, if kept within a rotation of +/- 30°, the laser beam is reflected back to the tracker, and the position can be calculated and sent to the Raspberry Pi.

Modified M3D micro+ 3D-printer

An off-the-shelf M3D micro+ was purchased for 5,000 SEK and modified. The reason for choosing this particular printer for the system design was the fact that it has a print head that moves in three dimensions, unlike many others with a moving print bed, and that it is hollow under the bed, making it possible to access the floor via a hole in the bottom. A combined reflector-pen-holder was 3D-printed and slid over the extruder head. The reflector had to be placed on, or as close to, the pen-axis as possible. A hole in the bottom and ball caster wheels enabled motion and access to floor space below the pen tip. 3D-printers are operated via G-code and a script that enabled this code to be sent from the main Python-program, was developed by a research engineer at the department of Automatic Control to support this project.

In Figure 4.2 the holder for the pen and reflector, made out of PLA plastic, can be seen. A magnetic nest for keeping the reflector in place, is screwed into it with an M6. The pen container is screwed into the upper part before the complete adapter is positioned on the printer.

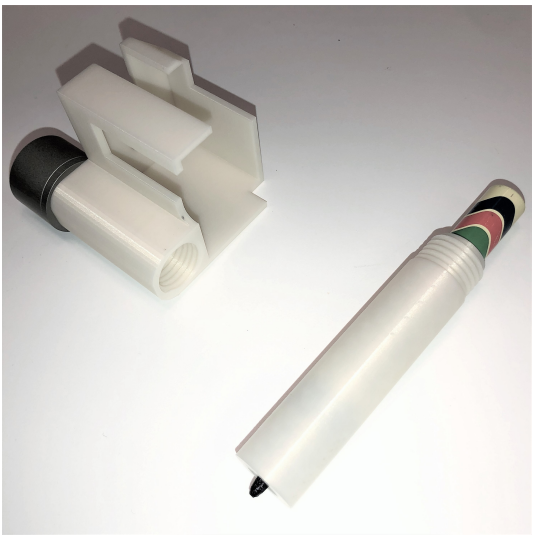


Figure 4.2 Disassembled pen holder.

An ideal design would be to find a position somewhere in the printer, that allowed pen and reflector to be located on the same vertical axis, AND a minimum distance apart. The geometry and the operation of the printer did not allow for a pen position in the center of the printer, so it was designed with an offset, defined as “offset point” in the program. The frame would block the laser beam if a shorter pen would be applied and the chosen design was unfortunately a compromise, resulting in several weaknesses of the design:

1. To know its position of the print head an operation called homing needs to be performed on a regular basis. One side of the holder had to be left open, not to interfere with a rod during homing, when extruder head is pushed to its maximum x- and y-positions, limiting the possibilities for ideal positioning of the pen.

2. The distance between the reflector and the pen tip is unfortunately long, not to have the frame blocking the laser beam.

3. The weight of the reflector is applied in a position that is poorly supported, leading to a tilt on the reflector-pen-axis and a deviation between reflector position and pen tip position.

Power supply

The Raspberry Pi and the M3D are both supplied using a 5V, 2,1A power bank, at an estimated cost of 400 SEK, including cabling. The TurtleBot comes with a 12V battery, supplying all internal devices.

Automation logic

An algorithm for the coordination of the tasks was developed and the functional architecture is presented in Figure 4.3. The programming was done in Python 3.

The positioning task is divided between TurtleBot and M3D printer. In the algorithm, it is the Bot that positions the system until the deviation from target is within reach for the printer. Then the printer performs fine-tuning until a set deviation is achieved and a drawing loop can be entered. This process is repeated until there are no more points in the list of targets.

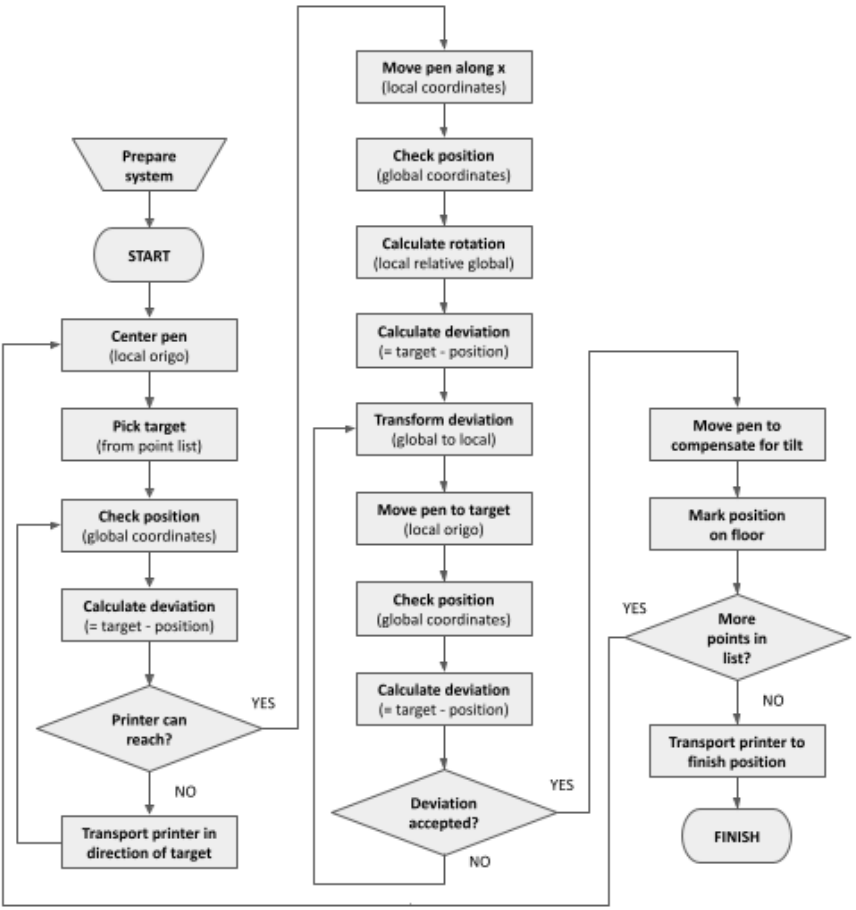


Figure 4.3 Algorithm for automatic bluelining.

Homogeneous transformation applied

The bluelining robot has a rigid connection between the TurtleBot and the 3D-printer. This means that the concept of homogeneous transformation is possible to apply to the system in one single coordinate-transformation step. The vehicle's rotation angle, β, about the laser tracker y-axis (named y_{global} in Figure 4.4) varies throughout the task, but the angle of the local frame relative to the global is, in every moment, the same for the entire system.

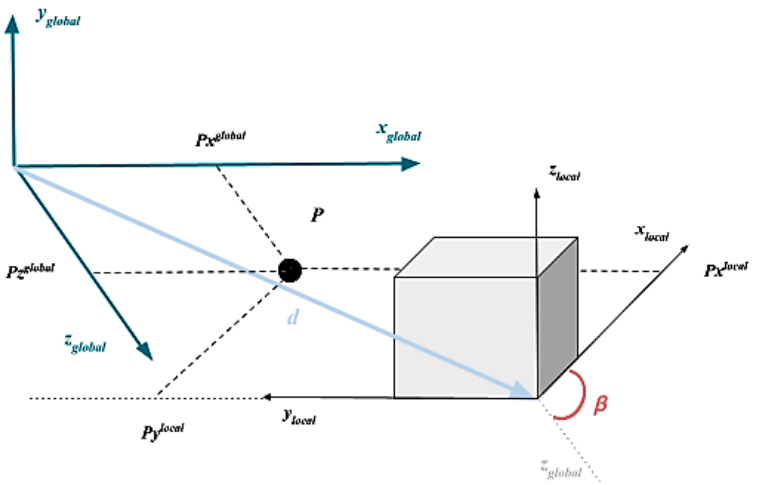


Figure 4.4 Projection of a point on the floor onto global and local frames.

