UNIVERSITY OF OSLO
Department of Informatics

A Stewart Platform Based Replicating Rapid Prototyping System with Biologically Inspired Path-Optimization

Master Thesis (60 pts)

Lars Skaret

May 2, 2011

Abstract

The idea of self-replication in robotics can be traced back to John Von Neumann in 1966. While total self-replication is still be many years away, research regarding this topic is underway at Bath University in England under the RepRap project name. This thesis aims at elaborating on the work done at Bath University and looks at the possibility of taking the idea one step further. To do this, a suitable robotic manipulator has been chosen for study, the Stewart platform. Design and simulation has been implemented and studied. Designing the platform resulted, among other things, in many interesting challenges, a printed prototype of a Stewart platform/arm and an interesting experimental design that will make a larger workspace possible. While many design solutions have been suggested, the design needs to be further studied, physically implemented and tested to prove it to be an alternative to the RepRap. The manipulator should have both CNC milling and 3D printing capabilities. The simulator has applications that make studying the movements of the platform according to a predefined G-code path possible. While making the end-effector follow a path, the simulator will also deny the 6 legs of the Stewart platform to move too fast or extend to long. Creating the simulator created a better understanding of the Stewart platform's mathematical characteristics.

Another aim of this thesis is to shorten the length of the tool path using biologically inspired path-optimization. A genetic algorithm and an ant colony optimization algorithm have been implemented to improve the tool path, and the results have been studied. A single G-code file was tested and the algorithms both managed to decrease the total length by about 6.5% of the total length, or 67% of inactive length, or 180 mm, or 7.5 seconds with a tool speed of 24 mm/s. The algorithms performed quite well, but should be tested on longer tool paths to investigate whether the optimization method might save a considerable amount of machining time.

Acknowledgements

This thesis has allowed me to explore the very interesting topic of robotics. For this, I wish to thank the Robotics and Intelligent Systems (ROBIN) research group. They have enabled me to study some of the most interesting fields I have experienced in my academic endeavors.

I wish to thank my two supervisors, Mats Høvin and Kyrre Glette for guidance, but also for allowing me the freedom to influence the choice of topic to a large degree. Head engineer at ROBIN, Yngve Hafting, deserves thanks for helping me with the practical challenges in creating printable designs and using 3D printer software.

I also wish to thank my two closest peers at the University, Magnus Lange and Akbar Faghihi Moghaddam (Shahab) for insightful discussions and useful tips. Last but not least, deserving the biggest appreciation is my Anne-Catherin for her patience during my late nights at the lab.

Contents

List of Figures

List of Tables

Abbreviations

CNC - Computer Numerically Controlled

RepRap - replicating rapid prototyper

CAD - Computer Aided Design

CAM - Computer aided Manufacturing

DOF - degrees of freedom

PM - parallel manipulator

SM - serial manipulator

R - rotational joint

P - prismatic joint

U - universal joint

TSP - Traveling Salesman Problem

FDM - Fused Deposition Modeling

LOM - Laminated Object Manufacturing

SLS - Selective Laser Sintering

STL - Stereolithography

PKM - Parallel Kinematics Machine

FFF - Fused Filament Fabrication

PNG - Portable Network Graphics

FPS - Frames per second

SUS - Stochastic Universal Sampling

PMX - Partially Mapped Crossover

ACO - Ant Colony Optimization

Chapter 1

Introduction

3D printing and CNC milling have been around for a long time and have to some extent even reached the consumer market.[61, 38] In the industry, 3D printing and sometimes CNC milling are used to create prototypes in a quick and cost effective manner. For the hobbyist (or amateur), the use is more directed towards creating almost anything, including prototypes, a building set for an RC airplane, parts for a robot, a box-container or simply a spoon. This thesis focuses on the hobby aspect of 3D printing and CNC milling that translates into some keywords that are vital to be maintained: affordability, simplicity, open-source and size. These are in addition to those common to 3D printing and CNC milling such as accuracy, rigidity (especially for CNC milling), speed and time.

This thesis has one foot planted in the theoretical world of path-optimization, kinematics and simulation. And the other foot planted in the practical world of design, motors and electronics. Both of these aspects have been explored as they are both essential for the development of a robot manipulator.

1.1 Self-Replication

The idea of self-replication of non-biological entities was first introduced by John von Neumann in the late 40s with a thought experiment and later published in the 1960s.[7] The concept of self-replication for *biological* entities is as old as life and it is what life is based on as all living creatures replicate themselves. One well developed project based on self-replication of non-biological entities is the RepRap project.

