Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2018

MSc THESIS

Optimising Motor Control in Actuator Alignment

Matthijs Geers

CE-MS-2018-13

Abstract

The automation of replanting seedlings into bigger trays (transplanting) has been a major industrialisation step in the horticultural sector. Modern machines are abundant in large companies and are quite effective, but they are very expensive both in purchase and maintenance. With major clients taking these costs for granted, designers have not stopped to consider ways to alleviate them. This work introduces an affordable electronic system that makes transplanters wireless, allowing for hot-swappable actuators and thus greatly reducing the cost of technicians and downtime as well as procurement. The expensive servo motor drives are replaced by dedicated microcontrollers, allowing the actuators to make their own decisions based on the constraints imposed by various standards of tray sizes and other circumstances. A method is presented to derive the most favourable trajectories, which are then enforced on the system through a closed-loop feedback system. The resulting performance approaches that of the system's predecessor, but at a much more reasonable price. This makes transplanters more affordable for small companies, allowing a broader market to reap the harvest of technological development and increasing worldwide horticultural yield as a result.

Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Optimising Motor Control in Actuator Alignment

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Matthijs Geers
born in Gouda, Netherlands

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Optimising Motor Control in Actuator Alignment

by Matthijs Geers

Abstract

The automation of replanting seedlings into bigger trays (transplanting) has been a major industrialisation step in the horticultural sector. Modern machines are abundant in large companies and are quite effective, but they are very expensive both in purchase and maintenance. With major clients taking these costs for granted, designers have not stopped to consider ways to alleviate them. This work introduces an affordable electronic system that makes transplanters wireless, allowing for hot-swappable actuators and thus greatly reducing the cost of technicians and downtime as well as procurement. The expensive servo motor drives are replaced by dedicated microcontrollers, allowing the actuators to make their own decisions based on the constraints imposed by various standards of tray sizes and other circumstances. A method is presented to derive the most favourable trajectories, which are then enforced on the system through a closed-loop feedback system. The resulting performance approaches that of the system's predecessor, but at a much more reasonable price. This makes transplanters more affordable for small companies, allowing a broader market to reap the harvest of technological development and increasing worldwide horticultural yield as a result.

Laboratory	:	Computer Engineering
Codenumber	:	CE-MS-2018-13

Contents

List of Figures **vi**

List of Tables **vii**

List of Acronyms **ix**

Acknowledgements **xi**

1 Introduction **1**
- 1.1 Background 1
- 1.2 The Transplanter 2
  - 1.2.1 Principle of operation 3
  - 1.2.2 Weak points 3
- 1.3 Research questions 4
- 1.4 Thesis outline 4

2 Requirements **5**
- 2.1 Motor requirements 5
- 2.2 Sensor requirements 7
  - 2.2.1 Motor proximity sensors 7
  - 2.2.2 Lateral position sensors 8
- 2.3 Communication requirements 8
  - 2.3.1 Calibration phase 9
  - 2.3.2 Operation phase 9
- 2.4 Error detection 9

3 Design **11**
- 3.1 Motor 11
  - 3.1.1 Stepper motors 12
  - 3.1.2 Brushed DC motors 14
  - 3.1.3 Brushless DC motors 14
  - 3.1.4 Comparison 15
- 3.2 Communication 16
  - 3.2.1 Calibration 16
  - 3.2.2 Full operational bus communication 18
  - 3.2.3 Limited bus communication 20
  - 3.2.4 Wireless communication 21
- 3.3 Trajectory generation 21
  - 3.3.1 Parabola shape 22
  - 3.3.2 Trapezoid shape 23

3.3.3 Splines 24
3.4 Motion control 26
3.4.1 PID control 27
3.4.2 Setpoint selection 28

4 System Implementation **29**
4.1 Hardware 29
4.1.1 Motor drive 29
4.1.2 Gearing 30
4.1.3 Motor type 32
4.1.4 Microcontrollers 33
4.2 Main controller 34
4.2.1 PLC communication 35
4.2.2 Motor controller communication 37
4.2.3 Communication scenario: calibration 38
4.2.4 Communication scenario: normal operation 39
4.3 Motor controller 40
4.3.1 Motor control 41
4.3.2 Trajectories 44

5 Results **45**
5.1 Communication 45
5.2 Trajectories 46

6 Conclusions & recommendations **49**
6.1 Conclusions 49
6.2 Contributions 50
6.3 Recommendations for future work 51
6.3.1 Calibration 51
6.3.2 Motor control 51

Bibliography **54**

List of Figures

1.1 Typical seedling trays . . . 2
1.2 Interior of the current Transplanter, with a close-up of a motor on the right . . . 3

2.1 Different types of grippers used . . . 7

3.1 Servo drive hardware used in the old Transplanter . . . 11
3.2 Cross-sections of a three-stack variable reluctance stepper motor, adapted from [1] . . . 12
3.3 VR stepper motor phase input currents . . . 13
3.4 Cross-sections of a hybrid stepper motor, adapted from [1] . . . 13
3.5 BLDC commutation diagram . . . 15
3.6 Comparison of characteristics of Direct Current (DC) motors, Variable Reluctance (VR) steppers and hybrid steppers . . . 16
3.7 DRAKON diagram of the combined calibration method . . . 19
3.8 Clockwise from left: examples of position, velocity and acceleration in a 5th order parabola trajectory . . . 23
3.9 Clockwise from left: examples of position, velocity and acceleration in a trapezoid trajectory, with slopes coloured blue and cruise velocity coloured orange . . . 24
3.10 Clockwise from left: examples of position, velocity and acceleration in a third order spline trajectory through predetermined points (denoted by dots) . . . 26
3.11 Block diagram of a system with closed-loop control . . . 27
3.12 Block diagram of a Proportional, Integral, Derivative (PID) controller . 28

4.1 Equivalent circuit of an electric motor phase winding . . . 29
4.2 Metal Oxide Semiconductor Field Effect Transistor (MOSFET) full-bridge connected to motor phase . . . 30
4.3 Mechanical schematic of the gear systems on the motor shaft . . . 31
4.4 Top level diagram of the control hierarchy . . . 34
4.5 Main controller embedded in the Transplanter . . . 35
4.6 Sequence diagram of principal system operation . . . 36
4.7 Finite State Machine (FSM) of the calibration process . . . 39
4.8 Set of mounted grippers without casing . . . 41
4.9 Step response of the model, showing its start-up characteristic, with displacement in encoder pulses on the y-axis . . . 42
4.10 Measured step response used to estimate the transfer function of the system, with displacement in encoder pulses on the y-axis. The step response of the resulting transfer function is plotted over the measured data . . . 43

5.1 Overview of the new Transplanter, with cases mounted on gripper electronics and a close-up of magnet mounting on the right 45
5.2 Measurements of the planned trajectory (orange) and actual trajectory (blue) in the top figure, and the expected velocity (orange) and actual velocity (blue) in the bottom figure 47
5.3 Measurements of the planned trajectory (orange) and actual trajectory (blue) in the top figure, and the expected velocity (orange) and actual velocity (blue) in the bottom figure 48

List of Tables

2.1 Motor requirements 6
2.2 Mass of different loads involved in the system 6
2.3 Proximity sensor candidates 8

3.1 Numerical comparison of different motors 16
3.2 Trajectory type candidates, with order in brackets where applicable . . . 26

4.1 Numerical comparison of different motors given that $v_{max} = 2$ m/s and $a_{max} = 4$ m/s^2 must hold; an extension of Table 3.1 32
4.2 List of calibration conditions 39
4.3 Message Queuing Interface usage example 40
4.4 Parabolic trajectory as measured after on-chip generation 44

List of Acronyms

PLC Programmable Logic Controller

PID Proportional, Integral, Derivative

EEPROM Electrically Erasable Programmable Read-Only Memory

RPM Rounds Per Minute

EMF Electromotive Force

VR Variable Reluctance

PWM Pulse Width Modulation

DC Direct Current

PCB Printed Circuit Board

BLDC Brushless Direct Current

I/O Input/Output

TCP/IP Transmission Control Protocol/Internet Protocol

OSI Open Systems Interconnection

FSM Finite State Machine

MOSFET Metal Oxide Semiconductor Field Effect Transistor

Acknowledgements

First and foremost, I would like to thank the members of my thesis committee. Not just for being part of this committee, but also for having been among the most inspiring lecturers I have had the privilege of being taught by over the course of my studies. I thank Arjan in particular for providing me with constant feedback acting as my project supervisor.

Needless to say, I would also like to thank Jasper for offering me the chance of participating in an exciting project at Inventeers. I am thankful for the friendly ambience and flat hierarchy that were made possible by all my colleagues at Inventeers. I would like to specifically thank Thomas for tirelessly cooperating with me on this project, Gijs for being a coach during difficult times, Erik for sharing his boundless knowledge on all things embedded programming, Leon for never ceasing to express his passion for privacy and open source and Bram for always being a willing sparring partner and, more generally, a goofball.

I would like to thank my friends, who remained by my side despite my continued unavailability at some points during the project (especially Rick for proofreading my mechanical equations). I also thank my brother Robbert, who has kept taking time out of his incredibly busy schedule to show interest in me and my project. Last but not least, I would like to thank my parents, without whose unrelenting support I certainly would not have made it this far.

Matthijs Geers
Leiden, The Netherlands
July 8, 2018

Introduction 1

This thesis was written to describe a project carried out for Inventeers, an independent R&D company settled in Leiden that operates business to business; its customers are companies that lack the expertise to carry out certain investigations and product designs. Inventeers is hereafter referred to as *the company*. The customer that gave the order for this project is called Visser Horti Systems, a company that specialises in the design and production of horticultural machinery. It is settled in 's-Gravendeel and will, in turn, be referred to as *the customer* from here on.

In this chapter, the project synopsis is described. Some background information is given about the state-of-the-art and the sector in which it operates, followed by a detailed description of where it is lacking and the projected improvements. Consequently, relevant research questions are posed that originate from these improvements.

1.1 Background

Ever since the Industrial Revolution about 150 years ago, the agricultural sector has gone through a rapid succession of technological developments. The invention of the steam engine shifted the sector from manual labour to high-power automation, increasing production by orders of magnitude. Manual tools like flails, hoes and shovels were replaced by steam-driven counterparts and even horses in front of ploughs were exchanged for primitive tractors. Further development was heralded by the rise of the combustion engine and, more recently, the electric motor. Applications of electric motors were relatively straightforward at first, driven through simple combinations of analogue electronics and sensors.

The dawn of the Information Age can easily be considered the next scientific revolution in history. The digitisation of computers and information in general unlocked the possibility to gather enormous amounts of data and unleash complicated mathematical theories on them. The past few decades and the past few years in particular (considering the decline of Moore's Law) have been a stage for rapid developments in software and digital sensors and actuators. These developments have pushed the boundaries to such an extent that, in many fields of science, so much information is now being generated that the real challenge is to figure out how to extract useful information from it to begin with.

Greenhouses have long been equipped with sensors, and maintaining a constant climate is hardly a novelty, but due to the cramped nature of a greenhouse in addition to the increased vulnerability of its plants, large-scale automation has proven difficult. Until recently, as the abundance of information brought in by extensive sensor networks has been of increasing benefit to the horticultural sector. Instead of regulating greenhouse-wide climate conditions, the physical conditions of subsets of the greenhouse or even of individual plants can now be used to pinpoint problems that might otherwise have gone unnoticed. Furthermore, labour-intensive processes that could formerly only be done by hand might now be outsourced to further automation.

This thesis focuses on one such process and its automation; transplanting or replanting. Transplanting is the process of moving seedlings from their plug trays (Figure 1.1) into larger trays or pots. This is an important process in the life-cycle of almost every plant grown in a greenhouse. Since not every seed makes it through the seedling phase successfully, it involves a great deal of selection, as the output trays or pots must always be filled to capacity with live seedlings. In the past, this selection process in particular absolutely required human intervention, but modern technology has made it possible to automate the entire process.

Figure 1.1: Typical seedling trays

1.2 The Transplanter

The customer has long been one of the leading companies in horticultural automation. One of their cornerstones is the Transplanter, or *Pic-O-Mat*. It consists of a fully automated system that, connected to a conveyor belt of seedling plugs, is able to accurately pick up and replant all of the seedlings from these plugs to trays or pots. An impression of its interior is shown in Figure 1.2. The input feed for one of the conveyor belts can be seen in the lower half of the figure. Above this conveyor belt is a wide bar that has a number of actuators attached to it, one of which is shown up close in the picture on the right. As can be seen, this actuator consists of a motor attached to the bar through a rack-and-pinion gear system, mounted with a gripper that can be activated to pick up and deactivated to let go of a seedling, powered by pneumatics. Each actuator is connected to a central hub through thick cable hoses that carry power and control to the motors and air pressure to the grippers.

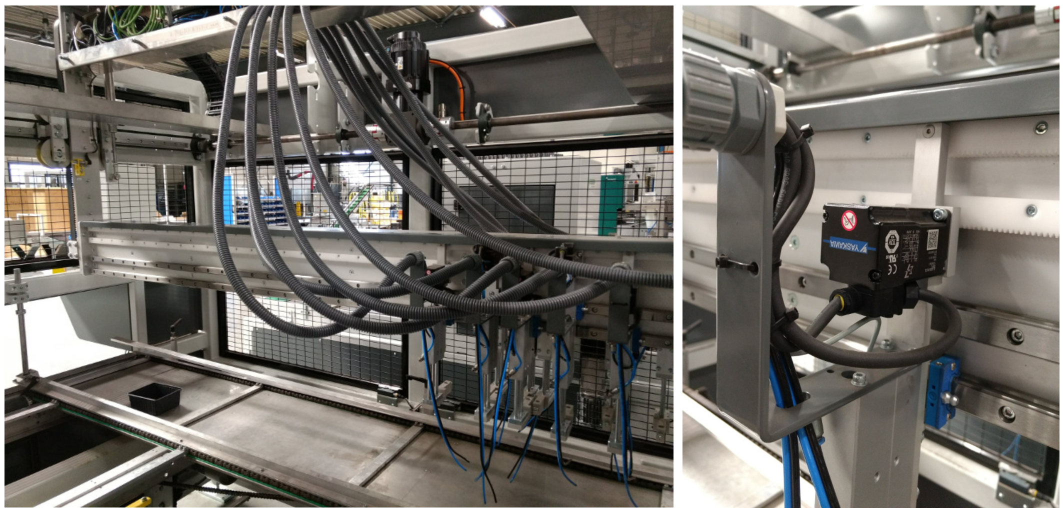


Figure 1.2: Interior of the current Transplanter, with a close-up of a motor on the right

1.2.1 Principle of operation

Important to notice from Figure 1.2 is that the bar along which the grippers can move also has two degrees of freedom. It can move up and down, as the grippers cannot actually do this themselves, but also back and forth. The latter allows the bar to switch between two different conveyor belts; one for the input plugs and one for the output trays or pots.

During a regular cycle of operation, the bar places itself above the input conveyor belt while the motors converge into a very dense formation. Shortly after this process, the bar is moved down, pressing the grippers into the plugs and simultaneously activating them to pick up their plants. The bar is then moved up again, and moves to the output conveyor belt while the motors realign into a sparser configuration, suitable for the greater distance between the destinations of the different grippers. As soon as they get there, the bar is once again pressed down, deactivating the grippers and depositing the plants. This is a very rapid process at up to 38000 seedlings per hour, and it repeats until all seedling plugs have been dealt with.

1.2.2 Weak points

As the main goal was to make the machine cheaper, its weak points were identified in terms of potential cost improvement. This led to two main weak points; its motors and its wired connections. The current machine is equipped with Yaskawa servo motors, which are extremely expensive in purchase as well as maintenance, as the drives used are mandatory and can only be programmed by their Japanese manufacturer. The wired connections of pneumatics and electronic power and control, as were shown in Figure 1.2, are encased in thick hoses that are clumsy and relatively feeble. On top of being prone to wear, the complicated connections make it very difficult for a user to replace an actuator without requiring a technician (costly) and suffering downtime until the repairs have been made (even more costly).

1.3 Research questions

Alleviating the weak points described poses some complicated design problems. Most importantly, the automatic control of a servo system has to be replaced by a micro-controller implementation, requiring generation of optimal trajectories and a way to accurately adhere to them, while guaranteeing that no collisions can occur between the motors involved. Furthermore, the wired connections to the actuators has to be eliminated to allow for modularity and hot-swapping. A main research question was composed in an attempt to solve these problems, along with a number of sub-questions to emphasise specifically relevant or difficult issues. The questions are as follows:

Using electronic control, how can multiple entities with certain inertial characteristics efficiently be moved along a linear rail at high speed without collisions?

Sub-questions:

- *What should an entity's trajectory look like, how is it influenced by the inertia of its payload, and how might this be compensated?*
- *What might the characteristics be of a cost-effective and accurate electronic system implementing this?*
- *In a physical implementation of the system, what might go wrong, and how can this be accounted for?*

1.4 Thesis outline

First off, Chapter 2 relates the research questions to their corresponding hardware and software requirements. Then, Chapter 3 expands on the theoretical approaches to handle these requirements. Chapter 4 goes on to flesh out the implementational details of these approaches, and is followed by Chapter 5, which provides relevant results obtained over the course of the project. Finally, Chapter 6 goes into detail on how these results may be used to answer the research questions, drawing conclusions accordingly and giving recommendations for future work to be done on the project.

2 Requirements

This chapter details the system requirements that result from the customer's desire for more modularity and lower cost. Various trade-offs and physical limitations are introduced and discussed.

2.1 Motor requirements

To determine realistic motor requirements, a number of calculations were done using simple physics formulas. A motor is defined with a rotor with radius r_r attached at the centre to a cogwheel with radius r_c. This cogwheel runs along a rack-and-pinion system. Any deviation in tooth pitch (density) between the rack and the pinion can be modelled by multiplication of this radius. The torque τ on the axis is $\tau = I_{tot} \cdot \alpha$ where I_{tot} is the total rotational inertia of the system, and α its rotational acceleration in rad/s^2. I_{tot} is defined as $I_{tot} = I_r + I_{load}$ where I_r is the rotational inertia of the motor and I_{load} is the rotational moment of inertia of the load moving along the rack-and-pinion system, defined as $I_{load} = \int_Q r^2 dm$ with m the mass of the rotating system and Q its volume. The rotor of the motor is the only part of the system that directly offers rotational inertia; the rest of the system only moves along the rail as a tangential system. Modelling this as a doughnut-shaped mass around the cogwheel with the centre of mass in the middle of the axis transforms it into a rotational mass with a moment of inertia $I_{load} = mr_c^2$. Thus, τ is rewritten as $\tau = I_r\alpha + mr_c^2\alpha$.

The rotational acceleration can be transformed to tangential acceleration a using $a = \alpha r$. This a is different for the cogwheel and the rotor as their radii differ. α, on the other hand, is equal for both rotations, so it suffices to express α in terms of the radius of the cogwheel as shown in Equation 2.1. Combining all of these equations yields Equation 2.2 where $a_{max} = a_c$, as the constraint must match the highest possible acceleration value.

$$a_r = a_c = \alpha r_c \rightarrow \alpha = \frac{a_c}{r_c} \tag{2.1}$$

The velocity of the motor on the rail can be defined as $v = \omega r_c$ with r_c, once again, the radius of the cogwheel mounted on the rail, and $\omega = 2\pi f$ the angular velocity in rad/s, where f is the rotation frequency in rounds per second. Since the relevant performance metric is Rounds Per Minute (RPM), its definition $RPM = 60f$ was combined with the previous equations to form Equation 2.3.