Figure 4.4 illustrates projection of a point and the relation between local and global frames during the bluelining task. The origin of the local frame, representing the 3D-printer, is located a vector d relative to the origin of the global frame (representing the laser tracker). The rotation about the global y-axis is in this example a positive angle, β, starting from the global z-axis and ending at the local x-axis. Note that the angle is exaggerated in the picture. As described in the requirements, the signal is lost if the angle is not kept within +/-30°. Also, note the difference from the general case, where the angle θ is located between global and local x-axes. In the bluelining case, the angle θ is equal to $90° - \beta$. The local z-axis is parallel to the global y-axis, and when there is no rotation, the local x-axis is parallel to global z-axis. The local, or body, frame can be expressed in the global coordinate system using the rotation matrix:

$$R_{body}^{global} = \begin{bmatrix} \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \\ \cos\beta & -\sin\beta & 0 \end{bmatrix} = \begin{bmatrix} \sin 0° & \cos 0° & 0 \\ 0 & 0 & 1 \\ \cos 0° & -\sin 0° & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

The targets to be marked in this project were defined in the global, laser coordinate system, but had to be recalculated to the local, printer system, in order to position the pen. In the applied case the floor is assumed to be flat and the local frame axes assumed to be perpendicular. The point P in Figure 4.4, represents a target during bluelining, and the matrix calculation performed to transform its global coordinates to the local, body, frame is described below. It follows the general method, though the rotation matrix looks different. Applied to this specific bluelining case, the theory of homogeneous transformation can therefore be used as described in Equation 4.1,

$$P^{body} = (T^{global}_{body})^{-1} \cdot P^{global} = T^{body}_{global} \cdot P^{global} \tag{4.1}$$

where the transformation matrix, T^{body}_{global}, is:

$$T^{body}_{global} = \begin{bmatrix} [& R^{global}_{body} &]^T & [-([R^{global}_{body}]^T \cdot d)] \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} [& R^{body}_{global} &]_{3x3} & [-(R^{body}_{global} \cdot d)]_{3x1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The transposed rotation matrix, R^{body}_{global}, is:

$$R^{body}_{global} = [R^{global}_{body}]^T = \begin{bmatrix} \sin\beta & 0 & \cos\beta \\ \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \end{bmatrix}$$

and the translation vector, d, with no movement along y-axis, is:

$$d = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} x \\ 0 \\ z \end{bmatrix}^{global}$$

A point expressed in the printer frame, *body*, is represented as a vector, P^{body},

$$P^{body} = \begin{bmatrix} x_b \\ y_b \\ z_b \\ 1 \end{bmatrix}^{body}$$

with the 1 being a dummy. A point in the laser frame, *global*, is represented as another vector, P^{global}:

$$P^{global} = \begin{bmatrix} x_g \\ y_g \\ z_g \\ 1 \end{bmatrix}^{global}$$

Equation 4.1 with variables inserted, written in vector form, becomes:

$$\begin{bmatrix} x_b \\ y_b \\ z_b \\ 1 \end{bmatrix}^{body} = \begin{bmatrix} \sin\beta & 0 & \cos\beta & -(\sin\beta \cdot \Delta x + \cos\beta \cdot \Delta z) \\ \cos\beta & 0 & -\sin\beta & -(\cos\beta \cdot \Delta x - \sin\beta \cdot \Delta z) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{body}_{global} \begin{bmatrix} x_g \\ y_g \\ z_g \\ 1 \end{bmatrix}^{global}$$

A simplified version of the matrix could have been used for rigid body motion on a plane (2D), but an implementation of the complete (3D) matrix in the automation program was done. This was to make it possible to take into account rotation about more than one axis in the future.

4.2 Evaluation of performance

Case studies were performed to validate the performance of the bluelining robot. The floor marking performed by the mobile robot was verified with the manual bluelining tool, currently used. It was placed over the mark and the coordinates were registered using the laser tracker. The tests were carried out in the SAM-team room to debug the code and evaluate the performance. The process was iterative and modifications were made along the way.

Case study 1 - Printing the Roman number IV

Fifteen points, forming the Roman number four *IV*, were bluelined, and Table 4.1 describes the targets and results.

Three targets were picked for control measurement; point six, seven and nine in the table above. Figure 4.5 is a plot of the three control measured results, plotted using MATLAB. The line between nominal target and actual result represents what is referred to as *radius* in Table 4.1. It is the hypotenuse of a triangle, with the sides *Dev x* and *Dev z*, calculated using Pythagorean theorem.

Point	*Target x, z*	*Actual x*	*Actual z*	*Dev x*	*Dev z*	*Radius* [mm]
1	1100, 0					
2	1100, -300					
3	1100, -400					
4	1100, -500					
5	1200, -400					
6	1200, -50	1213.202	-39.635	13.202	10.365	16.785
7	1200, 50	1213.013	60.850	13.013	10.850	16.943
8	1300, 100					
9	1300, -100	1313.766	-89.430	13.766	10.570	17.356
10	1300, -400					
11	1400, -500					
12	1400, -400					
13	1400, -300					
14	1400, -150					
15	1400, 150					

Table 4.1 Targets and results from printing the Roman number IV.

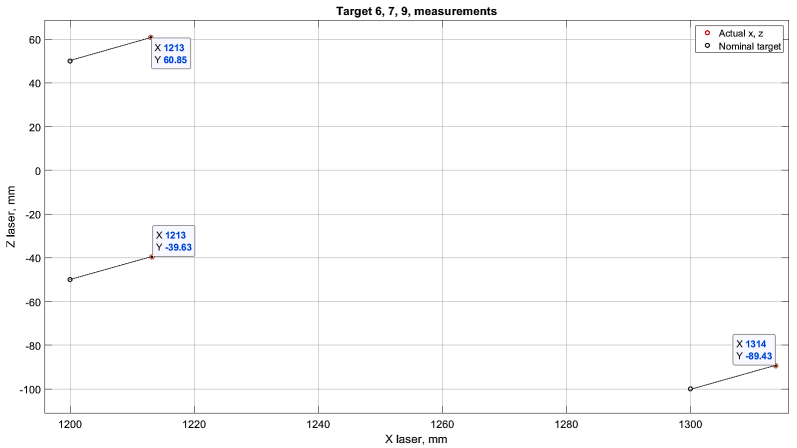


Figure 4.5 Targets and results for point six, seven and nine.

Figure 4.6 shows the fifteen plotted targets, circled and numbered. The formation came out clearly and the result was expected to be very good. However, after measuring three results expectations were less optimistic. It was apparent that the accuracy

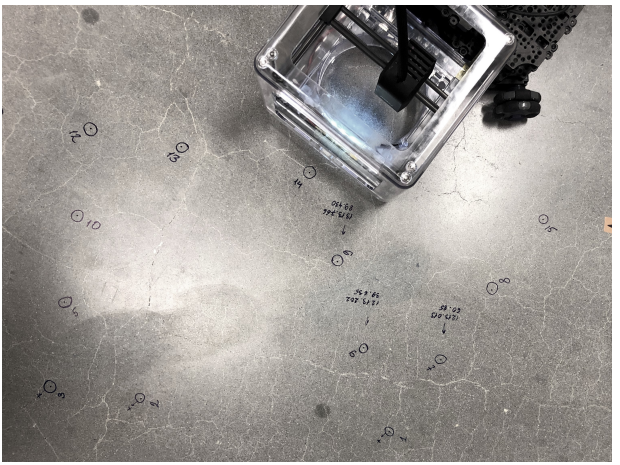


Figure 4.6 Before connecting the dots.

was not according to expectation, and a different approach was taken after visual inspection of the floor. The formation of the markings revealed the number four clearly, and it was decided to try to connect the dots and evaluate the accuracy based on that.