Defined specifically for stepper motors is the maximum resolution attainable from the steps in a motor rotation, with k the micro-stepping factor (further discussed in Section 3.1) and n_{steps} being the number of steps in a full rotation (Equation 2.4). This corresponds to the displacement caused by a single micro-step. Next is the physically

Torque	$\tau \geq I_r \frac{a_{max}}{r_c} + m a_{max} r_c$
RPM	$\geq \frac{30 v_{max}}{\pi r_c}$
Accuracy	$\sim 0.1mm$
Width	$\leq$ 3 cm
N_{motors}	up to 64

Table 2.1: Motor requirements

$$\tau \geq I_r \frac{a_{max}}{r_c} + m a_{max} r_c \tag{2.2}$$

$$RPM_{motor} \geq \frac{30 v_{max}}{\pi r_c} \tag{2.3}$$

$$\text{Resolution} = \frac{2\pi r_c}{k n_{steps}} \tag{2.4}$$

dictated maximum motor width, originating from the minimum plug size (15 mm) the machine must be able to support. In general, this maximum width is 30 mm, but in stepper motors, this width corresponds to the NEMA-11 standard [2], which states that the face of the motor must have a diameter of 1.1 inches, or 2.8 centimetres. Finally, the maximum number of motors as requested by the customer is 64 and the accuracy requested in [3] is ~0.1 mm. This accuracy signifies the minimum distance distinguishable by the system, but it should not be taken very strictly, as the discrepancy caused by the gripping of cogs is likely to cause an inaccuracy larger than this requirement. All motor requirements are listed in Table 2.1.

Increasing r_c yields a higher torque requirement from the motor, as in Equation 2.2, the right hand part of the equation is typically much bigger. This seems illogical, as the torque should remain the same no matter the size of a gearing step, but the constraint is on the acceleration, which relates to force through Newton's Second Law. Therefore, even if the torque remains constant, applying it at a bigger distance decreases the force proportionally. In other words, a larger cogwheel requires a larger torque from the motor to exert the same force on the load. Since r_c is in the denominator of Equation 2.3, this implies a trade-off between torque and RPM. This is further discussed in Section 4.1.2.

Another important thing to note is the weight of the load applied to each motor, as the inertial property of any amount of mass will resist movement actuated by the motors. This resistance puts a constraint on the minimum required torque (as shown in Equation 2.2), thus influencing the system requirements. Some measurements of system parts are listed in Table 2.2. The large and small pneumatic grippers in the table correspond with the largest and smallest grippers in Figure 2.1, which shows the options users will have in selecting a gripper.

Object	Mass (g)	Object	Mass (g)
Large pneumatic gripper	590	PCB plus components	150
Small pneumatic gripper	250	Payload	50
Supporting frame	650	Total (maximum)	~1440

Table 2.2: Mass of different loads involved in the system

Reasoning naively, the worst case scenario is an emergency deceleration from full

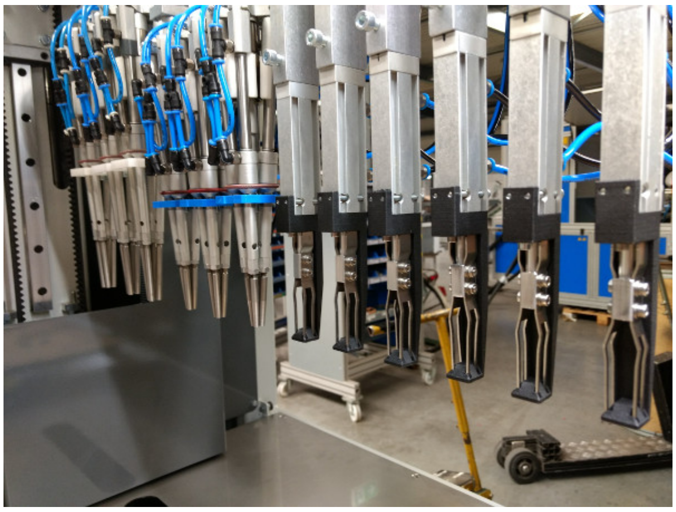


Figure 2.1: Different types of grippers used

speed, so the maximum required torque and maximum required RPM must be attainable simultaneously. However, this does not take intermittent overloading performance into account. Along with these considerations, input parameters remained that were to be determined through various trade-offs during the design phase, including maximum velocity, maximum acceleration, maximum load and gear ratio. This process is described in Chapters 3 and 4.

2.2 Sensor requirements

To assist in initialisation and error detection, the system had to be equipped with certain sensors. As for the motor resolution, the communication protocol between the customer and the company requested a target resolution of 0.1 mm [3]. Therefore, the motors had to be equipped with rotary encoders that are capable of approximately this resolution. Using the feedback from these encoders, a closed-loop feedback controller can be forced on the motors. Aside from the encoders, two types of proximity sensors were proposed; one for proximity detection between the motors, and one to calibrate a motor's absolute lateral position.

2.2.1 Motor proximity sensors

Attaching proximity sensors in between the motors, individual motors would be able to obtain some information about their neighbours. More specifically, they can trigger an emergency response if a neighbour comes dangerously close, thus preventing damage to the motors. Optical sensors work by using the principle of reflection to see how quickly their signal is bounced back on an object. Since they are often flat-faced, they typically fall off towards zero distance. [4] would be a suitable example because its face is dented, allowing its signal to pass through even if the object to be detected is pressed directly against it. The range requirement is dictated by the scenario where a motor at top speed has to stop because it detects a neighbour that is standing still. A range of approximately 1 cm was assumed to suffice.

It is unlikely for an optical sensor to be completely free of false positives due to the circumstances of its environment (dust, leaves, soil). This disadvantage, however, is not present in magnetic sensors, which are otherwise similar to optical proximity sensors in operation. Unfortunately, using magnetic sensors could pose other problems, like difficulties in properly lining up magnets and sensors.

An alternative to optical proximity sensors could be pressed switches. Moving two motors sufficiently close to each other then presses their respective switches on facing sides, which can be used as an immediate trigger to force the motors to a halt. This is more robust than the optical and magnetic alternatives, guaranteeing significantly less false positives if any at all, but at the cost of reaction speed; these sensors are entirely digital, which means that once they are triggered it could already be too late to stop.

Table 2.3 comprehensively lists the up- and downsides of the different types of sensors, showing that magnetic sensors are likely to be the best option.

	Dynamics/Range	False positives
Optical	+	+/-
Magnetic	++	+
Switch	- -	++

Table 2.3: Proximity sensor candidates

2.2.2 Lateral position sensors

Fitting the rail with magnets and the motors with magnetic proximity sensors gives motors a reference frame of coordinates from which they can obtain their absolute positions; every time a motor passes by a magnet on the rail, its sensor will trigger, allowing for an instant, accurate recalibration from a hard-coded look-up table. These sensors must not give any false positives, and must have a range large enough to detect magnets on the rail which, considering their physical proximity, should not be a problem. Taking price into consideration, [5] was expected to suffice for the magnetic sensors.

These magnetic sensors could be set up to detect a flip in polarity. This way, it would suffice to equip each motor with a single sensor, using two magnets whose polarity is aligned with the rail in opposing orientation. In this manner, the sensor could distinguish the two magnets by observing which way the detected polarity flips as it passes by a magnet. However, both magnets would have to fall within the range of every motor on the rail for this to work. If supplied with knowledge of the whereabouts of both magnets, the motors could then determine their dynamic response as well as their total range within the rail.

2.3 Communication requirements

Though the system proposed consists of a master controller with many slave motors, the master is in turn the slave of a Programmable Logic Controller (PLC) controller operated by the customer. As explained in Section 1.2, the rail system is merely a part of a larger system with numerous other sensors and actuators. For the purpose of this design, the surrounding system was considered a black box that only supplies timing information and target positions for the motors on the rail, plus possibly a selectivity parameter to activate specific actuators attached to the motors. The rail system was supplied with a Command Queuing Interface to accommodate this functionality [3]. This interface allows the main controller to store any commands it receives from the PLC in a dependency order specified by parameters within these commands. These commands can then, at their specified times, be spread amongst the different motor controllers as deemed necessary for proper execution.

2.3.1 Calibration phase

By performing a calibration sequence, the main controller can gather information on the motors it is to control. The goal of this calibration phase was to create a list of all motors and their positions on the rail, possibly using the sensors described in Section 2.2. Even with these sensors, there remains a communication challenge. If a central controller is considered the master, and the motors on the rail are considered slaves, the master initially knows nothing about its slaves, including how to distinguish them. Methods of remedying this are described in Section 3.2.1.

2.3.2 Operation phase

After calibration is completed, the system is ready to go into the operation phase. This phase is iterative, as explained in Section 1.2, with one iteration completing a full cycle where the actuators are activated, moved to the other side, deactivated and brought back into their starting positions again. At some point during each iteration, the PLC communicates location data for the next iteration to the master controller. Combining this location information and the previous locations of the motors, a trajectory to follow must be computed for every individual motor, and each motor must apply some form of motion control in order to sufficiently reject disturbances with respect to its target trajectory. This is further discussed in Section 3.3.

2.4 Error detection

There are a number of potential problems the system might encounter. Be it due to human factors, material wear, communication errors or something else entirely, the system must be able to deal with any problems accordingly. Shown below is a non-exhaustive list of potential problems and solutions, differentiated by cause. They are structured as follows: *Problem → Effect → Solution.*

Human factors

- Two motors swapped during power off or system pause → Mismatch between Id and order on rail found during calibration → Full recalibration
- Wrong input parameters set through PLC → Wrong motor alignment inputs → System refuses the inputs and reports to the PLC that they are invalid

Material wear

- No start/end magnet detected on the rail → Unable to finish calibration → Stall and send debug data to PLC
- Motor not moving despite being fed pulses → Impending collision with neighbour → Pull down bus for emergency stop, write debug data to Electrically Erasable Programmable Read-Only Memory (EEPROM) to be read by controller after reboot
- True positive from leading optical sensor → Neighbour detected unexpectedly → Slow down motor depending on proximity; report to controller above certain threshold
- True positive from lagging optical sensor → Neighbour detected unexpectedly → Neighbour's optical sensor and something else broken; stall and send debug for emergency stop
- One or more faulty motors on rail swapped out for new one(s) → Id/physical address mismatch → Full recalibration
- Significant change in system transfer due to wear → Mismatch between expected output and actual output → Try to compensate by changing control parameters, else stall system and debug

Communication

- Wrong operation signal length sent by PLC → Mismatch with detected number of motors on rail → Reject command and return debug data
- Motor microcontroller OR sensor not responding in calibration phase → System can't finish first step of calibration → Human verification step through communication status LEDs that light up on receiving a broadcast
- Excessive packet loss → Delay on communication → Increase interval to start after receiving target locations

Other

- Unidentified object stuck in front of optical sensor → False positive when checking for neighbour, system behaviour affected → Detect when a sensor is stuck in positive output, stall system and send debug data to PLC
- Motor not following trajectory properly → Wrong encoder output → Stall and debug (this is hard to differentiate from the case where the encoder output is correct but the trajectory is wrong)
- Motor overheating → Pause required immediately → Stall briefly, send warning to PLC? If persists, full stall and debug (keep track by summing dissipated power over time, reduced by expected cooling)

Design 3

In this chapter, the overall design process is described. Section 3.1 focuses on the motor's design parameters whereas Section 3.2 describes the facets of communication within the system. Section 3.3 discusses the generation of various trajectories, and finally, Section 3.4 expands on the enforcement of those trajectories through motion control.

3.1 Motor

For the purpose of the described design, any motor's relevant performance metrics can be defined as speed, power and accuracy. If the price or one of the performance metrics must be positively impacted, the others are typically negatively impacted. For example, an increase in speed would require a motor that is more expensive, has more power, and/or has less accuracy. Since the goal was to decrease the price, concessions could be made on one or more of the other factors to potentially achieve better results. Sections 3.1.1 through 3.1.3 describe the different types of motors considered and their merits and drawbacks, while Section 4.1.2 goes into more detail regarding the decisions made in gearing.

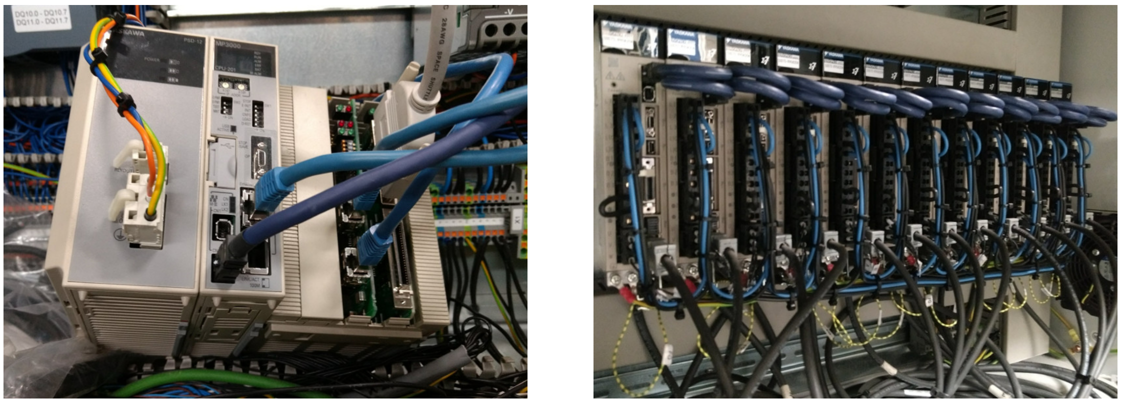


Figure 3.1: Servo drive hardware used in the old Transplanter

The type of motor used was a very important thing to consider. The original machine used servo motors. These can be any type of motor, but the thing that makes a motor a servo is its packaging. Servos are complete products, fully packaged along with controllers, drive systems and rotational sensors. They are easily managed by engineers building simple systems, as they only require digital speed and direction signals to work. A convenient way to make things cheaper would be to control the motors with custom microcontrollers instead. Since the system proposed is made from scratch, complete with custom-built Printed Circuit Board (PCB)s, any necessary controllers and drive circuits can be incorporated into the design with minimal overhead costs. This leaves only the motors and rotational sensors to be decided on, resulting in a significant price drop compared to the top-end servo motors in the original machine. Figure 3.1 shows the amount of hardware required in order to drive the old motors.

3.1.1 Stepper motors

Accuracy being a prime factor, it made sense to use stepper motors. These do not come pre-equipped with rotary sensors or control logic. By default, they are open-loop controlled, meaning they cannot compensate for any abnormalities that might occur during operation. This leaves it up to the designer to decide how to implement the feedback control, depending on what the application requires. As opposed to standard Direct Current (DC) motors, stepper motors are always brushless, operating through toothed rotors and stators. They generally come in two varieties - the Variable Reluctance (VR) stepper and the hybrid stepper, which are discussed separately.

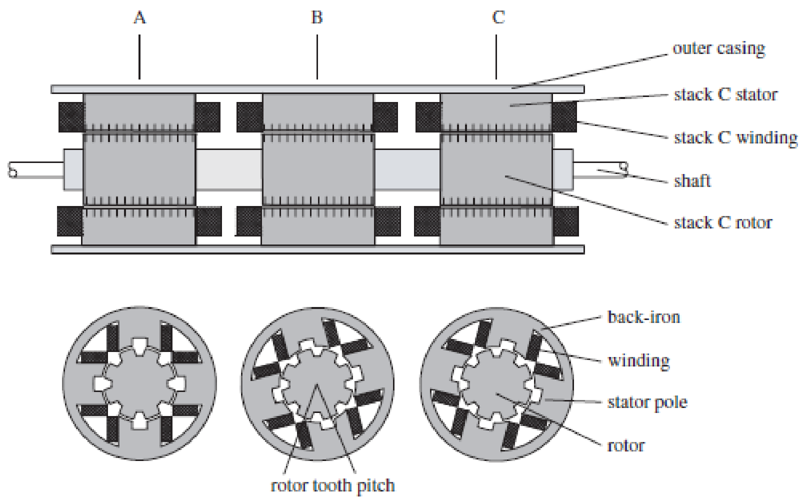


Figure 3.2: Cross-sections of a three-stack variable reluctance stepper motor, adapted from [1]

A diagram of a VR stepper is shown in Figure 3.2. Both its rotor and its stators are made of electrical steel, but only the stators are equipped with windings. As seen in the bottom diagram, the rotor can only ever be aligned with one phase at a time. The windings on each stator are wound the other way around on opposite sides to create positive and negative poles on either side of the rotor. The motor operates by the principle of magnetic reluctance; when current is supplied across a phase winding, the rotor is urged to align with it to achieve a state of minimum reluctance. Phases A, B and C can be powered on in turn to force the rotor to make steps in a specified direction (shown in Figure 3.3), which allows for a step length of $360/Np$ degrees where N is the number of stacks/phases and p is the number of rotor teeth. In the motor displayed in Figure 3.2, this corresponds to a step length of 15°.

An example of a hybrid stepper motor is shown in Figure 3.4. Its rotor is mounted

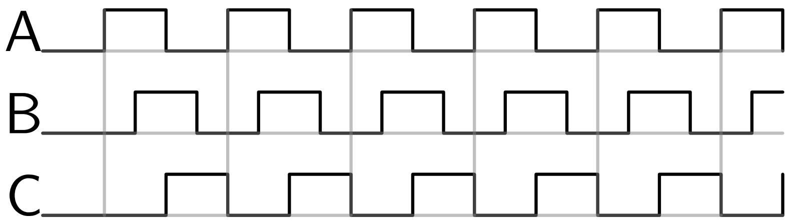


Figure 3.3: VR stepper motor phase input currents

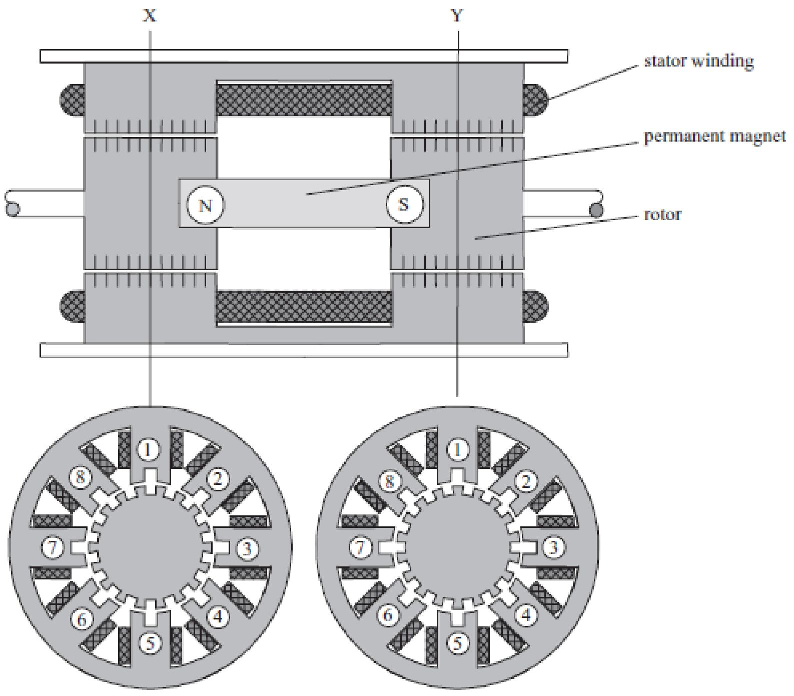


Figure 3.4: Cross-sections of a hybrid stepper motor, adapted from [1]

with a permanent magnet and the windings are on its stator teeth. It operates similarly in turning on phases one at a time to align the rotor in small consecutive steps, but a major difference is the fact that a hybrid stepper motor displays detent torque; even with all windings powered off, the permanent magnet in the rotor will resist movement between step positions. Depending on the application, this can be an advantage or a disadvantage. Like in a VR stepper, opposing stator poles are paired, but in this case they have the same polarity. The ”stacks” of a hybrid stepper do not correspond to its phases as they would in a VR stepper; comparing their figures shows that the windings in a hybrid stepper are connected from X to Y. The phases are, therefore, specific sets of poles. In the example of Figure 3.4, numbers 1, 3, 5 and 7 are phase A while 2, 4, 6 and 8 are phase B. Note that there is an angular offset of half a tooth pitch between X and Y, which corresponds to the opposing polarity of the two ends of the rotor magnet. In the bottom diagram, phase A is energised. Energising phase B here would lead to a tiny clockwise step. The synchronisation between these phases is similar to that in a VR stepper, as was shown in Figure 3.3, except in this case there are only two phases that are activated alternatingly. It takes 4 of these consecutive phase activations to make the rotor advance its angle by a single tooth pitch. Therefore, the step length can be defined as $360/4p$ degrees, with p the number of rotor teeth. In this example, this yields a step length of 5°.

3.1.2 Brushed DC motors

A brushed DC motor is the classical example of an electric motor. It has permanent magnets on its stator and windings on its rotor. These windings can be fed by a simple DC current, the direction of which is continuously alternated by the brushes they are connected through. Turning the motor causes the brushes to move, reversing the polarity of the rotor windings at exactly the right time. This reversal of polarity or, more generally, advancement in magnetic alignment to maximise torque output, is called commutation. The brushes are essentially sliding contacts, so even though brushed DC motors are by far the simplest motors to control, there are some downsides to using them.

3.1.3 Brushless DC motors

Brushless Direct Current (BLDC) motors, in contrast to brushed DC motors, have no physical commutator; they are also known as ”Electronically Commutated Motors”. Not using brushes is significantly more energy-efficient, alleviates sparks, wear and electrical noise and reduces the risks of overheating. BLDC motors are typically equipped with rotary hall sensors which trigger at specific angles and feed their pulses back directly to a control unit. This allows for commutation with barely any digital control logic, operated purely through analogue feedback. Figure 3.5 shows a schematic representation of a three-phase BLDC motor's commutation. Its phases are indicated by coils L_A, L_B and L_C, and, being three-phase, it typically has six commutation steps per rotation. These steps are denoted by the letters of the phases and the Os exactly in between, on the edge of the circle.

As most BLDC motors are commutated with hall sensors, they do not typically use rotary encoders. Doing this, however, it is possible to make them even more efficient. Removing the hall sensors nullifies the capability of automatic commutation without software interference, but offers greater flexibility and programmability. Since the system already has a microcontroller to drive the motor and deal with inputs from the customer, it is a small step to extend its functionality to handle the commutation as well. Using a rotary encoder, a motor rotation can be divided into a number of pulses, counted by the microcontroller. Figure 3.5 shows the positional sectors of the rotor, represented by roman numerals. Depending on which sector the rotor is in, the system constantly compares the angular value of the two neighbouring commutation points with the current encoder value. Then, when this comparison triggers, both the current sector and its corresponding neighbour points are updated accordingly. Immediately after every such trigger, there is an optimal coil charge configuration that, if applied continuously, automatically brings the rotor to the next commutation step, thus keeping the movement of the rotor in sync with the electrical movement of the magnetic field.

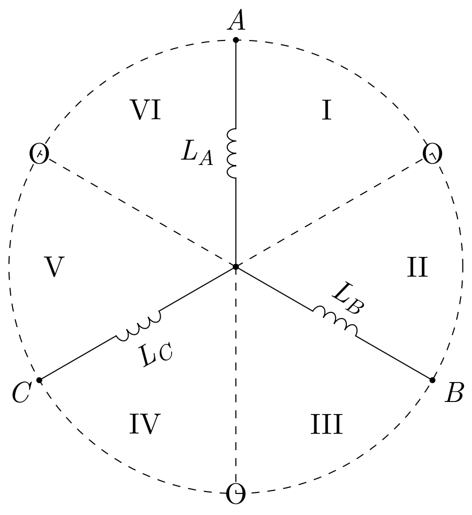


Figure 3.5: BLDC commutation diagram

In construction, a BLDC is actually quite similar to a stepper motor, but the key difference is that it does not divide its rotor into a large amount of discrete steps; it does not have teeth on its rotor nor stator. This trades precision for torque and speed; a BLDC will reach much higher speeds without the risk of losing synchronisation with its steps. However, using a rotary encoder instead of hall sensors, more commutation points could be introduced to Figure 3.5 to obtain a similar effect by powering the coils with a sinusoidal current instead of a block wave.

3.1.4 Comparison

Having described the advantages and disadvantages of the two main types of stepper motors and their respective differences when compared to brushed and brushless DC motors, a rough comparison was summarised in a Kiviat diagram, shown in Figure 3.6. Of interest is the trade-off in a stepper motor versus a BLDC motor; the former performs well at low RPM and, due to its discrete steps, excels in open-loop accuracy. A BLDC, on the other hand, is not limited by step control logic and its performance instead peaks at high RPM.

To quantify the differences between the different types of motors, a list was compiled with potential candidates of comparable size. For reference, the Yaskawa SGMJV-01A [6] servo that was used in the original machine was included. The BM-28L [7] stepper was chosen because of the company's experience with the manufacturer. Also included is the similar sized Trinamic QSH2818 [8] stepper. Finally, the Maxon DCX26L [9] and Vishan EC2864 [10] motors were included and all are shown together in Table 3.1, with their corresponding performance numbers.

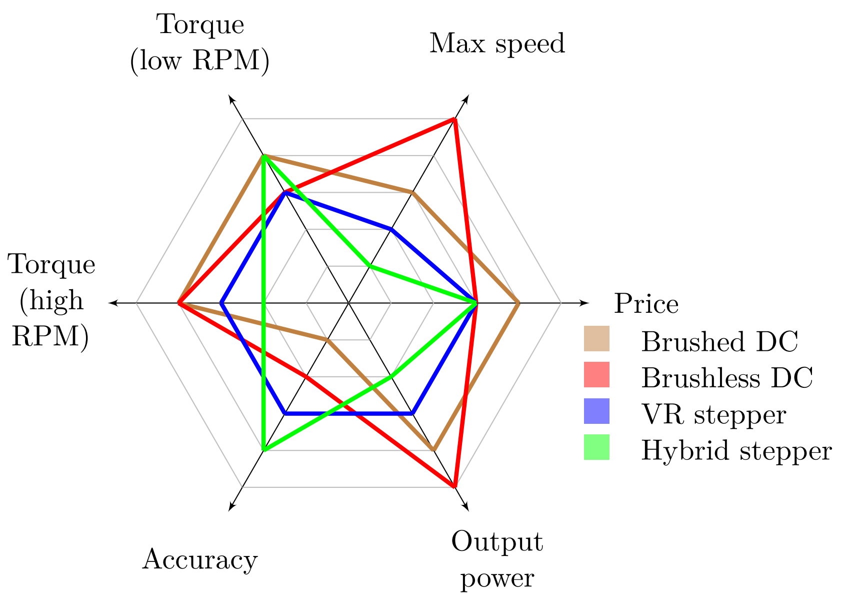


Figure 3.6: Comparison of characteristics of DC motors, VR steppers and hybrid steppers

Manufacturer	Yaskawa	Fastech	Trinamic	Maxon	Vishan
Model	SGMJV-01A [6]	BM-28L [7]	QSH2818 [8]	DCX26L [9]	EC2864 [10]
Price (euros)	800	150	48	200	30
Type	BLDC/Servo	Stepper	Stepper	DC	BLDC
Width (mm)	40	28	28	26	28
Maximum speed (RPM)	6000	2000	2000	13000	18000
Mechanical output (W)	113	11	5	46	56

Table 3.1: Numerical comparison of different motors

3.2 Communication

The coordination of communication was a central part of the design. It involves both calibrating and dynamically recalibrating the system as well as managing location communication between the customer's PLC and the individual motors and correcting and/or reporting errors along the way.

3.2.1 Calibration

Calibration is a crucial part of the system as, immediately after start-up, the controller knows nothing about the slaves it will have to work with. Calibrating the system consists of compiling a list of slaves attached to the motor bus, mapping all of their locations on the rail and reporting/correcting any problem that arises in the process. The process of identification and localisation can be divided into two, as described below, or simultaneously, as discussed in Section 3.2.1.3.

3.2.1.1 Slave identification

There are a number of methods to identify slaves on a bus connection. They can be summarised as follows:

- Brute-force search the address space
- Perform an optimisation of a brute-force search that makes use of a priori knowledge
- Use collision detection/management

The simplest way is to brute-force the entire physical address space of the slaves. Using the Big O-notation as defined for use in computer science in [11], this has a time complexity of $\Theta(2^n)$, where n is the number of address bits. Since the n of the microcontrollers used is 128 bits [12], this would take many years to complete. A potential optimisation of the brute-force approach is to use a binary tree search. The address space is recursively split in two to see whether any slave's address falls within either half's bounds, and if not, that part is discarded. This leads to a time complexity bounded between $\Omega(2n)$ and $O(2^{n+1})$; the time required depends greatly on the number of slaves present, decreasing drastically for a low number.

A third approach is to perform collision detection during the identification phase. Collision detection works by looping through the following steps:

- Broadcast from master: all slaves who have not been identified, reply with your address
- All eligible slaves wait for different random intervals while the channel is quiet
- If no slaves are eligible, break loop
- Slaves transmit their address as a reply to the master
- Since the channel is now busy, other slaves remain quiet and wait for the next broadcast

Naturally, chance allows for two slaves to use the same time interval. This is where a collision occurs. Unfortunately, RS485 does not have innate collision detection, but it *is* half-duplex [13]. This means that, while a slave is transmitting, it can see its own message on the bus, allowing it to check whether this matches the intended message. If this is not the case, it knows that a different slave is sending simultaneously, and the collision is detected. All conflicting slaves will again wait for a random interval to resubmit their reply, and the same mechanism is applied until a single slave has managed to get its message through and thus managed to obtain an address. Then, the loop repeats until all motors have been assigned.

However, having two controllers transmit two different values on a single bus causes a short between them, which might pose a threat to the communication chips involved. To protect them, collision detection might also be done by having motors pull the bus down within a certain Gaussian time range, and then waiting until this time slot is over to start transmitting. Within the time slot, other motors will find the bus already pulled down, signalling that the current time slot is not theirs. Then, when it is over, the one that pulled it down initially would be free to reply to the master in order to obtain an address. It will then stop participating in the process, which repeats until all motors have been assigned.

3.2.1.2 Position detection

As mentioned in Section 2.2, two types of sensors were introduced; optical and magnetic proximity sensors. Using the information from these sensors, certain conclusions can be drawn. Since at this point all motors have been assigned an Id, the controller can communicate with them specifically instead of broadcasting. Communicating with all motors one by one, all of them can then be moved to one side, as a result of which only a single motor will not have a neighbour towards the other side. After reassigning that motor's address to the first in order, it is moved to the other side. This process is repeated until all motors have traversed the full track, learning the length of this track in the process. At the end of this process there will be a mutual understanding between the main and motor controllers regarding Id numbers and ranges.

3.2.1.3 Combined method

Using the information from the proximity sensors, a different method was devised, which allows the master controller to compile a list of motors and positions at the same time. As an added advantage, each motor discovers all of its possible positions in the process and each is numbered in order, allowing for convenient debugging information. This method is summed up in a DRAKON [14] diagram in Figure 3.7.

3.2.1.4 Dynamic recalibration

After calibration is completed, all motors can save their Id to Electrically Erasable Programmable Read-Only Memory (EEPROM) while the controller saves every motor's physical address with corresponding Ids to flash or EEPROM. This allows for a quicker calibration run the next time the system is powered up; the motors can then be identified, commanded and tested individually, and a full recalibration would only be required if a problem comes up.

During operation, many factors can play a role in creating small offsets on the motor positions. Most of these will be cancelled out by using closed-loop control. However, as the closed-loop control is completely dependent on the rotational encoder signal, some deviation might persist if it is not perfectly accurate. Fortunately, the magnets used to determine motor positions during regular calibration will remain usable during runtime, allowing the motors to correct their positions anytime they happen to pass by a magnet.

3.2.2 Full operational bus communication

An initial idea was to control the whole system by communicating over the bus, with the master requesting every slave's location as often as communication restraints allow, and deciding purely on basis of those locations when stopping is necessary. This would

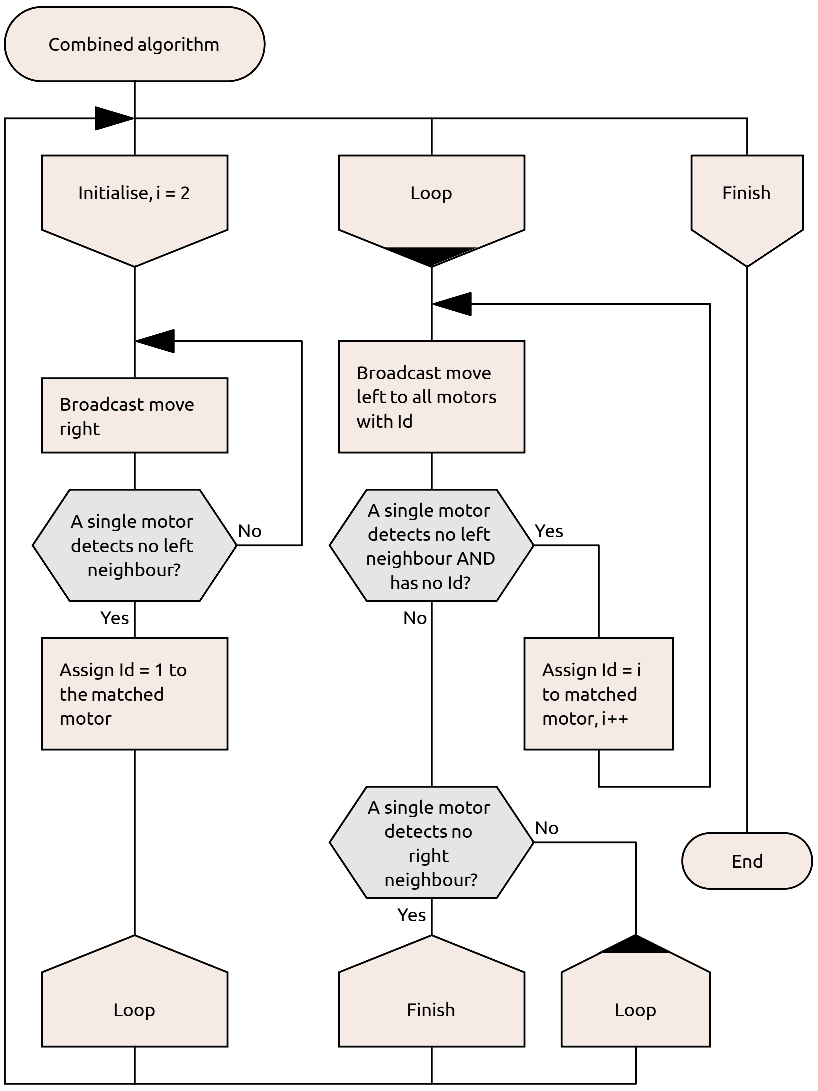

Combined algorithm
Initialise, i = 2
Loop
Finish
Broadcast move right
A single motor detects no left neighbour?
No
Yes
Assign Id = 1 to the matched motor
Loop
Broadcast move left to all motors with Id
A single motor detects no left neighbour AND has no Id?
Yes
No
Assign Id = i to matched motor, i++
A single motor detects no right neighbour?
No
Yes
Finish
Loop
End

Figure 3.7: DRAKON diagram of the combined calibration method

have been highly advantageous, as it would even have allowed the main controller to know the whereabouts and situations of all active motors, making it possible to adjust their trajectories based on positions of other motors. To find the maximum displacement resolution of the motors by communicating in this manner, a number of variables were defined:

- B, the maximum bitrate on the bus
- M, the size of a single packet in bits, where $M = N_{data} + N_{envelope}$;
  - N_{data}, the number of bits required for the data part of communication

  - $N_{envelope}$, the number of bits required for the rest of the packet; overhead

- N_c, the number of packets sent in a full iteration (some function of the number of motors on the rail and a packet loss factor)
- v_{max}, the maximum speed of a motor at any given time
- a_{max}, the maximum acceleration of a motor at any given time

The time required to make an emergency stop after the master detects an error was expressed as $\Delta t_{wc} = \Delta t_{comm} + \Delta t_{dec}$ with Δt_{comm} the maximum time to communicate once with all motors and Δt_{dec} the time required to decelerate a motor to a halt from full speed, according to Equations 3.1 and 3.2. Subsequently, the average speed during this time was expressed by Equation 3.3 and $\Delta s_{wc} = \Delta t_{wc} \cdot v_{avg}$ was defined as the worst case (maximum) distance travelled by a motor during the time between the diagnosis of an error and a full stop, yielding Equation 3.4 by combining Equations 3.1 through 3.3.

$$\Delta t_{comm} = \frac{(N_{data} + N_{envelope}) \cdot N_c}{B} \tag{3.1}$$

$$\Delta t_{dec} = \frac{v_{max}}{a_{max}} \tag{3.2}$$

$$v_{avg} = \frac{v_{max} \cdot \Delta t_{comm} + \frac{v_{max}}{2} \cdot \Delta t_{dec}}{\Delta t_{comm} + \Delta t_{dec}} \tag{3.3}$$

$$\Delta s_{wc} = v_{max} \cdot \frac{(N_{data} + N_{envelope}) \cdot N_c}{B} + \frac{v_{max}^2}{2 \cdot a_{max}} \tag{3.4}$$

Using preliminary values ($B = 115200$ Hz, v_{max} and a_{max} values from Section 4.1.3, N_{data} and $N_{envelope}$ minimised), an approximate resolution of $15mm$ was found. This was, however, assuming no brakes are present on the motors, which might impact Δt_{dec}. Worth noting is that, if the decision to make an emergency stop can be initiated by a single motor pulling down the bus, this would effectively eliminate Δt_{comm}.