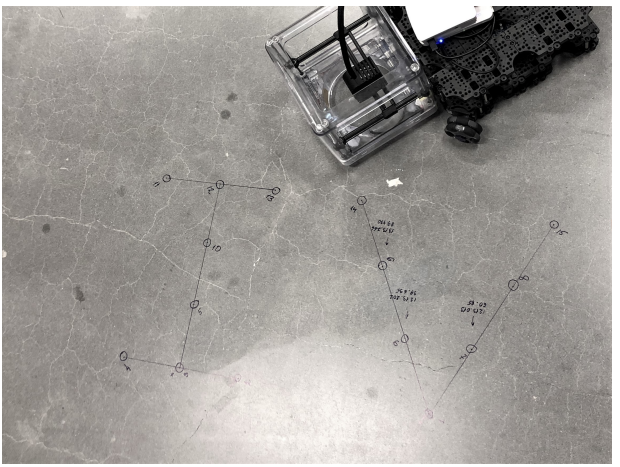


Figure 4.7 After connecting the dots.

In Figure 4.7 the marks have been connected with straight lines. As can be seen, the position of the marks relative to each other is very precise. They are not accurate, but the system shows consistency throughout the job. Figure 4.8 illustrates the difference between accuracy and precision.

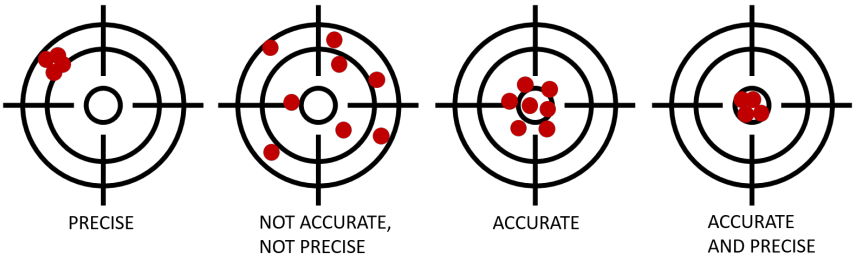


Figure 4.8 Accuracy compared to precision.

Conclusion, case study 1 The performance was evaluated as precise, but not satisfying in terms of accuracy. The variations were analyzed and it was found that the pen tilt was the reason. It was at this point in time that the concept of compensating for tilt was first introduced. Tests were to be redone with a compensation step added to the code. Also, a padding was decided to be used to fixate the pen. By decreasing the print depth (vertical motion) from 5 to 2,5 mm, the mark would be visible, while preventing to push harder than necessary.

Case study 2 - Bluelining a realistic job

A bluelining job, representing a real scenario, was set up. The job consisted of ten targets, saved in a point list in .txt-format. The main program, presented earlier, read the file and performed the sequence autonomously.

Figure 4.9 shows the nominal targets and marked results from three test runs, measured with the manual tool and plotted using MATLAB.

An attempt to compensate for the tilt of the pen was made, and Table 4.2 shows the result. The average radius achieved was 6,4 mm, which was an improvement compared to previous case study. However, the deviations had began to fluctuate in a much different matter than before.

Point	*Target (x, z)*	*Actual coordinates (x, z)*	*Radius [mm]*
1	1300, 0	run 1: 1305.904, -4.166	7.226
		run 2: 1304.603, -0.893	4.689
		run 3: 1306.449, 0.636	6.480
2	1400, 0	run 1: 1404.541, -3.090	5.492
		run 2: 1406.254, -0.435	6.270
		run 3: 1409.112, 3.651	9.817
3	1500, 0	run 1: 1504.154, -2.874	5.051
		run 2: 1504.107, 0.874	4.199
		run 3: 1507.516, 2.416	7.895
4	1600, 0	run 1: 1604.710, -2.706	5.432
		run 2: 1606.854, 0.200	6.857
		run 3: 1603.963, 2.956	4.944
5	1700, 0	run 1: test failed	
		run 2: 1706.734, 0.863	6.789
		run 3: 1704.466, 3.391	5.608
6	1800, -400	run 1: test failed	
		run 2: 1805.602, -400.836	5.664
		run 3: 1807.034, -397.389	7.503
7	1800, -200	run 1: 1806.014, -203.081	6.758
		run 2: 1806.097, -200.382	6.109
		run 3: 1807.204, -197.157	7.745
8	1800, 0	run 1: test failed	
		run 2: 1805.021, -0.131	5.023
		run 3: 1806.896, 2.986	7.515
9	1800, 200	run 1: 1805.836, 197.052	6.539
		run 2: 1806.581, 199.799	6.584
		run 3: 1806.835, 203.30	7.560
10	1800, 400	run 1: 1806.720, 396.346	7.649
		run 2: test failed	
		run 3: 1805.294, 402.555	5.878

Table 4.2 Targets and results from bluelining a realistic job.

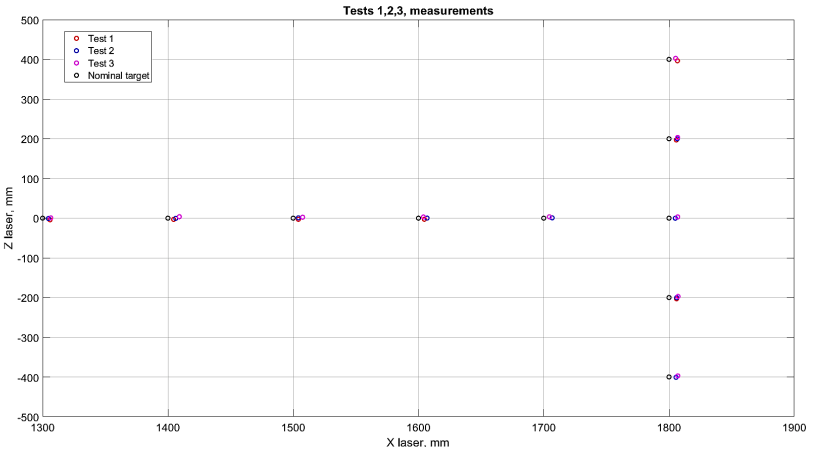


Figure 4.9 Ten targets bluelined.

The fluctuation can be seen when zooming in, as in Figure 4.10, where results from test run two and three are plotted. The visible targets are number five, seven, eight and nine from the case study *bluelining a realistic target*.

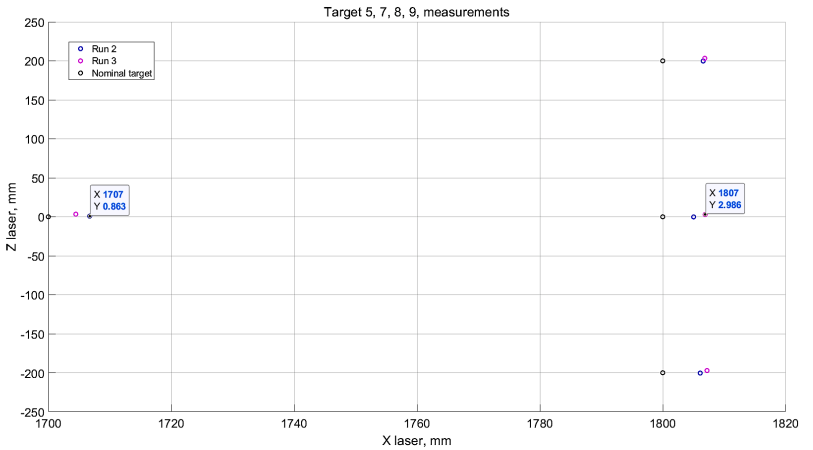


Figure 4.10 Targets 5, 7, 8 and 9 zoomed in.

Conclusion, case study 2 The average deviation from target had been decreased from 17 to about 6 mm. However, variations of a totally different kind than before were observed. The result was no longer off target in a constant manner. Analysis showed that a faulty method of compensation for tilt was applied, along with a

mix-up of CompX and CompZ variables in the code (see Figure 3.3 for a recap of meaning). Therefore, tests were to be redone with revised code.

Case study 3 - Positioning using only TurtleBot

Before the third case study, unfortunately, there was an electrical incident and the 3D-printer stopped working. While waiting for delivery of a replacement it was decided to use what was still available, and perform a case study with only the TurtleBot. That way it was possible to evaluate if the TurtleBot could position itself with high accuracy, and give an answer to the question whether using the printer for fine-tuning was indeed necessary.

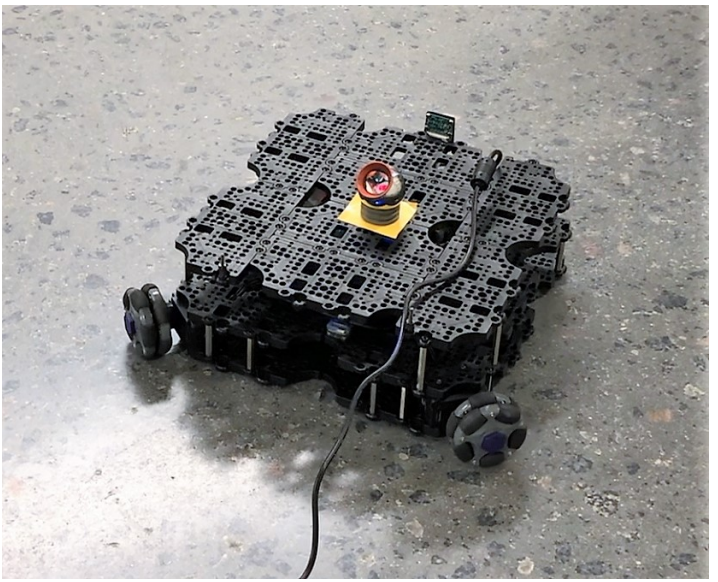


Figure 4.11 Setup when TurtleBot tested separately.

Ten targets were bluelined using only the TurtleBot, instructed to position the reflector within a distance from the target of ten millimeters. No marks were made on the floor, but laser tracker measurements were made during a ten second break between two targets, when the Bot was stationary. Five millimeters distance was also tested, but then the Bot never got to a point where it stopped and measurements could be carried out. MATLAB was used to analyze the data and Figure 4.12 shows the targets and results from three repetitions. Target ten had no successful results achieved, but the other results are presented in Table 4.3.

Point	*Target (x, z)*	*Actual coordinates (x, z)*	*Radius [mm]*
1	1300, 0	run 1: 1294.405, -2.791	6.252
		run 2: 1310.106, 7.179	12.396
		run 3: 1292.585, 5.478	9.219
2	1400, 0	run 1: 1391.648, 0.715	8.383
		run 2: 1393.350, -5.783	8.813
		run 3: test failed	
3	1500, 0	run 1: 1490.908, 2.851	9.529
		run 2: 1492.813, -2.163	7.505
		run 3: 1491.963, 0.721	8.069
4	1600, 0	run 1: 1588.706, 1.943	11.460
		run 2: 1591.728, 3.330	8.917
		run 3: 1587.416, 3.978	13.198
5	1700, 0	run 1: 1700.374, 5.552	5.565
		run 2: 1678.521, 1.694	21.546
		run 3: 1695.752, 0.093	4.249
6	1800, -400	run 1: 1806.416, -394.024	8.768
		run 2: 1790.179, -399.055	9.866
		run 3: 1774.196, -389.248	27.954
7	1800, -200	run 1: 1801.243, -193.197	6.916
		run 2: 1797.259, -210.488	10.840
		run 3: 1803.362, -196.168	5.098
8	1800, 0	run 1: 1761.574, 11.212	40.028
		run 2: 1782,585, 0.873	17.437
		run 3: 1777.499, 1.712	22.566
9	1800, 200	run 1: test failed	
		run 2: test failed	
		run 3: 1766.142, 194.722	34.267
10	1800, 400	run 1: test failed	
		run 2: test failed	
		run 3: test failed	

Table 4.3 Targets and results from positioning using only TurtleBot.

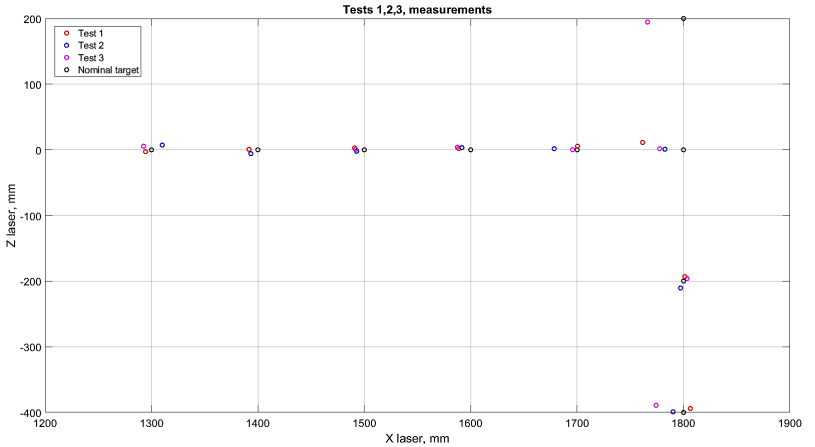


Figure 4.12 Nine targets bluelined successfully.

The reason for all three runs to fail for target number ten was that it was the last target in the loop, and that the code was written in such a way that the ten second pause was skipped in this run. Therefore, no measurement was possible.

Conclusion, case study 3 The average deviation from target, achieved using only the TurtleBot, was 13,3 mm (12,1 mm if one outlier removed). To match the accuracy of the manual method, a mechanism for fine tuning of the position is, indeed, necessary.

Further testing postponed

Due to the current pandemic, access to the facilities at Max IV became heavily restricted right after case study 3, and the practical testing of the bluelining robot was postponed until further notice. Therefore this thesis report had to be finalized based on the results conducted before the close-down. When practical testing can resume, the format in which the results were presented is proposed to be used as a framework.

5

Discussion

This master's thesis addressed disadvantages, experienced when laser tracker facilitated bluelining was performed manually, by automating the task using a mobile robot system. Primarily, the poor ergonomics was addressed with good result, and the proposed solution is operated by one person in a timely manner. If further developed, the bluelining robot has potential to improve working conditions, increase productivity and free up time for other tasks in the SAM-team. The response from the team members has been very positive and activities around automated bluelining will continue towards the long-term vision: to have a robot that can perform all types of laser tracker based measurement and alignment activities at MAX IV.

Goals To conclude the master's thesis in a frank manner it should be stated that the goal was not entirely achieved. The accuracy simply did not meet the requirements for taking it into use for bluelining at MAX IV today. On the other hand, it was a stretched target and a somewhat under-explored field, and there are indeed parts of the project that were a great success; A fully automatic bluelining robot, that is compatible with laser tracking technology, is designed, programmed and built. The code for automation is modular and prepared for upgrading the system with hardware that can provide higher accuracy in the future. Moreover, development of strategies and test methods for validation of performance are initiated. However, the testing of the robot was, due to time limitation and the described external factors, very brief and it is required far more testing done to be able to draw statistically reliable conclusions from the results.

Requirements A review of requirements fulfillment was carried out and it concludes the following:

Pen-positioning was a challenge and the system design was not able to avoid a tilt of the pen/reflector-axis. However, a possibility to compensate for the tilt was provided in the main program.

Accuracy; To evaluate this requirement more testing would be needed. The average deviation achieved was 6,4 mm, which is three times higher than the initial requirement. However, the requirement was revised early on, setting a goal to minimize the deviation rather than to aim for a fixed value. The compensation for tilt improved the accuracy and it is likely that continued adjustments of the bluelining robot and code will lead to further improvement. While performing manual bluelining it is possible to mark the floor with an accuracy of 1-2 mm, and that was not matched by the proposed design.

Range; The system was proven to operate well in the range 0,8-10 meters. If necessary in the future, it would be possible to extend the transport by TurtleBot down to thirteen millimeters. That is because the Bot managed to position the reflector closer to the target than the predefined 25 millimeters.