3.2.3 Limited bus communication

The approach in the previous section was based on the assumption that a reasonable amount of communication is possible on a bus riddled with sliding contacts. This might, however, prove difficult, as during operation, these contacts would likely break and re-establish contact with the bus regularly, causing highly unpredictable noise on the channel. While the impact of this problem remained uncertain until testing was done, the bulk of the necessary communication was preemptively moved to the phases where the motors are standing still. Since this is a time-critical process (the motors must be moving as often as possible), it is beneficial to minimise the time spent communicating.

Equation 3.1 holds to describe the time required to communicate some data to every motor on the bus. Enforcing a resolution of 0.1 mm on a system with $L_{rail} = 1.5$m requires a data signal N_{data} of 14 bits. As the system works with bytes, this was extended to 16 bits. With 32 motors and a packet loss factor of 20%, $\Delta t_{comm} \approx 55ms$. After receiving their location data, the motors go into operation mode, as described in Section 1.2. While waiting for its start signal and while crossing its required trajectory directly afterwards, each motor can compute the next iteration's trajectory using the new location data obtained from the master controller.

Using this method, every motor can compute its required location in real time and compare it against its actual location, making corrections where necessary. Combining this with an emergency stop initiated by pulling down the bus, each motor is capable of individually doing a significant part of the error checking mentioned in Section 2.4.

Alternatively, the master controller can compute all the required trajectories and transmit the resulting coefficients to the motors. The advantage of this is that it would allow the trajectories to take other motor's locations into account. However, this is much slower for two reasons. Firstly, all trajectory computation would take place on the master controller instead of being computed in parallel amongst the motors. And secondly, there would be significantly more data to transmit to the motors - a minimum of four floating point numbers. While these could all theoretically fit into the same packet, it would still at the very least increase the aforementioned value for Δt_{comm} by 50%.

3.2.4 Wireless communication

While the system is marketed as wireless, it is, in a sense, not strictly wireless, as the system's communication signals still move along a physically connected conductor; the communication bus. Instead, it might be possible to only transmit power over the track rails and leave communication to an actual wireless system. This might prove useful if one system has to control multiple rails that are not physically interconnected. However, the presence of all manner of electronics is likely to be detrimental to any forms of wireless communication, through their emission of electromagnetic interference. This is especially true for the large electric motors required to move the rail. These circumstances make wireless communication a questionable contender, but significant progress is being made in cancellation of electromagnetic interference in wireless systems. Furthermore, if it does turn out to yield an acceptable packet loss rate (which is to be determined through measurement), it would be easy to implement, as it can essentially be considered to operate in the same way as the proposed bus communication system. Like a metal bus with interconnect, the air through which it operates can also be considered as a single channel. However, progress in this direction is made difficult by the fact that a competitor owns a patent specifically on wireless communication in transplanting machines [15].

3.3 Trajectory generation

To drive the motors on the rail, simple target positions could be fed through feedback controllers to allow them to smoothly approach these targets. Such controllers could examine the difference between the target position and the motor's actual position and correct the motor input accordingly. However, since many other motors are present on the rail, not knowing in advance how they will move from their initial positions to their target positions might cause motors to move too close to each other, leading to unnecessary stalling. Furthermore, since the models proposed were to be used in a real world situation, it made sense to guarantee the continuity of physical parameters. Controllers would inevitably be part of the system, but instead of steering them using a step function, it would likely prove beneficial to exert more control over the motor trajectory by customising the controller input, allowing every motor to follow a similarly shaped trajectory and thus keeping distance between them relatively linear. This section describes several different methods of finding trajectories that comply with physical continuities, as discussed in [16]. Note that all trajectories are described in a single dimension; only the horizontal movement, along the rail, is controlled by the motors, as knowing about vertical movement is beyond their scope.

3.3.1 Parabola shape

Parabolas are second-order polynomials, with a round shape moving away symmetrically from their extreme. In this section, we consider higher-order polynomials based on the general, symmetric shape of a parabola. A significant advantage of polynomials is that their derivatives are very easy to compute. Defined is a polynomial of the order N to describe the trajectory of a motor: $q(t) = a_0 + a_1 t + ... + a_N t^N$. Its velocity is then the derivative $\dot{q}(t) = a_1 + 2a_2 t + ... + N a_N t^{N-1}$, and its acceleration the double derivative $\ddot{q}(t) = 2a_2 + 6a_3 t + ... + N \cdot (N-1) a_N t^{N-2}$. Using order N, $N+1$ boundary conditions are required to find valid equations to describe the trajectory. At order 3, the initial and final positions and velocities (q_0, q_f, $\dot{q}_0$, $\dot{q}_f$) can be supplied to find valid equations, as shown in Equation 3.5. Equation 3.6 shows the matrix form of this set of equations. This can be extended to also include boundary conditions on the acceleration ($\ddot{q}_0$, $\ddot{q}_f$), which is still allowed to be non-zero at the beginning and the end of the trajectory in Equation 3.6. This requires a fifth order system, as demonstrated in Equation 3.7. An example of a fifth order trajectory and its derivatives is shown in Figure 3.8 with $q_0 = 1$, $\dot{q}_0 = 0$, $\ddot{q}_0 = 0$, $q_f = 4$, $\dot{q}_f = 0$, $\ddot{q}_f = 0$, $t_0 = 0$, $t_f = 5$.

$$\begin{cases} q(t_0) = a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 \\ \dot{q}(t_0) = a_1 + 2a_2 t_0 + 3a_3 t_0^2 \\ q(t_f) = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 \\ \dot{q}(t_f) = a_1 + 2a_2 t_f + 3a_3 t_f^2 \end{cases} \tag{3.5}$$

$$\begin{bmatrix} q_0 \\ \dot{q}_0 \\ q_f \\ \dot{q}_f \end{bmatrix} = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{3.6}$$

$$\begin{bmatrix} q_0 \\ \dot{q}_0 \\ \ddot{q}_0 \\ q_f \\ \dot{q}_f \\ \ddot{q}_f \end{bmatrix} = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} \tag{3.7}$$

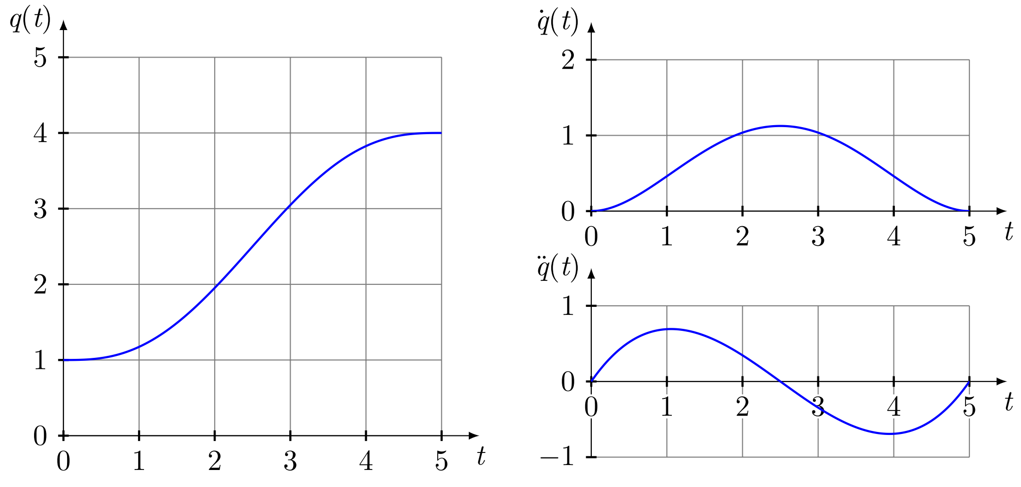


Figure 3.8: Clockwise from left: examples of position, velocity and acceleration in a 5th order parabola trajectory

3.3.2 Trapezoid shape

A trapezoid is defined as a symmetric shape that consists of an upslope and its corresponding downslope, connected at the top by a straight line (see upper right graph of Figure 3.9 for reference). Choosing a trajectory this way allows for some assumptions that make the computation relatively lightweight, at the cost of less continuity. Calculating the equation for the upslope, a polynomial is defined much like in the previous section, but with order 2; $q(t) = a_0 + a_1 t + a_2 t^2$. Likewise, the downslope is defined as $q(t) = c_0 + c_1 t + c_2 t^2$, and finally, the straight section is defined as $q(t) = b_0 + b_1 t$; a first order polynomial. Since these equations are lower order, they can be conveniently expressed by their boundary conditions q_0, $\dot{q}_c$ (cruise velocity), t_s (slope time), t_f, as shown in Equation 3.8 [16]. Taking the first and second derivatives of Equation 3.8 shows how the equations are defined by their boundary conditions, demonstrated in Equations 3.9 and 3.10. This also illustrates the conditions that t_s and $\dot{q}_c$ must be chosen so that $t_f \geq 2t_s$ and $a_{max} \geq \dfrac{\dot{q}_c}{t_s}$, correlating to the motor torque through Equation 2.2. Additionally, one of the boundary conditions must be chosen to match $q_f = q_0 + \dot{q}_c(t_f - t_s)$. An example of a trapezoid trajectory is shown in Figure 3.9 with $q_0 = 1$, $q_f = 4$, $t_f = 5$, $\dot{q}_c = 0.8$, $t_s = 1.25$.

$$q(t) = \begin{cases} q_0 + \dfrac{\dot{q}_c}{2t_s} t^2 & 0 \leq t < t_s \\ q_0 + \dot{q}_c(t - \dfrac{t_s}{2}) & t_s \leq t < t_f - t_s \\ q_f - \dfrac{\dot{q}_c \cdot (t_f - t)^2}{2t_s} & t_f - t_s \leq t \leq t_f \end{cases} \tag{3.8}$$

$$\dot{q}(t) = \begin{cases} \dfrac{\dot{q}_c}{t_s}t & 0 \le t < t_s \\ \dot{q}_c & t_s \le t < t_f - t_s \\ \dfrac{\dot{q}_c t_f}{t_s} - \dfrac{\dot{q}_c t}{t_s} & t_f - t_s \le t \le t_f \end{cases} \tag{3.9}$$

$$\ddot{q}(t) = \begin{cases} \dfrac{\dot{q}_c}{t_s} & 0 \le t < t_s \\ 0 & t_s \le t < t_f - t_s \\ \dfrac{-\dot{q}_c}{t_s} & t_f - t_s \le t \le t_f \end{cases} \tag{3.10}$$

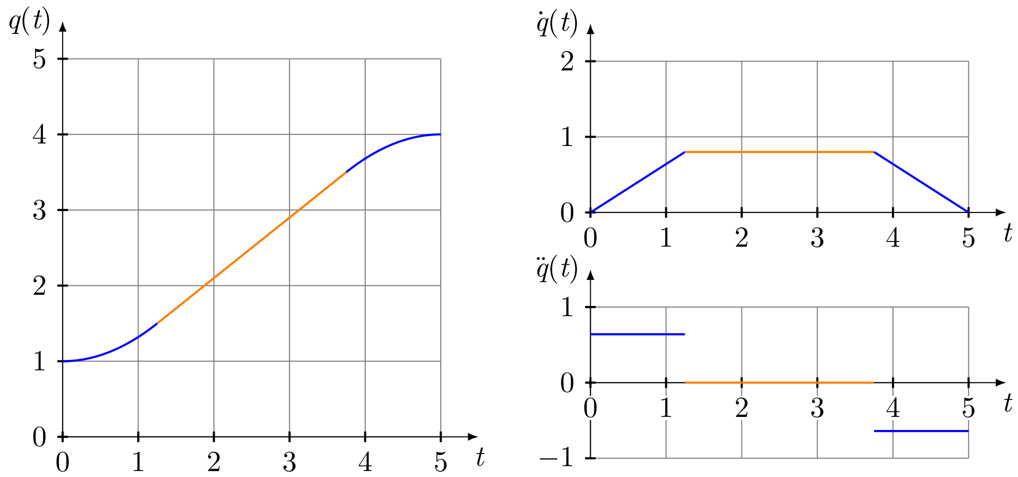


Figure 3.9: Clockwise from left: examples of position, velocity and acceleration in a trapezoid trajectory, with slopes coloured blue and cruise velocity coloured orange

3.3.3 Splines

It might be necessary for the trajectory to pass specific points along the route. This could be achieved by simply adding $2n$ (where n is the number of points to pass) to the order of a parabolic trajectory to force additional boundary conditions on it. However, as the order increases, computation increases exponentially, which quickly becomes a problem as more points are introduced. When compared to simply increasing the order of the trajectory polynomial, splines offer an alternative without losing much of the precision of the trajectory. Like the trapezoid shape, they are made by concatenating multiple lower-order polynomials. For example, a combination of systems like that in Equation 3.5 can be used to create n third order systems, guaranteeing continuous position, velocity and acceleration as well as allowing for boundary conditions for the initial and final velocities and positions (see Equation 3.11).

$$\begin{cases} q(t) = \{q_k(t), & t \in [t_k, t_{k+1}], k = 1, ..., n-1\} \\ q_k(\tau) = a_{k0} + a_{k1}\tau + a_{k2}\tau^2 + a_{k3}\tau^3, & \tau \in [0, T_k], (\tau = t - t_k, T_k = t_{k+1} - t_k) \end{cases} \tag{3.11}$$

As in Equation 3.6, initial and final conditions can be forced on the system. Using the conditions $q_k(0) = q_{k-1}(T_k)$, $\dot{q}_k(0) = \dot{q}_{k-1}(T_k)$, $q_k(T_k) = q_{k+1}(0)$ and $\dot{q}_k(T_k) = \dot{q}_{k+1}(0)$, Equation 3.12 was obtained to describe the constants a_{kN}.

$$\begin{cases} a_{k0} = q_k \\ a_{k1} = \dot{q}_k \\ a_{k2} = \frac{1}{T_k}[\frac{3(q_{k+1} - q_k)}{T_k} - 2\dot{q}_k - \dot{q}_{k+1}] \\ a_{k3} = \frac{1}{T_k^2}[\frac{2(q_k - q_{k+1})}{T_k} + \dot{q}_k + \dot{q}_{k+1}] \end{cases} \tag{3.12}$$

Since the intermediate velocities $v_2, ..., v_{k-1}$ were not yet known, the continuity condition of the acceleration in intermediate points was used: $\ddot{q}_k(T_k) = \ddot{q}_{k+1}(0) = 2a_{k2} + 6a_{k3} = 2a_{k+1,2}$. Filling Equation 3.12 back into this equation yields Equation 3.13. Writing these out for every value of k, a matrix is found, as shown in Equation 3.14. In this matrix, c is the right hand side of Equation 3.13, which consists only of known terms. This leaves a set of $k-2$ equations with $k-2$ unknowns (the initial and final velocity are known), which, like a k-order polynomial, has a computational complexity of $O(k^3)$. However, since the resulting matrix is tridiagonal, inverting it can be done in $O(k)$ using the Thomas Algorithm [17]. Multiplying both sides of Equation 3.14 by this inverted matrix yields a set of solutions for $\dot{q}_1, ..., \dot{q}_n$. Filling these back into 3.12 returns valid polynomials for every part of the spline. An example of a spline trajectory is shown in Figure 3.10 with boundary conditions $\dot{q}_0 = 0$, $\dot{q}_f = 0$, the coordinates of intermediate points denoted by dots in the graph and continuity of velocity and acceleration in these points. If the computation is fast enough, this approach is also suitable for mid-flight trajectory changes. This must, however, fall within certain boundaries, as no individual motor knows anything about the whereabouts of any of its peers.

Table 3.2 shows the advantages and disadvantages of implementing the different types of trajectories, and Chapter 4 specifies the implementation of the third order parabola trajectory, as it was expected to give sufficient continuity at the lowest computational cost.

$$\dot{q}_k T_{k+1} + \dot{q}_{k+1} 2(T_{k+1} + T_k) + \dot{q}_{k+2} T_k = \frac{3}{T_k T_{k+1}}[T_k^2(q_{k+2} - q_{k+1}) + T_{k+1}^2(q_{k+1} - q_k)] \tag{3.13}$$

$$\begin{bmatrix} T_2 & 2(T_1+T_2) & T_1 & 0 & 0 & 0 & 0 \\ 0 & T_3 & 2(T_2+T_3) & T_2 & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & T_{n-2} & 2(T_{n-3}+T_{n-2}) & T_{n-3} & 0 \\ 0 & 0 & 0 & 0 & T_{n-1} & 2(T_{n-2}+T_{n-1}) & T_{n-2} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_{n-1} \\ \dot{q}_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-3} \\ c_{n-2} \end{bmatrix} \tag{3.14}$$

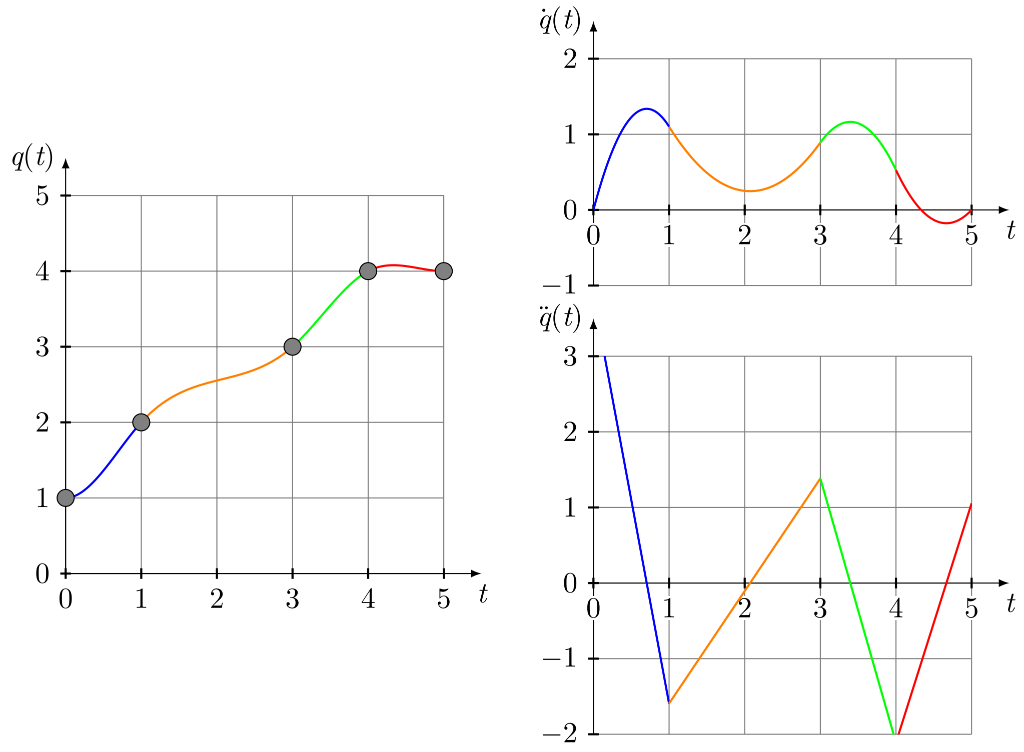


Figure 3.10: Clockwise from left: examples of position, velocity and acceleration in a third order spline trajectory through predetermined points (denoted by dots)

	Continuity	Computational weight	Flexibility
Parabola (3)	+	+	+/-
Parabola (5)	++	-	+
Trapezoid	+/-	++	+/-
Spline (3)	++	- -	++

Table 3.2: Trajectory type candidates, with order in brackets where applicable

3.4 Motion control

Every motor must follow its target trajectory, as defined in Section 3.3, as accurately as possible. The most basic form of operation is through open-loop control. This assumes that, when driven, the motor position will not deviate from its input signal. Stepper motors thrive on this, as they can make a specific number of steps very accurately. However, this assumes ideal circumstances, and there will always be a certain discrepancy between the actual trajectory and the target trajectory, which must be compensated through some form of feedback. A servo consists of a motor with built-in feedback, as its drive corrects it by comparing a rotary sensor value with its control input. Non-servo motors, however, do not have a native control system, so the control must be manually implemented by the designer either through hardware or software. The discrepancy $\epsilon(t)$

between the actual trajectory $\hat{q}(t)$ and the target trajectory $q(t)$ is defined as $\epsilon(t) = \hat{q}(t) - q(t)$. To use closed-loop control means to use this ϵ as a feedback parameter to correct the deviation that the system is producing. A closed-loop controller achieves this by trying to minimise the value of ϵ as time goes by. This is shown in Figure 3.11, where the system shown can be modelled with a certain transfer function; a function that, given a certain input, produces a certain output as a result. The goal of any controller as shown in the figure is to alter this system's input signal in such a way that its transfer function more agreeably (i.e. faster or more accurately) produces the required output.

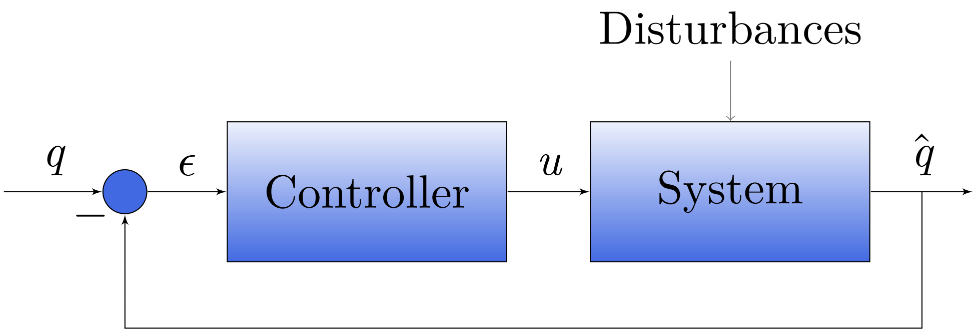


Figure 3.11: Block diagram of a system with closed-loop control

3.4.1 PID control

The simplest way to minimise the error is by using a Proportional, Integral, Derivative (PID) controller. As the name implies, a PID controller influences the control output by multiplying the error between the target and actual trajectories and their integral and derivative by three different filter parameters, and adding the results together. This is illustrated in Figure 3.12. Tuning these parameters, K_p, K_i and K_d, greatly influences the way the system responds to certain inputs. The resulting output u is given in Equation 3.15.

$$u(t) = K_p\epsilon(t) + K_i \int \epsilon(t)\,dt + K_d\frac{d\epsilon(t)}{dt} \tag{3.15}$$

Applying a step function, an instant change in target, to a system's input is the most convenient way to influence it. Giving this step function the amplitude of the target position effectively commands it to move to that position. Depending on the parameters of the PID controller (given that the controller is stable), this command is then automatically executed with a certain strictness. Its rise time (time to reach the position) and overshoot (factor by which it moves beyond its target) also greatly depend on the values of the controller parameters. Implementing PID implicitly defines a trajectory, effectively nullifying the need for predefined trajectories. However, given the unpredictable disturbances in the system, tuning the parameters is a difficult process that will yield different results for different motors. Both the track friction and the dynamic behaviour of the different motors are never perfectly identical.

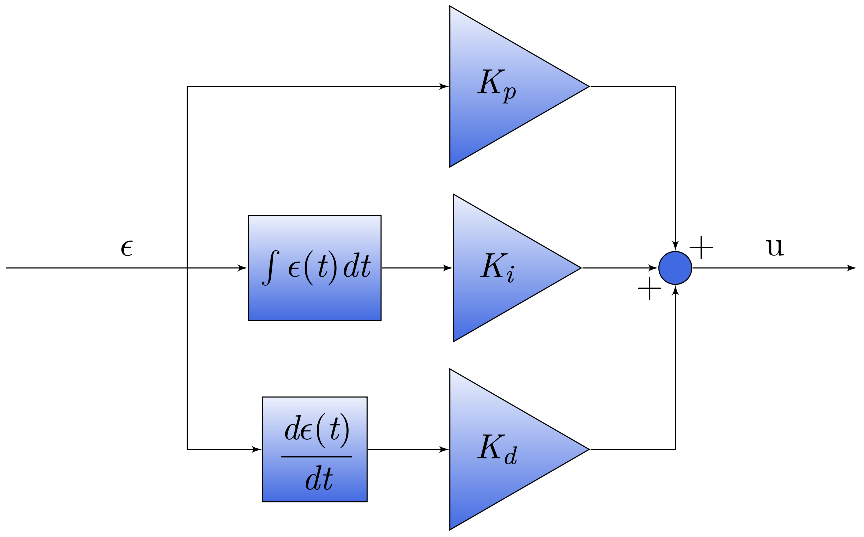


Figure 3.12: Block diagram of a PID controller

3.4.2 Setpoint selection

Every type of motion controller needs a setpoint; a target value for the system's output to attain. This can be static, in which case the system is let loose by making it target its goal immediately. This way, a PID's parameters implicitly define a trajectory to follow by forcing the system towards that goal. This behaviour is unpredictable, however, as the response greatly depends on system parameters that cannot be accurately predicted. Alternatively, a dynamic setpoint can be used that takes the actual setpoint as its input. With this approach, the trajectories defined in Section 3.3 can be used to generate paths with a specific arc and duration befitting the distance to be travelled. This way, only the difference between the current location and the dynamic target location is needed to know how fast the motor must be moving. This better fits the needs of the customer, as it allows all motors to take exactly the same amount of time to get to their destination and makes it impossible for different trajectories to cross.

System Implementation 4

This chapter introduces the different microcontrollers used and how their software was written to comply with all of the design considerations. Section 4.1 details the implementational facets of the hardware used, Section 4.2 describes the function and implementation of the main controller, and Section 4.3 expands on the specifics of motor control and its software implications.

4.1 Hardware

Over the course of the project, many different hardware considerations had to be made. This section gives an in-depth analysis of motor types, how to drive them and how to decide what gear size and type to use.

4.1.1 Motor drive

Regardless of the type of motor chosen, all can effectively be modelled as a bundle of windings with inductive and resistive properties, in addition to acting as a generator when a change in current is applied. This is demonstrated in Figure 4.1, with characteristic Equation 4.1. In this equation, u_{phase} is the applied voltage, i_a the induced voltage, R_a the resistance of the windings, L_a the inductance and e_a the back-Electromotive Force (EMF). From an electrical point of view, a brushed Direct Current (DC) motor is as simple as that; the brushes automatically guarantee commutation and thus reversal of the direction of current. For other types of motors, on the other hand, it is more complicated. The a in subscript implies that this circuit potentially represents a phase. For any motor with multiple phases, their sets of windings must be energised in order, as was described in Section 3.1.1.

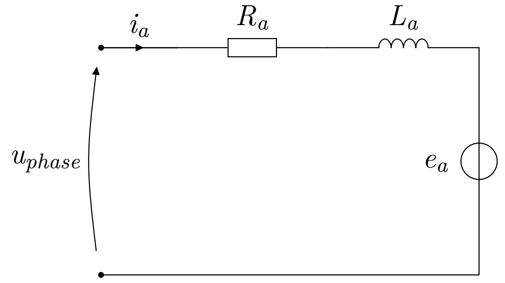


Figure 4.1: Equivalent circuit of an electric motor phase winding

$$\frac{di_a}{dt} = \frac{1}{L_a}(u_{phase} - i_a R_a - e_a) \tag{4.1}$$

To energise motor windings, H-bridges are typically used. In addition to making it easy to reverse the polarity of a winding, these bridges also help to relieve the circuitry of back-EMF when the supplied current is significantly reduced, e.g. when powering down. A typical setup with Metal Oxide Semiconductor Field Effect Transistor (MOSFET)s is shown in Figure 4.2. In the figure, a motor is displayed in the centre, as would be the case when operating a brushed DC motor. In any other case, this would be a single winding and a number of bridges equal to the number of phases would be required to use the motor. In this example, transistors A and D can be enabled to allow a current to flow through the motor from left to right, while enabling transistors B and C allows a current to flow from right to left, reversing the polarity and thus the turning direction of the motor. The diodes serve to provide a low impedance path for back-EMF induced by the motor when the input voltage is suddenly removed. This protects the MOSFET gates from being forced to conduct a large current, which might otherwise blow them up. Such implementations are easily incorporated into a Printed Circuit Board (PCB) design by using specialised off-the-shelf chips that contain the MOSFET gates as well as the capacitors that protect them.

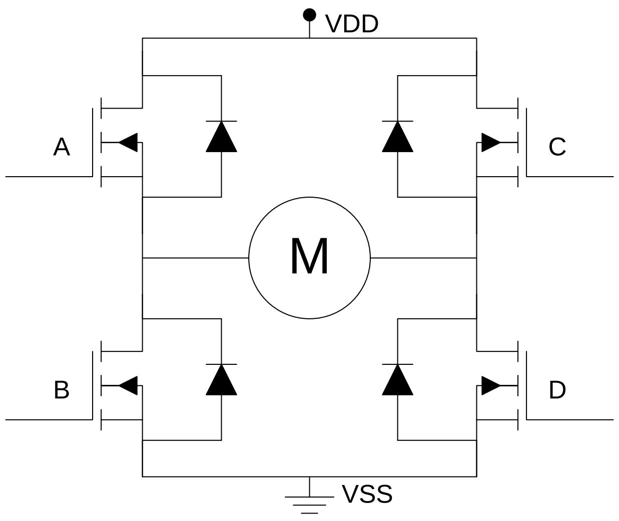


Figure 4.2: MOSFET full-bridge connected to motor phase

4.1.2 Gearing

Introducing gears into a system typically dissipates roughly 10-15% of the energy throughput for every gearing stage. Despite these losses, gearing can be helpful or even necessary for a number of reasons:

- Matching the torque and/or speed requirements of the application at the output shaft
- Moving the motor requirements into a domain that can (efficiently) be handled by a certain motor

- Transforming mechanical energy from one domain into another (rotational versus translational)

Figure 4.3 shows a mechanical representation of the gears present in the system. The leftmost gear combination is called a planetary gear train, due to the layout in which the teal-coloured gears (planets) ”orbit” around the central gear like in a solar system. This configuration can efficiently reduce the angular speed without influencing shaft alignment. The axes of the teal gears are connected to the rear plate through ball bearings, allowing the axes to rotate within the plate even as they push it around. The red-coloured outer gear is fixed, with the other gears moving around inside of it. Increasing the ratio N_1 of the outer gear radii with respect to that of the central one increases the reduction factor.

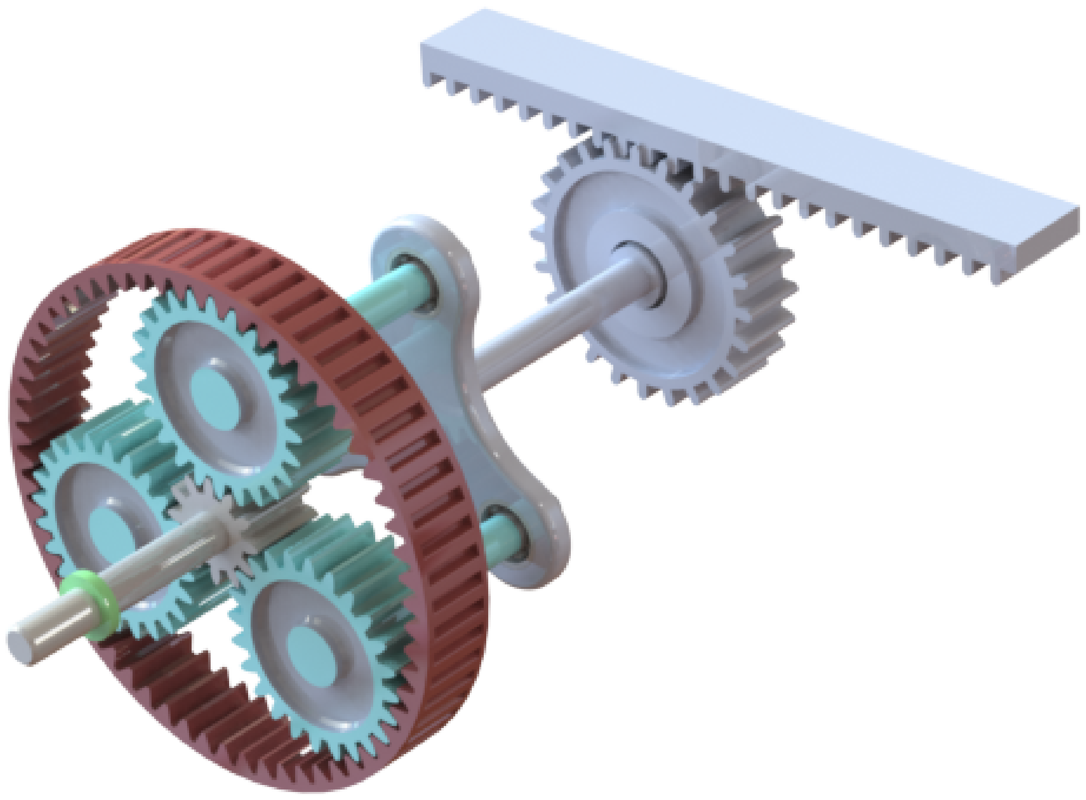


Figure 4.3: Mechanical schematic of the gear systems on the motor shaft

The rightmost gear combination is called a rack and pinion system, which allows rotational movement to be transformed into translational movement. The radius of the gear in this system as well as both tooth pitches (distance between two teeth on a toothed surface) determine the ratio N_2 of movement transformation.

In both gear systems, the gear ratio also influences the force produced on the other end. A reduction of rotational speed in the first phase leads to an equal increase in torque. In the second phase, an increase in gear ratio is signified by a larger gear, which leads to an increased translational velocity. In this case, the torque remains the same, but the output force is proportionally smaller because of the greater distance from the shaft.

In the system designed, these effects were combined in a trade-off because choosing $N1$ allowed the motor to perform at an efficient output power setpoint while choosing $N2$ dictated the resulting translational speed regardless of the intermediate conditions. Leaving out the first gear stage would have been preferable to save the energy it dissipates, but working at an inefficient power setpoint, the motor would have performed poorly, unable to meet either the torque or the speed requirements.

4.1.3 Motor type

Building on Table 3.1, additional calculations were done with the motors to determine their performance with regard to the system. The old transplanter motors had a maximum velocity of ~ 2.2 m/s and a maximum acceleration of ~ 8 m/s^2. The requirements on these were obtained by reducing the required values to 2 m/s and 4 m/s^2, which is a reasonable decrease given that, stepping down from a bigger motor, the output power must be halved at the very least. These values were used in Equations 2.2 and 2.3 to solve for m and obtain numerical results for its maximum value with the different potential motors. Rough optimisations were taken for the gear ratios, rounded to whole numbers, and the maximum load includes the weight of the motor itself. Electrical input power was calculated from the maximum continuous input voltage and current, and mechanical output power from the theoretical output torque and angular speed, with gearhead losses estimated at 10% per stage, assuming all ratios can be achieved through a single stage. Friction is excluded, but so is intermittent overloading, of which all motors are capable to some extent. The results are shown in Table 4.1.

Manufacturer	Yaskawa	Fastech	Trinamic	Maxon	Vishan
Model	SGMJV-01A [6]	BM-28L [7]	QSH2818 [8]	DCX26L [9]	EC2864 [10]
Price (euros)	800	150	48	200	30
Type	BLDC/Servo	Stepper	Stepper	DC	BLDC
Width (mm)	40	28	28	26	28
Input speed (RPM)	5305	1768	1768	8840	14144
Input torque (Nm)	0.180	0.060	0.028	0.050	0.042
Electrical input (W)	168	23	16	66	81
Optimal gear ratio	3	1	1	5	8
Output speed (RPM)	1768	1768	1768	1768	1768
Output torque (Nm)	0.486	0.060	0.028	0.250	0.302
Mechanical output (W)	113	11	5	46	56
Maximum load (kg)	11.1	1.34	0.58	5.09	6.95

Table 4.1: Numerical comparison of different motors given that v_{max} = 2 m/s and a_{max} = 4 m/s^2 must hold; an extension of Table 3.1

When working with stepper motors, micro-stepping can be used to influence the trade-off of accuracy and stability versus torque production and power consumption. In micro-stepping, multiple stator phases in a motor are excited at a specific level, achieving a balance of fields that puts the rotor teeth in an intermediate step between two sets of stator teeth that it simultaneously attempts to align with. Energising multiple phases at the same time undeniably increases power consumption as well, but the torque production is unlikely to rise with the same magnitude. The amount of micro-stepping is denoted by a micro-stepping factor that multiplies the amount of steps in a full rotation, denoted by k in Equation 2.4. Without micro-stepping, the resolution of a typical stepper motor is 0.16 mm. Since [3] requests a resolution of 0.1 mm, half-stepping should suffice as far as accuracy goes. How further micro-stepping might affect both stability and torque production, however, is better tested through simulation and/or prototyping.

Although meeting the constraint on accuracy is very easy using a stepper motor, DC motors are more capable of meeting practically every other requirement posed in Table 2.1. If the system would be controlled through an open loop, steppers would definitely be the sensible choice. Using closed-loop feedback control, however, this lack of accuracy can likely be mitigated. The accuracy of this control requires some testing, but if it turns out to be sufficiently accurate for the application, there will be no need to even consider stepper motors, as accuracy is their one strong point. If it is not, however, stepper motors will be required to match the strict accuracy requirement.

As was shown in Table 2.2, the system must be able to support roughly 1.4 kg, which means that the stepper motors examined are unlikely to be able to provide sufficient mechanical power to reach the required acceleration and velocity. The company was hesitant about trusting Chinese data sheets and did not want to risk waiting for very long shipping times. Therefore, since brushed DC motors are cheaper and easier to control than their brushless counterparts, it was decided to use the Maxon DC motor in the machine.

4.1.4 Microcontrollers

As discussed in Chapter 1, the system designed is part of and effectively managed by a larger system. In the eyes of this larger system, the system described is considered a black box, which receives certain inputs and produces certain outputs as a result. For this design, the top-level system can in turn be considered a black box. Where one box ends and the other starts is an arbitrary line, which was drawn on the edge between both systems' main controllers. For the purposes of this design, it suffices to consider the top-level system as a single Programmable Logic Controller (PLC), which transmits target outputs to the lower system through an ethernet connection. This lower system, in turn, consists of a main controller that communicates with controllers hierarchically beneath it. The main controller is an Atmel ATSAM4SD32B [12], which, through an RS485 bus, is connected to many individual motor controllers. These motor controllers are equipped with relatively small Atmel ATxmega128A1U [18] chips, in addition to the sensors and actuators described in Chapter 2. These chips were selected for being low-cost commercial off-the-shelf products, allowing a little more leniency for the main controller since there is only one in a machine, as opposed to one for every motor.