Operation and *Communication* requirements were both fulfilled.

Speed was never evaluated quantitatively, as other requirements came first, but the impression is that it meets the target of 30 points in four hours including setup. There is a possibility that it will be even faster than required once the working procedures are practiced more extensively.

Power supply was an area where performance was better than required. It was possible to supply the M3D with a 5V, 2.1A power bank, since the extruder (requiring higher current) was not run.

Cost; the budget was managed within the stated 5,500 SEK (5,000 for the M3D and 400 for electrical components). The price of the M3D has since been decreased and a replacement unit can be purchased from the U.S. manufacturer for 3,300 SEK, including import fees and tax.

Weight; The payload limit for the TurtleBot is 30 kg, and towing the M3D including added parts and reflector was no issue.

Other; The performance was tested in 22°C, but not in high temperature environment, as requested for the future application. Small amounts of construction dust did not cause any problem that cleaning of the omni wheels could not solve.

Performance evaluation Access to the laser tracker equipment, only possible at MAX IV, was a limiting factor throughout the project. The case studies had to be well prepared and it was not possible to go back and redo them whenever. An attempt of coding a virtual laser was made and it worked well for testing the UDP-communication. For position feedback, however, the laser tracker had to be used.

Valuable lessons learned came out of the testing, even if they may seem like unfinished work. Unfortunately, there was an electrical issue and the 3D-printer stopped working before the practical testing was completed. The performance evaluation is work in progress, rather than anything else. If there would have been opportunity for it, the testing would be redone, with mistakes like the faulty compensation for tilt, corrected. That way the results of that test, presented in this report, could have been replaced. It was, however, decided to present even the poor results, as a way to show how tests were performed, and to provide know-how for the day the trials are resumed.

Looking back at the project, the decision to carry on with the M3D micro+ printer, despite the concerns about robustness and performance, was the correct one to take. Finishing with a more complete and optimized fine-tuning unit, while not managing to achieve the integration with the transport unit through the main program, would have been considered less of an achievement. Yes, the accuracy is not as good as with manual bluelining, but the ability to perform a job in an autonomous sequence, with all subtasks managed, hopefully built trust and increased the likelihood of continued investments. There is, indeed, potential to perform accurate and precise bluelining with a robotic system in the near future.

Automatic bluelining The addressed task was described as an event that required: compatibility with laser tracking, target management, characteristic motion during transport and printing, as well as autonomous completion of the entire job sequence. When evaluated in terms of that description, it is concluded that the mobile robot system, designed and programmed in this master's thesis, successfully performed automatic bluelining.

6

Conclusion

Drawn conclusions from this master's thesis is that the laser tracker system, currently used, provides positional coordinates via UDP in a reliable manner and that it certainly is possible to localize and position a mobile robot based on its feedback. Omni wheels provide motion that lets a vehicle be rotated a certain relative angle, while moving in all directions on a plane. An IMU works well to compensate for rotational drift in the SAM-team room, but it is unlikely that it works in areas where the magnetic field of the particle accelerator is stronger. The TurtleBot cannot get close enough to the target by itself, and it will be required to apply a fine-tuning module. A M3D micro+ 3D-printer is not accurate enough if only modified as in this project. More extensive rebuilding would be required and it is probably a better idea to custom build a technical means rather than going for off-the-shelf products. That way certain design parameters, like the reachable area and the distance between reflector and pen tip, could be optimized, and compromises avoided.

Looking ahead As mentioned, it will most likely be necessary to find a replacement for the IMU due to disturbance from magnetic fields. One option is to perform a similar operation as that which the printer does when finding its rotation, but instead letting the entire vehicle move.

One action that has not been discussed earlier is the potential to replace the marker pen with, for example, an inc jet printer. This would make possible a dot with a smaller radius than 1,5 mm.

When a custom made bluelining robot is developed, it is strongly recommended to design it as one physical unit, rather than two separate that are connected and integrated in a common program. This would give benefits to both coding and the physical design. The main program could, with the proposed system design, actually not control the way the TurtleBot performed its positioning task. The Bot was programmed in ROS and the main program could only control *when* the positioning by Bot should be switched ON or OFF; As described in Figure 4.3, by entering block "Transport printer in direction of target", or not. However, for this early stage of the concept development, the separate transport and printing units had advantages; It was possible to run the printer separately, placing it within reach of the target by hand,

and performing fine-tuning. And when the printer was under repair, the transporter could continue operating. A modular design provides such functionality.

An initiative is started at the Department of Automatic Control that attempts to replace the TurtleBot with a compact transporter, or potentially a combined transporter/printer, equipped with Dynamixel motors and omni wheels. That way, the need for students to install software to run ROS during development can be avoided. Further, there is already one student project ongoing, developing a custom-fit system that will be controlled via the program developed in this master's thesis. Both concepts appear promising, and it will be very interesting to follow their progress.

Bibliography

Berntsson, J., K. Haapamäki, and S. Jeppsson (2020). "FRTF20 - Bluelining MAX IV".

Biao M, Guorui Y, Weidong Z, Yinglin K (2014). "Measurement error analysis and accuracy enhancement of 2D vision system for robotic drilling". *Robotics and Computer-Integrated Manufacturing* **30**:2, pp. 160–171.

Freidovich, L. B. (2017). *CONTROL METHODS FOR ROBOTIC APPLICATIONS, Lecture notes*. Umeå University.

Glad, T. and L. Ljung (2006). *Reglerteknik, grundläggande teori*. Studentlitteratur/Appia.

Höst, M., B. Regnell, and P. Runeson (2006). *Att genomföra examensarbete*. Studentlitteratur/Appia.

Kamat, V., R. Manthal, and C. Menassa, (Eds.) (2016). *Ambient Data Collection in Indoor Building Environments Using Mobile Robots*. Vol. 33. International Symposium on Automation and Robotics in Construction. ISARC, MI, USA.

MAX IV Laboratory (2020). *Magnus Malmgren and Alina Andersson - SAM team at MAX IV Laboratory, film*. `https://youtu.be/JueNflX_VW4`. Accessed: 2021-01-04.

New River Kinematics Metrology Institute (2020a). *SpatialAnalyzer User Manual*. New River Kinematics, Inc., 436 McLaws Circle, Williamsburg, VA 23185, U.S.

New River Kinematics Metrology Institute (2020b). *Website of New River Kinematics, Inc.* `https://www.kinematics.com/index.php`. Accessed: 2021-01-05.

Parmar J., Savant C. (2014). "Selection of wheels in robotics". *International Journal of Scientific & Engineering Research* **5**:10, p. 341.

Pyo, Y., H. Cho, R. Jung, and T. Lim (2017). *ROS Robot Programming*. ROBOTIS Co., Ltd.

Robotics at Lund University (2020). *Robotics research: Bluelining for installation of equipment at MAX IV*. `http://robotics.lu.se/video/`. Accessed: 2021-01-04.

The Pressroom Inc. (2021). *The pressroom & newsprint glossary*. `https://www.pressroom2inc.com/resources/glossary.html`. Accessed: 2021-01-23.

Wikipedia (2021). *Blueprint*. `https://en.wikipedia.org/wiki/Blueprint`. Accessed: 2021-01-24.

Willmott, P. (2011). *An Introduction to Synchrotron Radiation: Techniques and Applications*. New York: Wiley.