A minimal architectural representation of this subdivision is shown in Figure 4.4. In

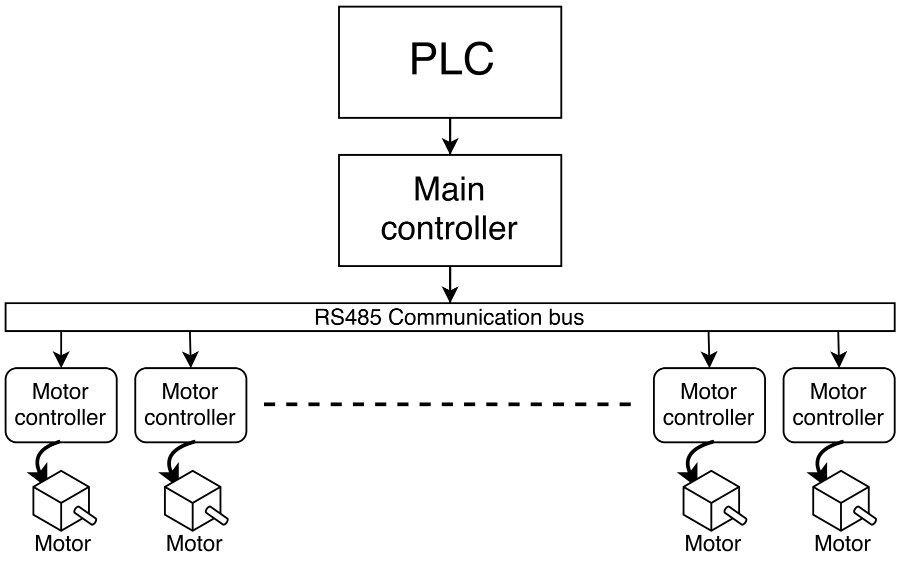


Figure 4.4: Top level diagram of the control hierarchy

this diagram, the arrows represent master-slave relationships, with the arrows pointing towards the slaves. In a master-slave communication system, slaves are not capable of initiating communication and spend most of their time waiting for commands from their respective masters. This leads to a hierarchy conducting a command chain from top to bottom, propagating through the RS485 bus from the main module to the motors. Discussions with the customer were imperative in making sure all three levels of the hierarchy were able to perform the tasks required to ensure proper operation of the global system. These tasks were listed as commands in [3], and are elaborated on in Section 4.2.

The main controller was programmed in *C++*, while the motor controllers were programmed in *C*. For both controllers, the company provided a framework of files to facilitate the connection of standard PCB peripherals like networking interfaces and other Input/Output (I/O) ports. Two separate networking interfaces were set up; one to allow the main controller to receive commands from the PLC, and one to allow communication between controllers across the bus channel (see Figure 4.4 for reference). Figure 4.5 shows how the main controller is embedded into the customer's machine; the blue ethernet cable is the connection from the PLC to the main controller, while the pink cables are the two RS485 communication channels connected to the bus.

4.2 Main controller

Since the main controller is essentially a central hub connecting different parts of the system, it is responsible for many different facets of communication. As was shown in Figure 4.4, this controller acts as a master to the motor controllers and as a slave to the PLC. This section covers the communication aspect of the system, while Section 4.3 goes into more detail on generating trajectories and operating motor drive chips to adhere

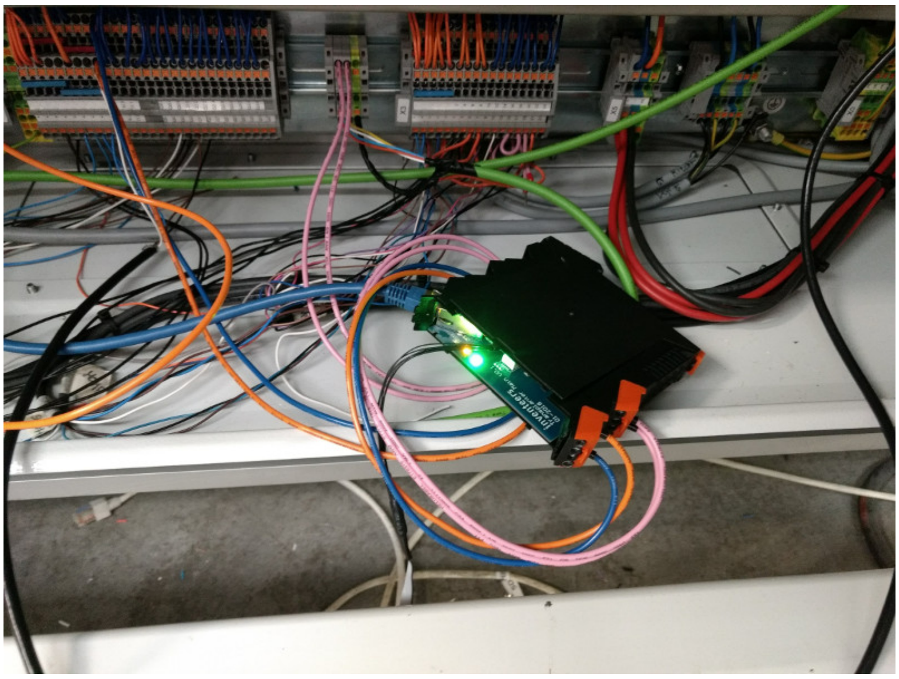


Figure 4.5: Main controller embedded in the Transplanter

to them. Setting aside various commands to debug, update and manually control the system, the most important commands for normal operation are the ones listed below and chronologically shown in Figure 4.6. These follow the principal operation of the system.

- Start calibration: propagates from top level to motor level to indicate that a calibration sequence (according to Figure 3.7) must be initiated by the system. The resulting motor positions and ranges are then communicated back upwards
- Set next positions: sets target positions for the motors, to be executed by all motors simultaneously once all have acknowledged their readiness
- Go: sent to all motors in order, after each one has received and acknowledged its new target position. A reply is sent by each motor once it has reached its destination

4.2.1 PLC communication

General purpose network communication typically uses Transmission Control Protocol/Internet Protocol (TCP/IP) which, in the context of the Open Systems Interconnection (OSI) model [19], sits at layers three and four - network and transport. This defines an extensive envelope in which to package any data to be sent, complying with rules regarding error correction, 32-bit addressing, time-to-live information and many other things. This yields a minimum total overhead of 160 bits per packet. By request from

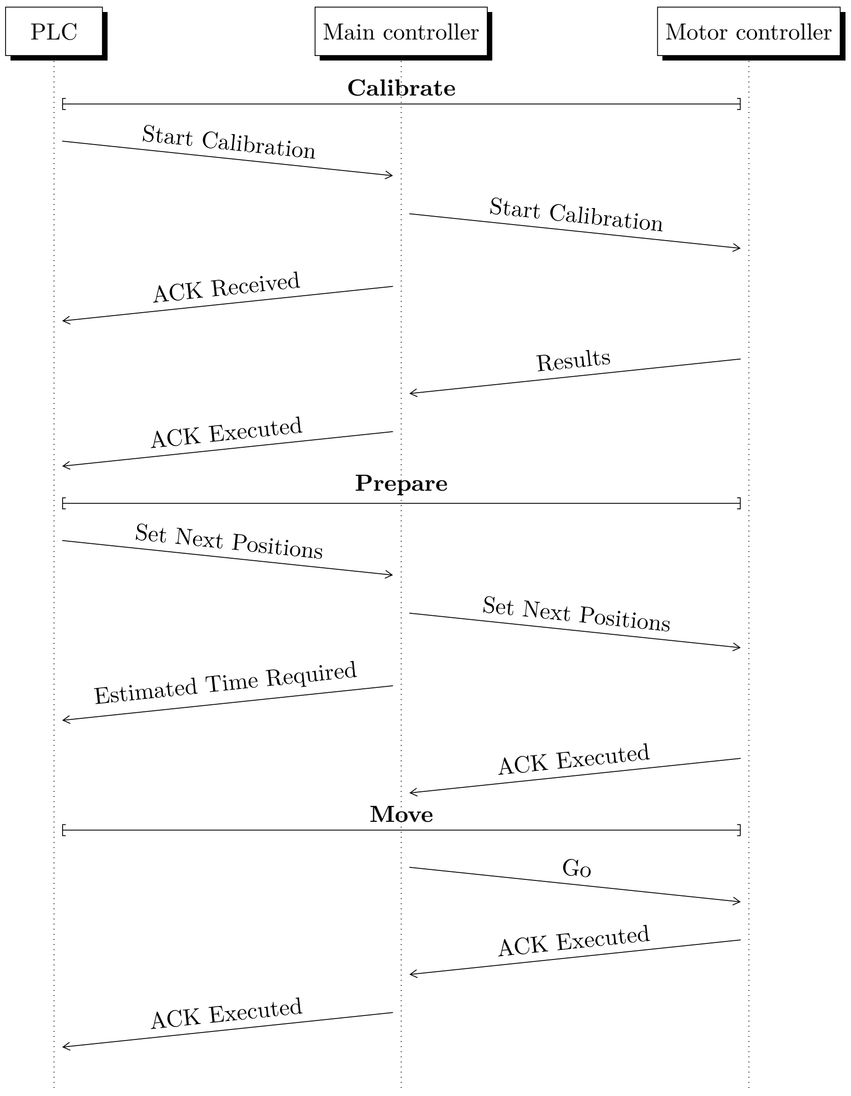


Figure 4.6: Sequence diagram of principal system operation

the customer, communication between the PLC and the main module was done through TCP/IP for the sake of simplicity as well as portability.

To facilitate communication and proper, in-time execution of commands, a message queuing interface was developed. Instead of risking loss of synchronisation by using a first in, first out implementation, control of the ordering of commands was left to the PLC. Within the data envelope as defined by the TCP/IP protocol, messages in this system have the following data fields:

- Command number: which one of the possible commands this message signifies
- UID: unique ID that identifies this specific message
- Parent UID: ID of parent message, which must wait for its children messages to be

executed before it is reported as done (ignored if UID = parent UID)

- Ref UID: ID of reference message, which must be executed before this command is allowed to start (ignored if UID = ref UID)
- Triggerpoint: boolean (1 or 2) that indicates whether a message is to be executed either after its reference message has started (1) or after it has been executed (2)
- Delay: time delay before the message can start, in milliseconds
- Parameters: numbers relating to details of the corresponding command. For example, an execution time paired with target locations for every motor, in the case of a *set next positions* command

The implications of these data fields are most easily understood by referring to the example in Section 4.2.4.

4.2.2 Motor controller communication

Simple microcontrollers like the ones described in this work can be considered *bare metal*; they execute instructions directly on logic hardware without utilising higher level management structures like in operating systems. Using RS485 in this system in the context of the OSI model, only the physical layer is defined. Needless to say, this has major advantages when it comes to throughput, latency and other overhead. The corresponding disadvantages are negligible - the need to write device drivers is offset by the fact that the company provided driver libraries for most common components, and complex concurrency is not required for the application. Ease of debugging might have saved some time in the long run, but that was a small price to pay.

A significant advantage of bare metal programming is customisability. Communication through RS485 is quite time-critical and would thus benefit from any possible latency optimisation. Instead of using the aforementioned TCP/IP packets, a custom, smaller packet format was defined that implements the minimum required functionality. It is built up as follows:

- Start of Frame - $0xAA_{\text{hex}}$ (1 byte)
- Packet length (1 byte)
- Source and target addresses (2 bytes)
- Unique ID and command number (2 bytes)
- Data (247 bytes max, dictated by 1 byte packet length)
- Checksum and End of Frame - $0xEF_{\text{hex}}$ (2 bytes)

This yields a total overhead of 64 bits; 96 bits less than in TCP/IP. Most of the benefit comes from the reduced target and destination address space, which must be 32 bits each in TCP/IP. In this case, however, given the limited number of motors in the machine, a single byte sufficed to cover the full address space. Address 255 being the broadcast address, address 254 was defined as the address of the main module. During calibration, each of the motors is numbered in turn, up from 0, allowing for a maximum of 254 motors.

4.2.3 Communication scenario: calibration

Calibration is initiated by having the PLC send a *Start calibration* command, using comnmand number 18 in the packet structure defined in Section 4.2.1. The corresponding parameters are a definition of the direction to move in to impel positive motion, depending on orientation of the motor as well as the bar, and a distance from which the ratio between distance and encoder steps can be calculated.

In Section 3.2.1.3, the calibration method used was defined. This required communication in response to broadcast messages. In a traditional master-slave system, this would not be possible. However, in a custom implementation of packet handling, redundancy checks and retransmissions, nothing is set in stone. A method was devised where a broadcast is replied to with an acknowledge, and the number of individual replies can be counted. Multiple replies at once would garble up the communication bus, producing noise that does not refactor to valid acknowledges. This then continues until circumstances are such that, by posing certain conditions, exactly one reply is produced, ending the corresponding phase of the calibration process. The process works as follows and is also illustrated by means of a Finite State Machine (FSM) in Figure 4.7, where Move Positive is the initial state:

Phase 1: Move to positive end: broadcast to move towards the positive end, then periodically check how many replies there are to the inquiry whether motors have a neighbour in the negative direction. When this number is exactly one, repeat a few times to eliminate false positives. After consecutive successes, conclude that all motors are lined up on the positive end, with only the most negatively positioned motor replying (*condition 1* in Figure 4.7).

Phase 2: Assign address: broadcast an address to set. Recipients check whether they already have an address (in which case the command is ignored) and whether they have recently been communicating with the main module through the previous command. If this is the case, the motor concludes that the address is meant for it and adopts it as its new address, while the main controller waits for verification that the new address has been set successfully.

Phase 3: Move to negative end: broadcast to move towards the negative end. Recipients check whether they have an address, in which case they execute the move command. Unassigned motors are requested to reply, but only if they have no neighbour in the negative direction, which immediately becomes the case once its neighbour is assigned an address and starts moving. Once again, only the most negatively positioned unassigned motor will be replying, and if exactly one reply is reliably received, the corresponding motor is assigned an address (*condition 2* in Figure 4.7). The main controller then bounces between phases 2 and 3 until phase 3 yields exactly zero replies (*condition 3* in Figure 4.7), signifying that all motors will have been assigned by then.

Calibration concludes by the main module sending each motor information about track length and magnet locations. In turn, each motor is sent information about either

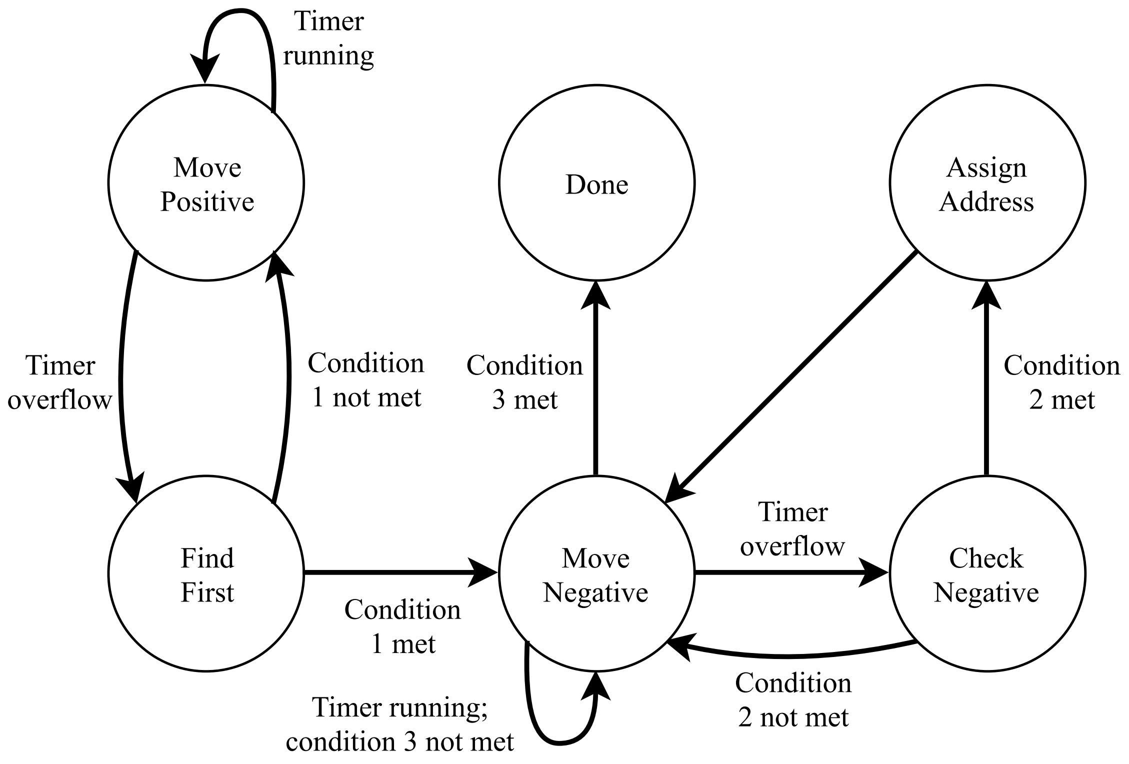


Figure 4.7: FSM of the calibration process

Condition 1	After positive move, all motors lined up on positive side; exactly one motor with just one neighbour
Condition 2	After negative move, all *unassigned* motors lined up on positive side, exactly one of which has just one neighbour
Condition 3	All motors assigned and lined up on negative side

Table 4.2: List of calibration conditions

the location of the track magnets or the total length of the bar. This information is combined with the total number of steps in each motor's full range to transform this range into the units supplied by the PLC, returning this information as a reply. Subsequently, checks are performed on the feasibility of this data, invalidating the calibration if a problem occurs. This invalidation also occurs if, after calibration, any motors are left unassigned on the bar. Information about the nature of any problems that might occur is communicated to the PLC.