A

MAIN PROGRAM

```
#!/usr/bin/python3
import math
import numpy as np
import M3D
import socket
import time
import threading
from inspect import currentframe, getframeinfo

# BLUELINING - THE AUTOMATED PROCESS:
# preparation:
# - position Laser Tracker
# - create laser coordinate system in Spatial Analyzer software
# - make sure laser follows reflector and place reflector on Printer
# - save point list text file in the same folder as blue.py
# set variables textFile, TOLERANCE, OFFSETX, OFFSETY, OFFSETZ, COMPX, COMPY, COMPZ,
    debug, ipBlue, ipTransporter, RADIUS
# start blue.py on pi@printer and let it run until it prints bluelining done
# start turtle bot as transporter:
    #1 Roscore; open terminal, type ssh...
    #2 Bringup; open new terminal, ssh as #1
    and type roslaunch turtlebot3_bringup turtlebot3_robot.launch, press enter
    #3 Launch;  open new terminal, ssh as #1
    and type roslaunch FRTF20blueliningmaxIV launch.launch, press enter
    #4 Listen;  open new terminal, ssh as #1
    and type rostopic pub /start std_msgs/Bool "data: true", press enter
# stop turtle bot; in same terminal as #4: press ctrl+c,
type rostopic pub /start std_msgs/Bool "data: false", press enter, press ctrl+c
TOLERANCE = 0.06                        #[mm] = how close you want to be before you draw
OFFSETX = 73.037                        #[mm] = printer coordinate that brings pen tip
                                            to a place where it can reach +/- RADIUS in x
OFFSETY = 60.248                        #[mm] = printer coordinate that brings pen tip
                                            to a place where it can reach +/- RADIUS in y
OFFSETZ = 27.998                        #[mm] = printer coordinate that brings pen tip 2mm
                                            above ground
RADIUS = 2.000                          #[mm] = radius printer can reach
COMPX = -9.26                           #[mm] = laser coordinates, input due to pen tip not on
                                            same axis as reflector, based on rotation angle =0
COMPY = -9.0                            #[mm] = laser coordinates, input due to pen tip not on
                                            same axis as reflector, based on rotation angle =0
COMPZ = -7.0                            #[mm] = laser coordinates, input due to pen tip not on
                                            same axis as reflector, based on rotation angle =0
ipBlue = "192.168.100.107"              # = ip of computer running blue.py
ipTransporter = "192.168.100.102"       # = ip of computer performing transport (eg Pi@Bot)
textFile = "pointListA.txt"             # name of textfile with bluelining points
debug = False                           #set True if debugging printouts desired
```

```
class Transporter:

    def __init__(self):
        pass

class Point:

    def __init__(self, x = 0.0, y = 0.0, z = 0.0):
        self.x = x
        self.y = y
        self.z = z
        self.name = None

    def __str__(self):
        return 'x, {}, y, {}, z, {}'.format(self.x, self.y, self.z)

    def np_matrix(self):
        # create a row vector from the point object
        return np.matrix([[self.x], [self.y], [self.z]])

    def give_name(self, name):
        self.name = name

class Printer:

    def __init__(self, transporter):
        self.transporter = transporter
        self.m3d = M3D.M3D()

    def __str__(self):
        return "current nozzlePosition is: {}".format(self.get_nozzlePosition())

    def get_nozzlePosition(self):
        M114 = self.m3d.push('M114')
        point = Point(M114.x, M114.y, M114.z)
        if debug:
            cf = currentframe()
            print('line no: {}; current position expressed in printer coordinates:
            {}'.format(cf.f_lineno, point))
        return point

    def goHome(self):
        self.m3d.push('G28')
        self.get_nozzlePosition()
        if debug:
            cf = currentframe()
            print('line no: {}'.format(cf.f_lineno))

    def move(self, X, Y):
        self.m3d.push('G91')                    #setting relative movement
        self.m3d.push('G0 X%f Y%f F1800.0' % (X, Y))
        # force movement above to be finished
        self.get_nozzlePosition()

    def adjust_height(self, Z):
        self.m3d.push('G90')                    #setting absolute movement
        self.m3d.push('G0 Z%f F90.0' % (Z))
        self.get_nozzlePosition()

    def goTo(self, X, Y):
        self.m3d.push('G90')                    #setting absolute movement
        self.m3d.push('G0 X%f Y%f F1800.0' % (X, Y))
        self.get_nozzlePosition()
```

```
    def draw(self):
        self.m3d.push('G91')                    #setting relative movement
        self.m3d.push('G0 Z-2.5 F90.0')         #lower the pen
        self.get_nozzlePosition()
        time.sleep(3)
        self.m3d.push('G0 Z2.5 F90.0')          #raise the pen
        self.get_nozzlePosition()

    def circle(self, offsetPoint, compensationPoint, RADIUS):
        xComp = compensationPoint.z # different x/y/z here is intentional
        yComp = compensationPoint.x
        #go to position where the angle is zero in unit circle
        self.goTo((offsetPoint.x + RADIUS + xComp), (offsetPoint.y + yComp))
        self.m3d.push('G91')                    #setting relative movement
        self.m3d.push('G0 Z-2.5 F90.0')         #lower the pen
        self.get_nozzlePosition()               #making sure to let operation finish
        time.sleep(1)
        #Move in a circle counter clockwise
        for i in range(0, 361, 6):
            angle = math.radians(i)
            xNext = offsetPoint.x + (RADIUS*math.cos(angle)) + xComp
            yNext = offsetPoint.y + (RADIUS*math.sin(angle)) + yComp
         self.goTo(xNext, yNext)
        self.m3d.push('G91')
        self.m3d.push('G0 Z2.5 F90.0')          #raise the pen
        self.get_nozzlePosition()               #making sure to let operation finish

class Communicator_UDP:

    def __init__(self):
        self.distanceMessenger = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.positionMessenger = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.IP = ipBlue            # = ip of computer running blue.py
        self.IP2 = ipTransporter    # = ip of computer performing transport (eg Pi@Bot)
        self.DISTANCE_PORT = 65432
        self.POSITION_PORT = 65433
        self.HEADER = 1024
        self.DIST_ADDR = (self.IP, self.DISTANCE_PORT)
        self.POS_ADDR = (self.IP, self.POSITION_PORT)
        self.DIST_ADDR2 = (self.IP2, self.DISTANCE_PORT)
        self.POS_ADDR2 = (self.IP2, self.POSITION_PORT)

        self.distanceMessenger.bind(self.DIST_ADDR)
        self.distanceMessenger.settimeout(5)

        self.positionMessenger.bind(self.POS_ADDR)
        self.positionMessenger.settimeout(5)

    def __str__(self):
        return 'Running blue.py: {}; Performing transport: {}'.format(IP, IP2)

    def track_position(self):
        try:
            self.positionMessenger.setblocking(False)
            while True:
                # Throw away buffered data
                print(self.positionMessenger.recvfrom(self.HEADER))
        except BlockingIOError:
            self.positionMessenger.setblocking(True)
        try:
            bytesPair = self.positionMessenger.recvfrom(self.HEADER)
        except socket.timeout:
            return False
```

```
        position = bytesPair[0]
        splitPos = position.strip(b'\x00').split(b",")
        xPos = float(splitPos[1])
        yPos = float(splitPos[3])
        zPos = float(splitPos[5])
        posPoint = Point(xPos, yPos, zPos)
        if debug :
            cf = currentframe()
            print('line no: {}; Tracking position... Timestamp: {}; posPoint:
            {}'.format(cf.f_lineno, time.time(), posPoint))
        return posPoint

    def send_UDP(self, positionPoint, distancePoint):
        #blue.py acting as client, mimic what laser tracker does:
        send two byte messages with coordinates
        posMessenger = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        distMessenger = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        if debug :
            cf = currentframe()
            print('line no: {}; sending UDP...'.format(cf.f_lineno))
        posMsg = b'X,%f,Y,%f,Z,%f' %(positionPoint.x, positionPoint.y, positionPoint.z)
        distMsg = b'X,%f,Y,%f,Z,%f' %(distancePoint.x, distancePoint.y, distancePoint.z)
        try:
            posMessenger.sendto(posMsg, self.POS_ADDR2)
            distMessenger.sendto(distMsg, self.DIST_ADDR2)
            if debug:
                cf = currentframe()
                print('line no: {}'.format(cf.f_lineno))
                print('sent position: ' + str(posMsg))
                print('sent distance: ' + str(distMsg))
        finally:
            posMessenger.close()
            distMessenger.close()
            if debug:
                cf = currentframe()
                print('line no: {}; stopped sending UDP'.format(cf.f_lineno))

class Blue_queen:

    def __init__(self, TOLERANCE, RADIUS, offsetPoint, compensationPoint):
        self.TOLERANCE = TOLERANCE      # [mm]
        self.RADIUS = RADIUS            # [mm]
        #self.printer = Printer(Transporter()) # Transporter not needed for TurtleBot
        and not used for now
        self.communicator = Communicator_UDP()
        self.rotationAngle = 0.0        # [radians], according to spec minAngle
        = -pi/6 rad (-30 deg) maxAngle = pi/6 rad (30 deg)
        self.offsetPoint = offsetPoint  # expressed in printer coordinates
        self.compensationPoint = compensationPoint  # expressed in laser coordinates
        self.position = None            # expressed in laser coordinates
        self.target = None              # expressed in laser coordinates
        self.pointList = [ ]            # stores all bluelining points for the job
        self.transporter_should_move = False
        self.cond_position = threading.Condition()
        position_task = threading.Thread(target = self.communication_task)
        position_task.daemon = True
        # Start position listening task
        position_task.start()

    def __str__(self):
        return 'current target: {}'.format(len(self.target))
```

```
    #returns a point with distances expressed in printer coordinates to laserPoint
    counting from the reflectorPosition, based on current rotationAngle

    def calculate_printer_movement(self, laserPoint, rotationAngle, reflectorPosition):
        translationVector = reflectorPosition.np_matrix() #treated as a local origo
        laserVector = np.matrix([[laserPoint.np_matrix()],
                                 [1]])
        rotationMatrix = np.matrix(
                        [[math.sin(rotationAngle), 0.0, math.cos(rotationAngle)],
                        [math.cos(rotationAngle), 0.0,    -math.sin(rotationAngle)],
                        [0.0, 1.0, 0.0]])
        if debug:
            cf = currentframe()
            print('line no: {}; rotationMatrix:
            \n{}'.format(cf.f_lineno, rotationMatrix))
        empty = np.matrix([0.0, 0.0, 0.0])
        dummy = np.matrix([1.0])
        transformationMatrix = np.matrix(
                    [[rotationMatrix, -rotationMatrix*translationVector],
                    [empty, dummy]])
        if debug:
            cf = currentframe()
            print('line no: {}; transformationMatrix: \n{}'.format(cf.f_lineno,
            transformationMatrix))
        transformedVector = transformationMatrix*laserVector
        if debug:
            cf = currentframe()
            print('line no: {}; transfVector:
            \n{}'.format(cf.f_lineno, transformedVector))
        pointVector = transformedVector.item(0,0)
        x = pointVector.item(0,0)
        y = pointVector.item(1,0)
        z = pointVector.item(2,0)
        printerPoint = Point(x, y, z)
        if debug:
            cf = currentframe()
            print('line no: {}; printerPoint: {}'.format(cf.f_lineno, printerPoint))
        return printerPoint

    def wrong_calculate_compensation_for_tilt(self, laserPoint, rotationAngle):
        x = math.sin(rotationAngle)*laserPoint.x + math.cos(rotationAngle)*laserPoint.z
        y = math.cos(rotationAngle)*laserPoint.x - math.sin(rotationAngle)*laserPoint.z
        z = 0
        printerPoint = Point(x,y,z)
        return printerPoint

    def calculate_compensation_for_tilt(self, laserPoint, rotationAngle):
        pX = laserPoint.z
        pY = laserPoint.x
        pXprime = pX / math.cos(rotationAngle)
        pYprime = pY / math.cos(rotationAngle)
        z = 0
        printerPoint = Point(pXprime, pYprime, z)
        return printerPoint

    def find_rotation(self):
        self.printer.goTo(self.offsetPoint.x, self.offsetPoint.y)
        p1 = self.get_position()
        p1 = self.get_position()
        p1 = self.get_position()
        # move maximum reachable distance in negative x
        self.printer.move(-self.RADIUS, 0)
        p2 =  self.get_position()
        p2 =  self.get_position()
```

```
        p2 =  self.get_position()
        # move maximum reachable distance in positive x
        self.printer.move(2*self.RADIUS, 0)
        p3 =  self.get_position()
        p3 =  self.get_position()
        p3 =  self.get_position()
        # move back to offset point, where printer can reach RADIUS in x and y
        self.printer.goTo(self.offsetPoint.x, self.offsetPoint.y)
        p4 =  self.get_position()
        p4 =  self.get_position()
        p4 =  self.get_position()

        distX = p3.x - p2.x
        distZ = p3.z - p2.z
        self.rotationAngle = math.atan(distX/distZ) #[rad]
        if debug:
            cf = currentframe()
            print('line no: {}; p2: {}; p3: {}; rotationAngle: {} rad
            (={})'.format(cf.f_lineno, p2, p3, self.rotationAngle,
            math.degrees(self.rotationAngle)))
        return self.rotationAngle

    def calculate_distance_to_target(self, trackedPoint=None):
        if trackedPoint == None:
            trackedPoint = self.get_position()
        position = trackedPoint.np_matrix()
        target = self.target.np_matrix()
        deltaVector = target - position
        if debug:
            cf = currentframe()
            print('line no: {}; positionMatrix: {}; targetMatrix: {}; distance:
            {}'.format(cf.f_lineno, position, target, deltaVector))
        distPoint = Point(deltaVector.item(0, 0), deltaVector.item(1, 0),
                          deltaVector.item(2, 0))
        return distPoint

    def get_position(self):
        with self.cond_position:
            self.cond_position.wait()
            point = self.position
        return point

    def communication_task(self):
        while True:
            p = self.communicator.track_position()
            with self.cond_position:
                self.position = p
                self.cond_position.notify_all()
                should_move = self.transporter_should_move
            # Inform users waiting for position update
            print(should_move)
            if should_move:
                delta = self.calculate_distance_to_target(p)
            else:
                delta = Point()
            print(delta)
            self.communicator.send_UDP(self.position, delta)

    def start_transporter(self, on):
        with self.cond_position:
            self.transporter_should_move = on

    def calibrate_pen(self):
        self.printer.goHome()
```

```
        self.printer.adjust_height(self.offsetPoint.z)
        self.printer.goTo(self.offsetPoint.x, self.offsetPoint.y)

    def setup(self):
        self.create_job()
        self.distribute_target()
        self.calibrate_pen()

    def create_job(self):
        file_object = open(textFile, "r")
        job = file_object.readlines()
        for row in job:
            splitRow = row.split(",")
            name = str(splitRow[0])
            x = float(splitRow[1])
            y = float(splitRow[2])
            z = float(splitRow[3])
            point = Point(x, y, z)
            point.give_name(name)
            self.pointList.append(point)
        if debug:
            cf = currentframe()
            print('line no: {}; {} points created in list'.format(cf.f_lineno,
            len(self.pointList)))
            for point in self.pointList:
                print('name: {}; coordinates: {}'.format(point.name, point))

    def distribute_target(self):
        #remove the first point object from pointList and set it as target
        self.target = self.pointList.pop(0)
        if debug:
            cf = currentframe()
            print('line no: {}; distribution done. {} points remaining;
            Target: {}'.format(cf.f_lineno, len(self.pointList), self.target))

    def accuracy_printer_Xaxis(self):
        self.printer.goTo(self.offsetPoint.x, self.offsetPoint.y)
        printerPoint1 = self.printer.get_nozzlePosition()
        laserPoint1 = self.get_position()
        laserPoint1 = self.get_position()
        laserPoint1 = self.get_position()
        self.printer.move(-self.RADIUS, 0.0)
        printerPoint2 = self.printer.get_nozzlePosition()
        laserPoint2 = self.get_position()
        laserPoint2 = self.get_position()
        laserPoint2 = self.get_position()
        self.printer.move((self.RADIUS)/2, 0.0)
        printerPoint3 = self.printer.get_nozzlePosition()
        laserPoint3 = self.get_position()
        laserPoint3 = self.get_position()
        laserPoint3 = self.get_position()
        self.printer.move(self.RADIUS, 0.0)
        printerPoint4 = self.printer.get_nozzlePosition()
        laserPoint4 = self.get_position()
        laserPoint4 = self.get_position()
        laserPoint4 = self.get_position()
        self.printer.move((self.RADIUS)/2, 0.0)
        printerPoint5 = self.printer.get_nozzlePosition()
        laserPoint5 = self.get_position()
        laserPoint5 = self.get_position()
        laserPoint5 = self.get_position()
        b = np.matrix([[laserPoint1.z], [laserPoint2.z], [laserPoint3.z],
                    [laserPoint4.z], [laserPoint5.z]])
        A = np.matrix([[printerPoint1.x, 1.0], [printerPoint2.x, 1.0],
```

```
                    [printerPoint3.x, 1.0], [printerPoint4.x, 1.0],
                    [printerPoint5.x, 1.0]])  # 5x2 matrix
        x,residuals,rank,s = np.linalg.lstsq(A, b, rcond=-1)
        k = x.item(0,0)
        m = x.item(1,0)
        print('residuals: {}'.format(residuals))

def test_transporter(q):
    q.setup()
    q.find_rotation()
    n = len(q.pointList)
    for i in range(n):
        while True:
            p = q.get_position()
            dist = q.calculate_distance_to_target(p)
            calc = q.calculate_printer_movement(
                                q.target, q.rotationAngle, q.get_position())
            error = math.sqrt(dist.x**2+dist.z**2)
            print("calc: ", calc)
            time.sleep(10)
            if error <= RADIUS:
                q.start_transporter(False)
                break
            else:
                q.start_transporter(True)
        print("Next point")
        q.distribute_target()
    while True:
        print(q.get_position())

def bluline_only_transporter(q):
    #required to comment out that the Printer is created in the blue queen init,
    don't forget to change back if running with printer connected
    q.create_job()
    q.distribute_target()
    n = len(q.pointList)
    for i in range(n):
        while True:
            #get_position repeated to make sure an updated value is used
            position = q.get_position()
            position = q.get_position()
            position = q.get_position()
            dist = q.calculate_distance_to_target(position)
            error = math.sqrt(dist.x**2+dist.z**2)
            if error <= RADIUS:
                q.start_transporter(False)
                print("arrived at target")
                time.sleep(10)
                with open('output.txt', 'a') as text_file:
                    print('target name: {}; target coord: {}; \n current position: {};
                    distance(radius) to target: {} \n \n'
                    .format(q.target.name, q.target, position, error), file=text_file)
                break
            else:
                q.start_transporter(True)
        q.distribute_target()
    #move away from last point
    pos = q.get_position()
    stopPos = Point(pos.x + 500, 0, pos.z + 500)
    q.target = stopPos
    #keep sending positions to turtlebot not to get communication error
    while True:
        print(q.get_position())
```

```
def test_UDP(q):
    while True:
        positionPoint = q.get_position()
        print('Time: {}; Position: {}'.format(time.time(), positionPoint))

def test_calc_printer_motion(q):
    target = Point(4000,0,1000)
    q.printer.goHome()
    q.find_rotation()
    printerMotion = q.calculate_printer_movement(target,
    q.rotationAngle, q.get_position())
    print("distance to move: ", printerMotion)

def perform_bluelining(q):
    q.setup()
    n = len(q.pointList)
    for i in range(n):
        while True:
            #get_position repeated to make sure an updated value is used
            position = q.get_position()
            position = q.get_position()
            position = q.get_position()
            dist = q.calculate_distance_to_target(position)
            error = math.sqrt(dist.x**2+dist.z**2)
            if error > RADIUS:
                #printer cannot reach
                q.start_transporter(True)
            else:
                #not close enough, need to fine tune
                q.start_transporter(False)
                while True:
                    motion = q.calculate_printer_movement(q.target,
                    q.rotationAngle, q.get_position())
                    q.printer.move(motion.x, motion.y)
                    position = q.get_position()
                    position = q.get_position()
                    position = q.get_position()
                    dist = q.calculate_distance_to_target(position)
                    error = math.sqrt(dist.x**2+dist.z**2)
                    if error <= TOLERANCE:
                        #close enough to draw()
                        #to check that values are within possible local coordinates
                        penPosition = q.printer.get_nozzlePosition()
                        #taking the change to local frame into account
                        q.printer.move(q.compensationPoint.z, q.compensationPoint.x)
                        printedPosition = q.printer.get_nozzlePosition()
                        q.printer.draw()
                        with open('output.txt', 'a') as text_file: #a means append
                            print('targetName: {}; angle: {}; penPosBeforeComp: {};
                            penPosAfterComp: {} \n \n'.format(q.target.name,
                            math.degrees(q.rotationAngle), penPosition, printedPosition),
                            file=text_file)
                        q.printer.goTo(q.offsetPoint.x, q.offsetPoint.y)
                        q.distribute_target()
                        break
    #move away from last point
    pos = q.get_position()
    stopPos = Point(pos.x + 500, 0, pos.z + 500)
    q.target = stopPos
    print('bluelining done')
    #keep sending positions to turtlebot not to get communication error
    while True:
        q.get_position()
```

```
def test_repeatability(q):
    for i in range(5):
        q.printer.goTo(45, 85)
        compensationX = q.compensationPoint.z
        compensationY = q.compensationPoint.x
        targetX = 70
        targetY = 40
        q.printer.goTo(targetX+compensationX, targetY+compensationY)
        q.printer.draw()

if __name__ == '__main__':

    offsetPoint = Point(OFFSETX, OFFSETY, OFFSETZ)
    compensationPoint = Point(COMPX, COMPY, COMPZ)
    q = Blue_queen(TOLERANCE, RADIUS, offsetPoint, compensationPoint)
    perform_bluelining(q)

pass
```