4.2.4 Communication scenario: normal operation

Once the system has been calibrated, it is ready to receive commands from the PLC. A typical use case scenario with three grippers is shown in Table 4.3. The different commands have the following effects:

- **UID 1**: open all grippers after 100 milliseconds
- **UID 2**: take two seconds to move motors to dense locations at 110, 120 and 130 arbitrary units (defined by user), 100 milliseconds after the previous command has been executed. During this process, the bar carrying the motors is moved down towards the seedlings
- **UID 3**: selectively close grippers; the outermost grippers close while the middle one remains inactive, 100 milliseconds after the move of UID 2 has completed
- **UID 4**: take two seconds to move motors to sparser locations at 20, 50 and 80 arbitrary units, 1000 milliseconds after the grippers have closed. Extra wait time is implemented to allow the bar carrying the motors to move up before lateral movement is resumed
- **UID 5**: halfway into the move of UID 4, open the rightmost gripper to discard its seedling. This could happen if a bad seedling is detected through an external sensor connected to the PLC
- **UID 6**: open remaining gripper 500 milliseconds after the move of UID 4 has finished

Command	UID	Parent UID	Ref UID	Triggerpoint	Delay	Parameters
21	1	1	1	2	100	0:0:0
19	2	2	1	2	100	2000:110:120:130
21	3	3	2	2	100	1:0:1
19	4	4	3	2	1000	2000:20:50:80
21	5	5	4	1	1000	0:0:1
21	6	6	4	2	500	0:0:0

Table 4.3: Message Queuing Interface usage example

4.3 Motor controller

The motor was equipped with an encoder, which is a sensor mounted to the axis that keeps track of how far it has rotated. This produces a discrete and accurate amount of pulses; 500 in the case of the encoder used. These pulses were set up to generate hardware events on the motor controllers, allowing them to bypass synchronous counting which might either garble up the system with interrupts or allow for pulses to be missed if other interrupts interfere. A framework like in Figure 3.11 was programmed, transforming these encoder pulses into an actual position $\hat{q}$, comparing this to a requested position q, and feeding the difference ϵ between these two into a control block, which in turn computed an input voltage value u to supply to the motor. Figure 4.8 shows how the motor controllers are embedded into the machine, mounted on top of the motors together with all the associated electronics.

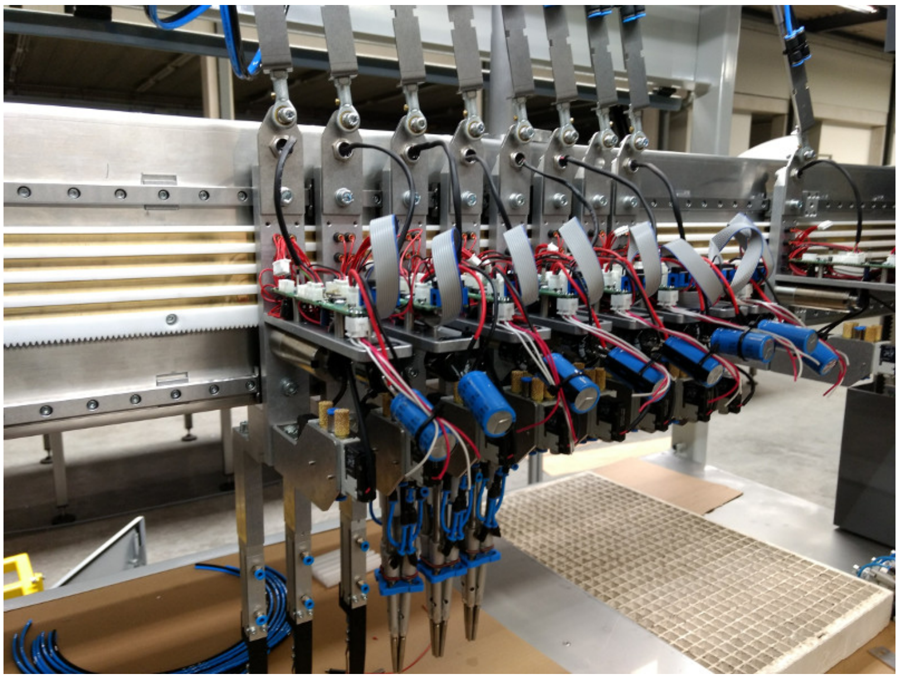


Figure 4.8: Set of mounted grippers without casing

4.3.1 Motor control

To control the motors through Proportional, Integral, Derivative (PID) filtering, proper values for the filter parameters K_p, K_i and K_d had to be found to be filled in for Equation 3.15. Three ways to obtain these values were identified:

- Heuristically iterating values to come to a satisfactory result
- Creating a theoretical model of the system to obtain a system transfer function that can be tuned through algorithms
- Driving the system with a step input to obtain step response data, which can, in turn, also be used to obtain a transfer function tuneable through algorithms

In the initial implementation, the heuristic iteration method was applied. One example of such an approach is the Ziegler-Nichols Ultimate Gain method as described in [20], which is a method of bumping up the gain until the system barely oscillates, at which point the characteristics of these oscillations can be used to estimate PID filter values that would produce a stable system. However, since this method was unable to produce satisfactory filter values, the decision was made to use transfer function estimation instead.

The first of these estimations is the theoretical model. Starting from Figure 4.1 to model a motor winding, this was built on and combined with the physical model for a rotational mass as proposed in [21]. These equations can then be combined to form a third order transfer function for a DC motor with voltage input v and position output q, as shown in Equation 4.2. Note that this model assumes a constant magnetic field and equality of the torque constant and back-emf constant (denoted by K). In this model, b is a frictional constant. The step response output of this system was obtained through simulation and is shown in Figure 4.9. Using the PID Tuner tool supplied by MATLAB [22], a PID filter was parametrised to yield an arbitrarily low overshoot and a settling time in the order of magnitude of tens of milliseconds. Valid filter values for this are shown in Equation 4.3.

$$\frac{Q(s)}{V(s)} = \frac{K}{s((Js+b)(Ls+R)+K^2)} \tag{4.2}$$

$$\begin{cases} K_p = 0.1952 \\ K_i = 1.0017 \\ K_d = 0.0095 \end{cases} \tag{4.3}$$

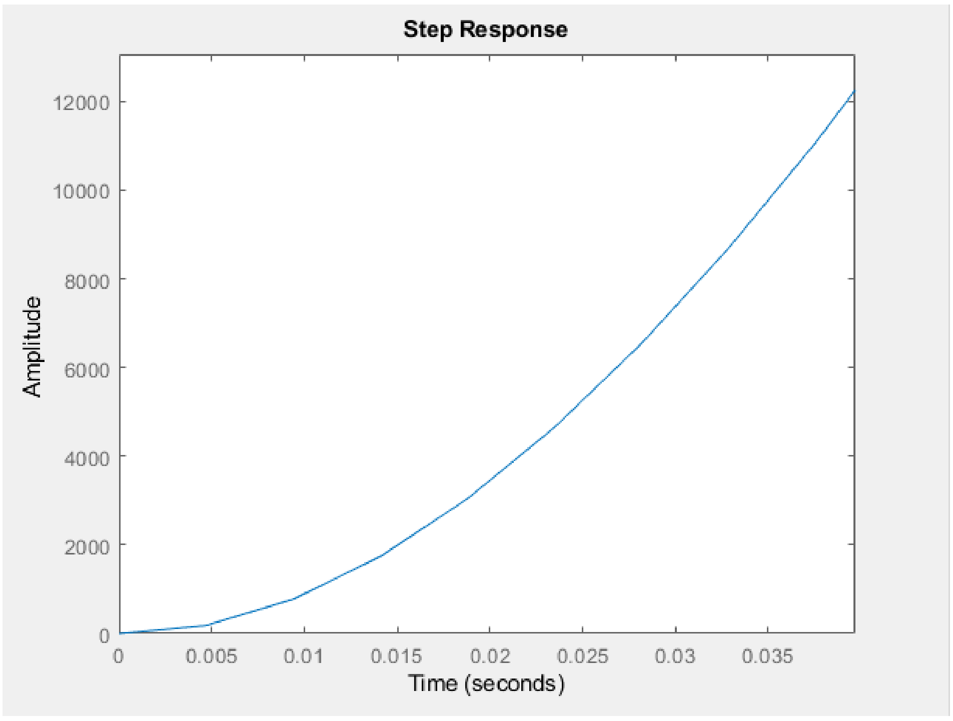


Figure 4.9: Step response of the model, showing its start-up characteristic, with displacement in encoder pulses on the y-axis

There are some downsides to using a theoretical transfer function. Assumptions like the magnetic field being constant cause deviations from real world values, however small they may be. More importantly, estimating values to be filled for the model mostly consists of weighted guesswork, leading to an unreliable response. The second method of obtaining a transfer function should be more reliable. To achieve this, a constant input power of roughly 70% was supplied to the input of the system, measuring the generated trajectory as well as the output it supplied in the process. Using the response this yielded, a transfer function was estimated using the PID Tuner tool. The result of using 300 iterations of the Interior-Point algorithm [22] is shown in Equation 4.4, with $K_g = 74.122$, $T_{p1} = 0.0398$ and $T_{p2} = 0.021093$, and Figure 4.10 showing the 99% fit to estimation data. Surprisingly, a two-pole (and thus second order) gave a good fit for a system estimated to be third order by models. The PID filter parameters were calculated to yield an overshoot and settling time similar to the parametrisation for the theoretical model, and are shown in Equation 4.5.

$$\frac{Q(s)}{V(s)} = \frac{K_g}{(1 + T_{p1}s)(1 + T_{p2}s)} \tag{4.4}$$

$$\begin{cases} K_p = 0.9837 \\ K_i = 17.7557 \\ K_d = 0.0136 \end{cases} \tag{4.5}$$

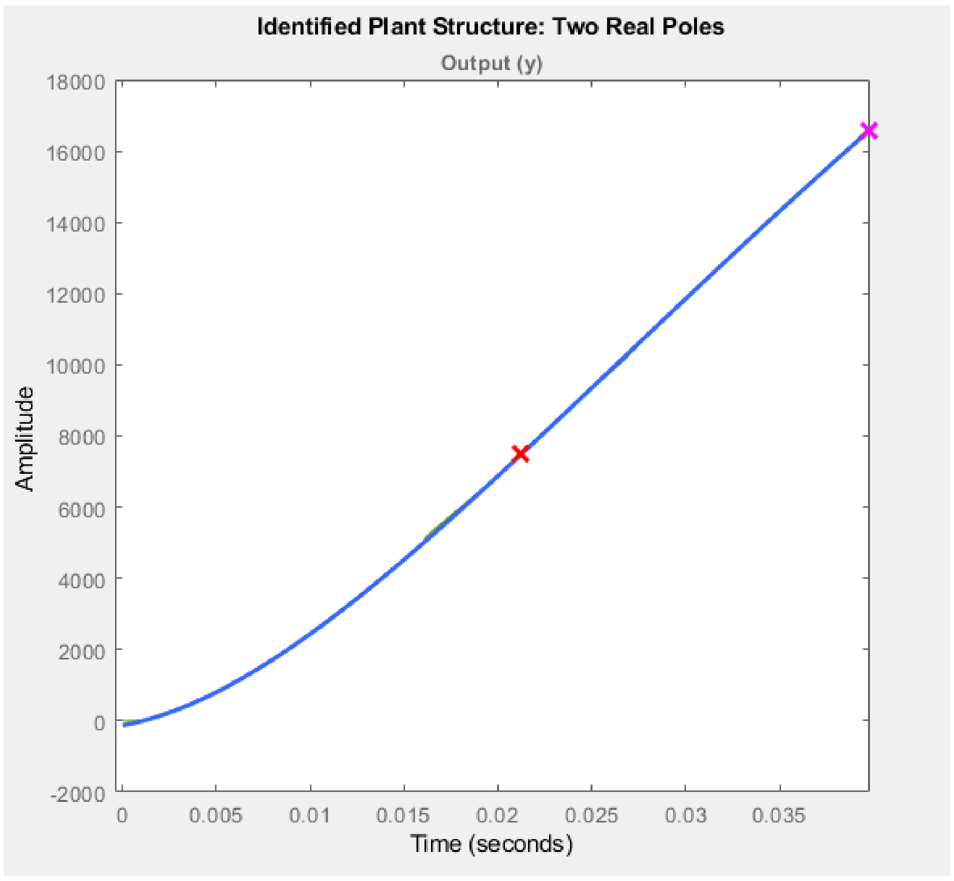


Figure 4.10: Measured step response used to estimate the transfer function of the system, with displacement in encoder pulses on the y-axis. The step response of the resulting transfer function is plotted over the measured data

A striking similarity is seen between Figures 4.9 and 4.10. This suggests that, despite the difference in order between the resulting transfer functions, they are indeed quite similar. Since a relatively small value was used for L, this might cause the third pole of the system to be relatively low in magnitude. Regardless, there are two major differences between the figures. Firstly, the measured model reaches a higher amplitude, and secondly, the measured system is quicker to respond to the step input. This could be due to the fact that the model takes both friction and load mass into account, while the measurement data was obtained in an unladen configuration.

4.3.2 Trajectories

Generating realistic trajectories posed a trade-off in computational complexity versus the amount of constraints that could be enforced on it. Third order parabola trajectories were a reasonable choice, as they generate a subtle acceleration and deceleration curve (or low jerk) while guaranteeing continuity in the position, velocity and acceleration domains. Equation 3.5 was rewritten with $t_0 = 0$, $\dot{q}_0 = 0$, $\dot{q}_f = 0$ and expressed in terms of $t_f =$ execution time, $q_0 =$ current position, $q_f =$ target position to be solvable for the polynomial constants. This yielded Equation 4.6. Unfortunately, the division terms in a_2 and a_3 as well as the t multiplication terms in Equation 3.5 make it problematic to use integers in the implementation, forcing the use of floating point numbers. However, profiling the computation of a third order polynomial proved that this only took about 7 µs every iteration, which is likely to be acceptable, especially considering that the frequency of this computation does not necessarily have to be very high.

Figure 4.4 shows an example of a trajectory generated by the on-chip software, which was extracted after generation by sending the resulting time data all the way back up through the communication hierarchy. This method was also used to generate the output trajectories and location data shown in Chapter 5.

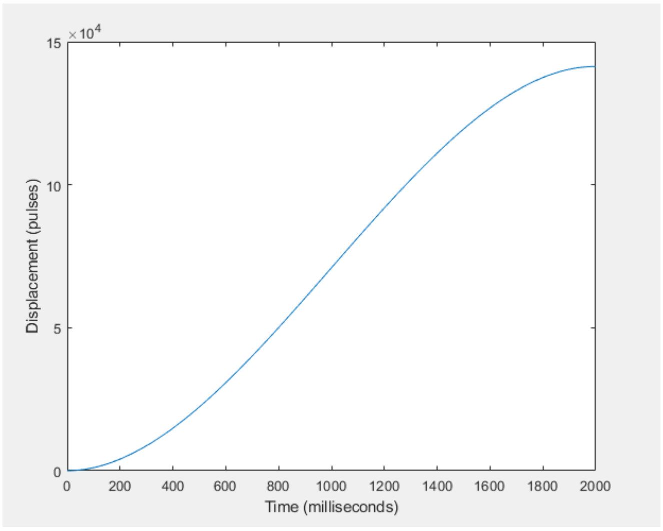


Table 4.4: Parabolic trajectory as measured after on-chip generation

$$\begin{cases} a_0 = q_0 \\ a_1 = 0 \\ a_2 = \dfrac{3(q_f - q_0)}{t_f^2} \\ a_3 = \dfrac{-2(q_f - q_0)}{t_f^3} \end{cases} \tag{4.6}$$

Results 5

This chapter describes the results generated over the course of the project, quantitative or otherwise. Section 5.1 discusses the problems encountered with communication during testing, and Section 5.2 attempts to quantify the system's performance with respect to following trajectories.

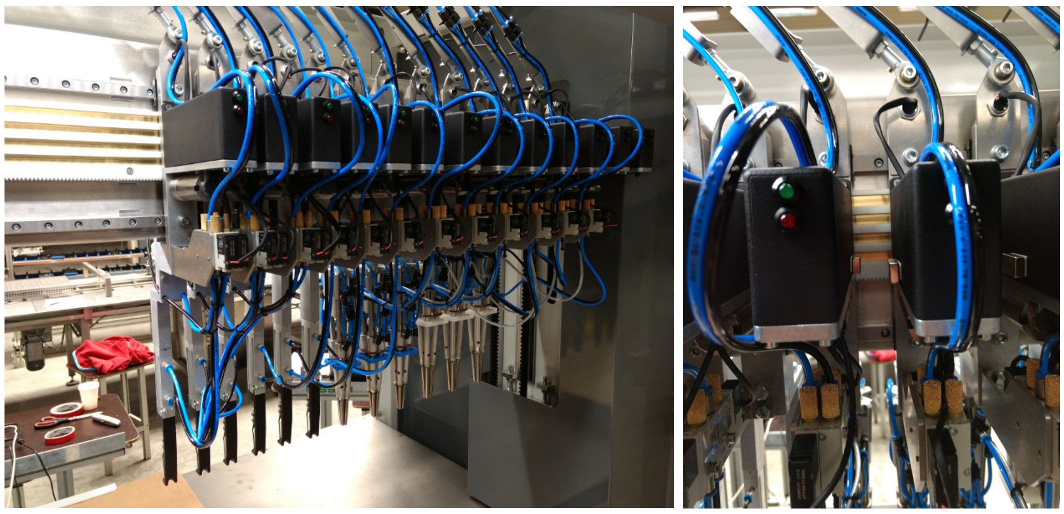


Figure 5.1: Overview of the new Transplanter, with cases mounted on gripper electronics and a close-up of magnet mounting on the right

5.1 Communication

As it turned out, communication proved difficult with the intended approach. The algorithm discussed in Section 3.2.1.3 worked perfectly well with a limited number of motors on the track. The motors would properly receive messages broadcast by the main controller, and replied to them as expected. However, as the number of motors increased, so did the number of replies to broadcasts. Eventually, this garbled up the communication bus (as seen in Figure 5.1) to such an extent that some motors were no longer able to receive the calibration commands, effectively excluding them from the process altogether. Naturally, this completely obstructed the calibration and the system was caught in an infinite loop of communication errors.

To remedy this, the (unique) addresses of all motors were hard-coded in the motor controller software. Subsequently, all broadcasts were changed to targeted communication, using these unique addresses. During the calibration process, the motors were then merely sorted instead of assigning ordered addresses. Using this approach, the algorithm written could be resumed as usual once the sorted list of motors was compiled.

Another challenge encountered was magnet alignment. To allow for motor proximity detection, magnetic sensors were placed on both sides of every motor. Needless to say, the motors were mounted with magnets on opposing sides to allow them to be seen by their neighbours. Accurately positioning the sensors proved even more difficult than positioning the magnets. This was due to the fact that they were connected to the Printed Circuit Board (PCB) through long, pliable pins. As seen in the right picture in Figure 5.1, some of the magnets were placed outside the plastic casing. This was done to allow motors equipped with large grippers to be able to detect their neighbours despite the increased minimum distance between them.