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
March 2021

Document Number
TFRT-6123

Author(s)
Lisa Klinghav

Supervisor
Alina Andersson, MAX IV Laboratory, Sweden
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Tore Hägglund, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Mobile Floor-Marking Robot, utilizing Feedback from Laser Tracker

Abstract

Equipment installation at MAX IV, a state-of-the-art particle accelerator in Sweden, requires accuracy and precision in the range of micrometers. The measurement technology used is laser tracking, with which the position of a movable reflector is determined, by reflecting the laser beam emitted from a stationary tracker. Data describing the real-time position of the reflector is handled in a software called SpatialAnalyzer, managing it within the point cloud of statistically secured reference nodes, located all around the facility. The laser tracker is operated via a PC, whereas adjustment of the position of the reflector-equipped inventory is a manual, sometimes trying exercise, performed by measurement engineers. If this task were to be automated, it would free up time and resources, as well as improve the working condition for the employees.

The purpose of this master's thesis was to develop a mobile floor-marking robot to performbluelining atMAXIV. Bluelining is the first step of equipment installation and it is a process of projecting reference points from a 3D-model to a drawing on the floor. In this step, for example, drill marks for bolts of a foundation of an optic table are to be drawn. A 1,5 mm radius marker-pen is used and a guiding tool, holding the reflector, is positioned by hand to mark the target.

The proposed design consists of a modified 3D-printer attached to a mobile robot equipped with omni wheels. It is controlled via a main program that runs on a Raspberry Pi and it is powered via rechargeable batteries. Performancewas evaluated in case studies and the average deviation from target achieved was six millimeters. As the testing was brief, it is recommended to perform further activities to define accuracy, precision and repeatability. However, the indications are that it is possible to perform automatic bluelining with modified off-the-shelf products, receiving positional coordinates from a laser tracker system. The result could be further improved with a custom-made robot, preventing compromises in its automation and design.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-57

Recipient's notes

Security classification

http://www.control.lth.se/publications/