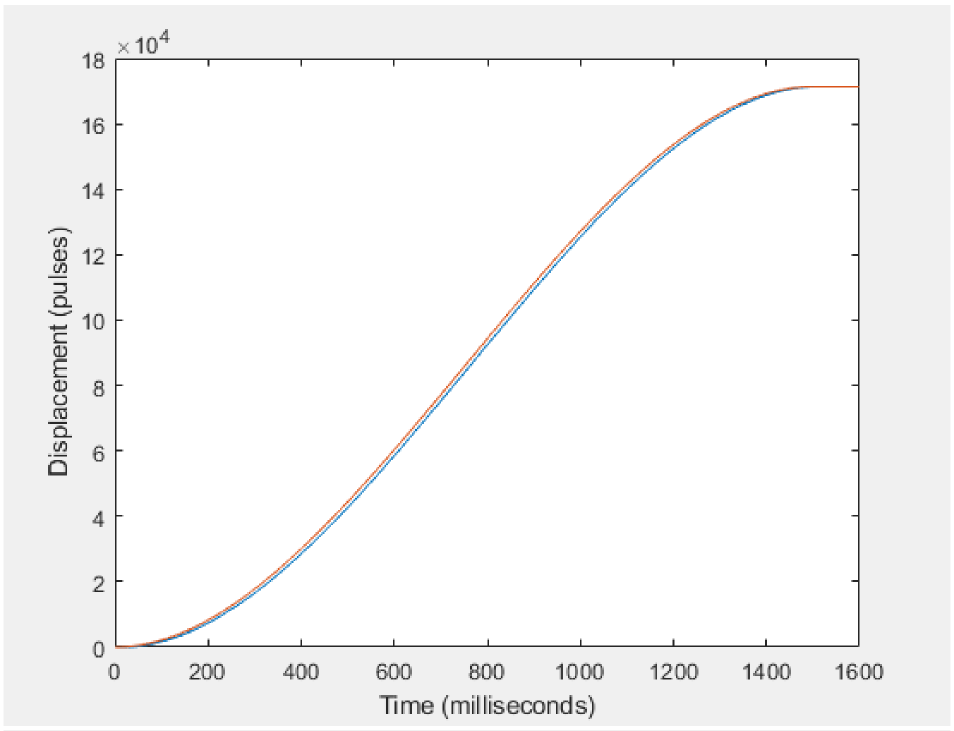
5.2 Trajectories

Taking target locations and expected durations as inputs, parabolic target trajectories were generated and supplied to the system. The difference between a motor's actual location and expected location was then used to determine the input power supplied to the motor. A typical scenario is shown in Figure 5.2. In this case, the motor is given 1.5 seconds to move to a specified location. Over the course of the trajectory, the difference between the planned trajectory (in orange) and the actual trajectory (in blue) is relatively small. This indicates that the parameters have been matched properly. Figure 5.3 shows a different example. Here, the motor reaches its upper velocity cap despite falling behind on the trajectory. As a result, the motor finishes long after the expected duration of 1 second, paired with a significant overshoot. This indicates a mismatch between the motor characteristics and the input parameters; the motor was unable to move at the requested velocity.

The figures shown were obtained by driving the motors proportionally with respect to their generated ideal trajectories, as other implementations were quick to produce significantly more overshoot than predicted by the models. Using only a proportional factor showed promising results, given that specific edge cases were covered (which was not the case in Figure 5.3). These edge cases can be summarised as follows:

- The maximum target velocity of the trajectory supplied must be attainable by the motor (i.e. the distance must be sufficiently short and the time granted sufficiently long)
- If a motor's normal operation is interrupted because it runs into an obstruction, it must recalculate its trajectory to obtain a new start-up curve once the obstruction is gone

The measured velocity in Figures 5.2 and 5.3 was obtained by differentiating the displacement data and filtering it with a minimum order low pass filter. Despite these measures, it still looks quite noisy. However, since this noise is also present in the generated expected velocity, it is safe to conclude that it is introduced during measurement, through quantisation and from lack of synchronisation between updating values and sampling them; the former runs on a soft timer while the later runs on an interrupt.



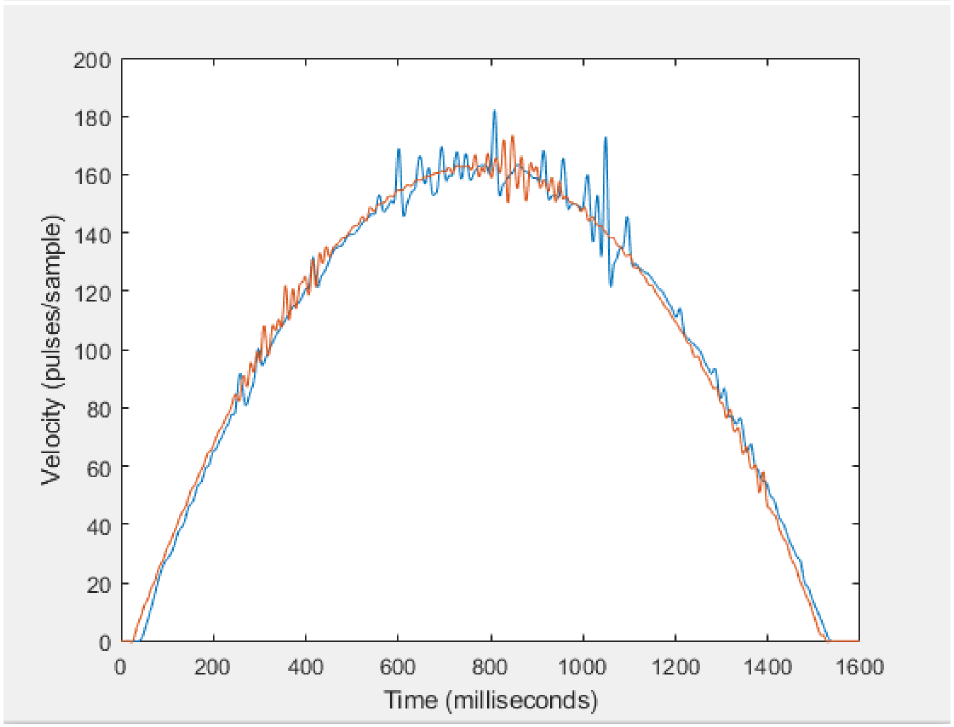


Figure 5.2: Measurements of the planned trajectory (orange) and actual trajectory (blue) in the top figure, and the expected velocity (orange) and actual velocity (blue) in the bottom figure

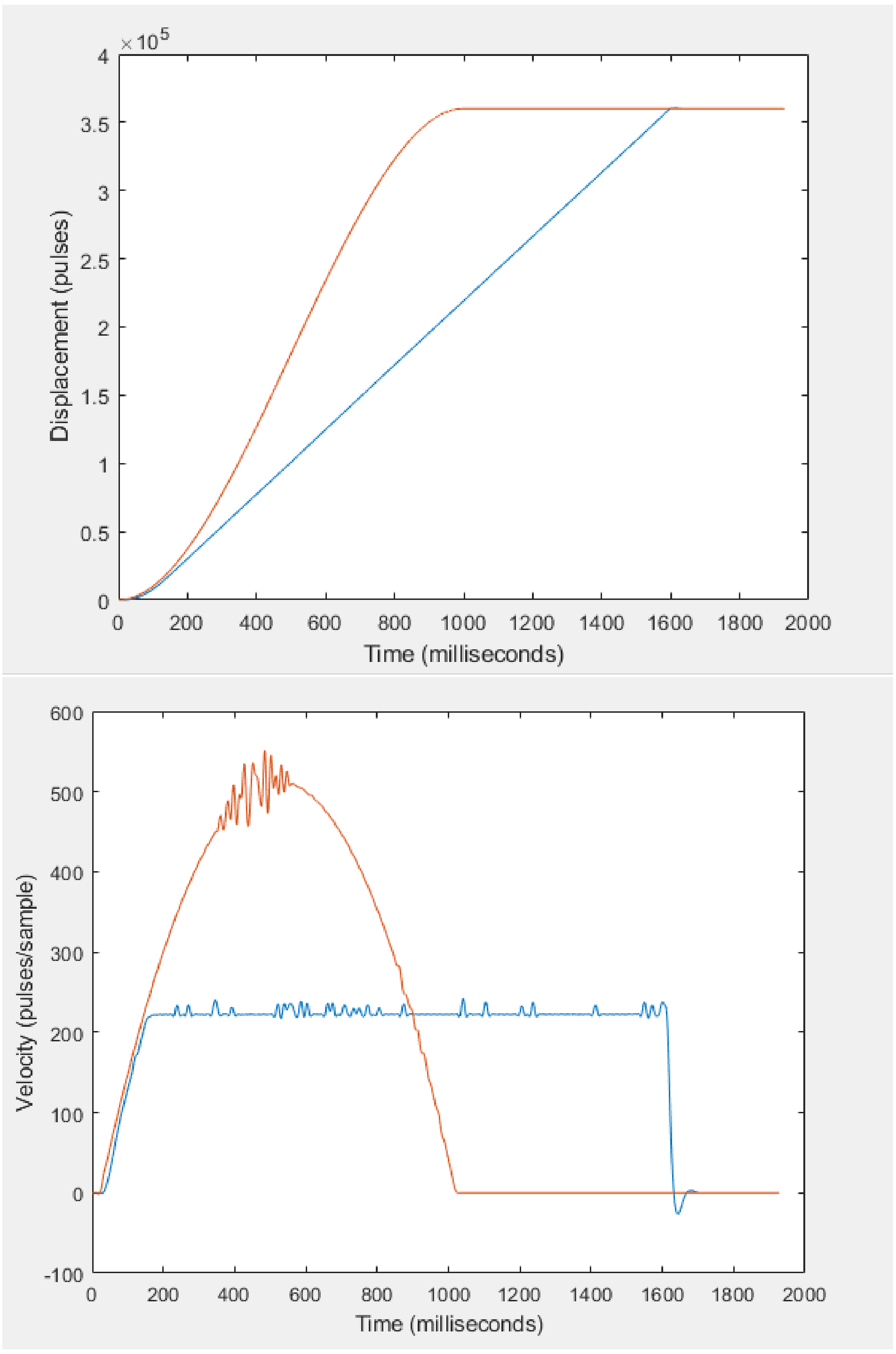


Figure 5.3: Measurements of the planned trajectory (orange) and actual trajectory (blue) in the top figure, and the expected velocity (orange) and actual velocity (blue) in the bottom figure

Conclusions & recommendations 6

In this chapter, the original research questions as listed in Chapter 1 are looked back upon to see how they can be answered using the information gathered in the rest of this thesis. A list of contributions by the author is compiled and, finally, a number of recommendations for future work are done.

6.1 Conclusions

The research questions are tackled one by one, beginning with the sub-questions:

What should an entity's trajectory look like, how is it influenced by the inertia of its payload, and how might this be compensated?

- A parabolic (polynomial) trajectory of third order provides sufficient leniency with regards to motor jerk, but is lacking in positional efficiency due to its brief moment at maximum speed
- A trapezoidal trajectory makes efficient use of the motor's maximum speed at the cost of higher jerk. A combination of both types appears to be the best option (further discussed in Section 6.3).
- Considering the total mass of the system, the inertia of a payload held by a gripper has no considerable effect on the behaviour of the system

What might the characteristics be of a cost-effective and accurate electronic system implementing this?

- By replacing the servo drives of the previous system with a self-made embedded system on a custom Printed Circuit Board (PCB), the cost of the existing system was significantly reduced
- The wires of the existing system were replaced by transferring all power and communication to bus rails, which makes repairs simple, allowing users to circumvent the cost of technicians and downtime
- Controlling the motors through an electronic closed-loop feedback system, the accuracy achieved was only limited by the resolution of the rotary encoder

In a physical implementation of the system, what might go wrong, and how can this be accounted for?

- At the start of the project, a list of potential problems was compiled and this list was added to whenever new things came up (Section 2.4)
- Fail-safes have been built in to detect problems during calibration, like motors being left unassigned after calibration was seemingly successful and overheating, overcurrent and stall detection that can stop the motor if further operation might otherwise cause damage
- Proximity sensors were integrated into the system, triggering interrupts whenever a neighbour comes too close

Using electronic control, how can multiple entities with certain inertial characteristics efficiently be moved along a linear rail at high speed without collisions?

In conclusion, returning to the main research question, this work presented a list of technical requirements based on a business case of making the system cheaper and more efficient (Chapter 2) followed by a design chapter, where these requirements were taken into consideration and translated into the parametrisation of a system that is able to meet these requirements (Chapter 3). The implementation of this design, with the relevant trade-off decisions made, was described in Chapter 4, and finally, performance data measured from the system was presented in Chapter 5. The resulting design consists of a hardware and software architecture, which is capable of executing commands supplied by the user, and a functional prototype was built and presented at the Greentech exhibition in Amsterdam on the 12th of June, where the machine ran for three days before the eyes of thousands of visitors.

6.2 Contributions

The project described in this thesis was a collaborative effort, with multiple people from different disciplines involved contributing their own ideas and implementations. Listed below are some of the key contributions of the author of this work:

- Modelling and parametrisation of system requirements in hardware as well as software
- Design and implementation of:
  - the calibration algorithm
  - ideal trajectory generation
  - motor control

The Message Queuing Interface described in Section 4.2.1 is a vital part of the system and, though the author of this work was involved in some of its developments and actively used its architecture to implement new functions for it, its main design and implementation was not their responsibility.

6.3 Recommendations for future work

Although a functional prototype was observed live at the Greentech exhibition in Amsterdam, a significant amount of work remains to be done to prepare the machine design for commercial success. This section provides recommendations for improving the existing implementation.

6.3.1 Calibration

As noted in Chapter 5, the broadcast method described in this work proved incapable of handling the required amount of motors on a single bus. A temporary solution was to hard-code the address of every motor controller, counting up from 1. Needless to say, this is not scalable at all and violates the modularity cornerstone of the product. Therefore, it is very important that collision detection is implemented as described in Section 3.2.1. Furthermore, if a Proportional, Integral, Derivative (PID) approach is opted for, an extra calibration phase should be added where the motors are sequentially fed a step input and their resulting displacement is measured. This data could then be used to estimate tuning parameters like in Figure 4.10. However, since this estimation was done on a modern, multithreaded computer system in MATLAB, more research is required to determine whether the microcontrollers used are capable of handling these heuristic algorithms in a reasonable amount of time.

Another crucial challenge during calibration was the placement of magnets for proximity sensors. Since the sensors were analogue, tuning their sensitivity was quite troublesome. Perhaps a differential algorithm, looking at the rate of change of this analogue value, would perform better, but it would still require correct alignment to a certain extent. Since mass production is unlikely to fix this problem completely, more research should be done to look into alternative proximity sensor implementations, perhaps even beyond the options offered in Table 2.3.

Finally, an extra calibration phase should be introduced during which an encoder pulse measurement determines the velocity resulting from a certain input power, as this value will be influenced by material wear as well as irregularities in friction caused by bolt tightness and track structure. Considering how all motors must be able to move along the rail simultaneously, the system will then have to use the weakest link (i.e. the motor producing the least velocity as a result from its input electrical power) to base the maximum velocity on. Since the power supplied to each motor was proportional to its positional difference, this calibration value could then be used as an offset to make sure that the resulting velocity is the same for each motor. In case the value found is significantly lower for a single motor, the main controller should report this to the Programmable Logic Controller (PLC) to prompt the user to replace that motor or perform maintenance on it.

6.3.2 Motor control

As was shown in Figure 5.3, the implemented motor controller was unreliable if edge cases were not caught. Catching these should be guaranteed somewhere in the communication hierarchy, invalidating supplied trajectories that are impossible to follow. A disadvantage of this is that the limited maximum velocity of a parabolic trajectory severely limits the capabilities of the motors. Though trapezoid trajectories have a harsh start-up curve with relatively high jerk, their intermediate state of cruise velocity could make them a lot faster than parabolic trajectories. To combine the advantages of both, it would likely be better to start each movement with a parabolic acceleration curve, which then transitions into a trapezoid's cruise velocity once the start of the trajectory has been managed with minimum jerk. Near the end, where the downward slope of the trapezoid velocity would be, the deceleration curve of the parabolic trajectory could then be used again.

It is still unclear whether the poor performance of integral and derivative action in the PID controller was caused by having overlooked some characteristic of the system during design or implementation. While full PID control is not necessary to accurately control the system, more research should be done to find the underlying cause of this discrepancy.

Bibliography

[1] P. Acarnley, *Stepping Motors - A Guide to Theory and Practice*, 4th ed. The Institution of Electrical Engineers, London, 2002.

[2] *NEMA ICS 16, Industrial Control and Systems - Motion/Position Control Motors, Controls, and Feedback Devices*, National Electrical Manufacturers Association, 1300 N. 17th Street, Rosslyn, Virginia 22209, 2001.

[3] *Communicatie protocol Transplanter*, Inventeers, Le Pooleweg 7, 2314 XT Leiden, The Netherlands, dec. 2017.

[4] *CN70 - Reflective Optical Sensor with Transistor Output*, Vishay Intertechnology, Inc., 63 Lancaster Avenue Malvern, PA 19355-2143, jul. 2012.

[5] *DRV5032 Ultra-Low-Power Digital-Switch Hall Effect Sensor*, Texas Instruments, Post Office Box 655303, Dallas, Texas 75265, nov. 2017.

[6] *AC Servo Drives, $\Sigma - V$ Series Product Catalog*, Yaskawa America, Inc., 2121 Norman Drive South, Waukegan, IL 60085, may 2017.

[7] *Ezi-Step Micro Stepping System*, Fastech Co., Ltd., 811E Plano Parkway, Suite 110A, Plano, TX 75074, nov. 2015.

[8] *QMOT Motor QSH2818 Manual*, Trinamic Motion Control GmbH & Co. KG, Sternstraße 67D - 20357 Hamburg, Germany, nov. 2007.

[9] *DCX 26 L Graphite Brushes*, maxon motor ag, Brünigstrasse 220, 6072 Sachseln, Switzerland, may 2017.

[10] *EC Series Slotless Brushless DC Motor EC2864*, Vishan Motor, 5th Floor, building C, Qiaotong Yuanling Industry Park, Shiyan Town, Bao'an, Shenzhen China, 2017.

[11] Thomas H. Cormen, Charles E. Leiserson, Clifford Stein, Ronald L. Rivest, *Introduction to Algorithms*, 1st ed. Mit Press Ltd, july 2009.

[12] *AT91SAM ARM-based Flash MCU, SAM4S Series - Preliminary Summary*, Atmel Corporation, 2325 Orchard Parkway, San Jose, CA 95131, jul. 2012.

[13] R. Smith. Quick reference for rs485, rs422, rs232 and rs423. [Online]. Available: http://www.rs485.com/rs485spec.html

[14] S. Mitkin, *DRAKON, the Human Revolution in Understanding Programs*, oct 2011.

[15] Carlo Raoul Maria Ghioni, "A transplanting machine," European Patent EP 1 652 418B1, 10 28, 2009. [Online]. Available: https://patents.google.com/patent/EP1652418B1/nl

[16] C. Melchiorri, *Trajectory Planning for Automatic Machines and Robots*, 1st ed. Springer-Verlag Berlin Heidelberg, 2008.

[17] W. Ames, *Numerical Methods for Partial Differential Equations*, 2nd ed. Academic Press, New York, 1977.

[18] *Atmel XMEGA AU Manual*, Atmel Corporation, 2325 Orchard Parkway, San Jose, CA 95131, apr. 2013.

[19] *Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*, ISO/IEC, nov. 1994.

[20] Gene F. Franklin, J. David Powell, Abbas Emami-Naeini, *Feedback Control of Dynamic Systems*, 6th ed. Pearson PLC, 2010.

[21] University of Michigan. Control tutorials for matlab and simulink - motor position: System modeling. [Online]. Available: http://ctms.engin.umich.edu/CTMS/index.php?example=MotorPosition&section=SystemModeling

[22] The MathWorks, Inc. Designing pid controllers with pid tuner. [Online]. Available: https://nl.mathworks.com/help/control/getstart/designing-pid-controllers-with-the-pid-tuner-gui.html