With self-replication there are, as I see it, two ways to go, simplicity or complexity. The complex viewpoint results in a large machine, or even an autonomous factory, capable of producing almost anything. This is an area that is very costly and thus hard to research. The simple aspect is to start with a simple machine and try to make it create parts of itself. This is what the people at Bath University have done with their RepRap project. RepRap is short for Replicating Rapid Prototyper and is basically a small 3D printer that is quite affordable. As the machine is so mechanically simple, the challenge is to make it able to do more things without sacrificing its simplicity. For a highly complex machine the challenge would have been the opposite, to make

it simpler without sacrificing functionality.

1.2 Goals of the thesis

With the RepRap project as a stepping stone, this thesis will mainly investigate the possibilities of increasing functionality without sacrificing simplicity. To do this, a type robotic manipulator has to be chosen and studied. Thus, the entry point of this thesis is the hobby world of 3D printing and CNC milling, exemplified with the RepRap project. To study the robotic manipulator, three goals have been created:

1. Create and compare a new design with the RepRap 3D printer developed at Bath University, England. The new design should be an improvement in certain fields as discussed in chapter 2.4.3.

2. Create a simulator to explore the robot manipulator and allow further experiments.

3. Look at the tool path-optimization (minimizing the length of the tool path) problem and implement biologically inspired algorithms to solve it.

The tool path-optimization problem has been included for two reasons. It is relevant to 3D printing and CNC milling, shortening the tool path will also shorten operation time. Secondly, it is an interesting research topic that has received attention lately. See chapter 2.7 for a discussion. The three goals suggest different approaches to the world of the robotic manipulator. It is believed that this will give a better insight as several aspects gets highlighted.

1.3 Outline

As mentioned, the thesis is more or less divided in three; however the three topics overlap each other in many areas and are connected. After the introduction, the thesis continues with a chapter on the background of the different topics. Also, based on a discussion, the Stewart platform is chosen as the robotic manipulator to study. The next chapter, Tools, deal with the different tools used in the three main chapters, Design, Simulator and Biologically Inspired Path-Optimization. The Design chapter presents different designs created with CAD software and discusses many aspects relating the realization of the designs and how the design compare to the RepRap. The next chapter, Simulation, moves a bit away from the practical world and investigates some mathematical theories behind the Stewart platform. Also, to implement the simulator, G-code is studied and a G-code parser is created. The tool path length is tried to be shortened in the chapter on biologically inspired path-optimization. The movements where the tool is inactive are studied and two algorithms are implemented and tested with a G-code file. The results from the tests are discussed and the applicability of the algorithm and type of path-optimization is assessed. The final chapter concludes the three main chapters and tries to use the experience from them to decide whether the Stewart platform as presented can be an alternative to the RepRap. Further work is also discussed in the last chapter.

Chapter 2

Background

Much of the background theory is presented in this chapter. Also, some discussion regarding central aspects of this thesis is made. First, rapid prototyping, 3D printing and CNC milling are introduced. Then, the RepRap project is described and discussed. After that the types of robotic manipulator is explored and the one to be studied in this thesis is chosen. The chosen manipulator is further presented, exploring the relating research. The areas of CAD and CAM, G-code and path-optimization are then introduced. Finally, the genetic algorithm and the ant colony optimization are introduced.

2.1 Rapid Prototyping

Rapid prototyping is a way to create an inexpensive prototype directly and fast. Descriptions of the components are created on a computer and then directly manufactured.[6] Also, instead of creating the whole design, simplifications and/or miniatures of the final prototype can be created. This is often very useful in the designing stages of different engineering tasks.

Two of the methods used for rapid prototyping are 3D printing and milling with CNC milling machine. There are several different types of each, from the simplest hobby version to multi million commercial ones. A more advanced type of rapid prototyping is used to work with hard metals and to enhance the accuracy and quality. This technology is very expensive and outside the scope of this thesis. Other deposition based manufacturing methods than 3D printing (or fused filament fabrication) include Stereolithography, Fused Deposition Modeling (FDM), Laminated Object Manufacturing (LOM) and Selective Laser Sintering (SLS).[27] The different technologies used for depositing the material is outside the scope of this thesis, and have not been studied in detail.

2.2 3D printing

As mentioned, there are many different types of 3D printers. To keep inside the scope of the this text, mainly a simplified description of the 3D printing technology used at Bath University for their RepRap project will described (FFF - Fused Filament Fabrication). This technology uses an extruder that melts a material and then extrudes the material onto a board. The material

is often a type of polymer, for example ABS (the "Lego-brick polymer") or nylon.[6] The board and/or the extruder can move in the xyz-planes to build 3-dimensional objects. Every layer is held together because of the characteristics the material has when it is hot. It is called 3D printing because it is quite similar to the more regular 2D printing that is used for printing on paper.

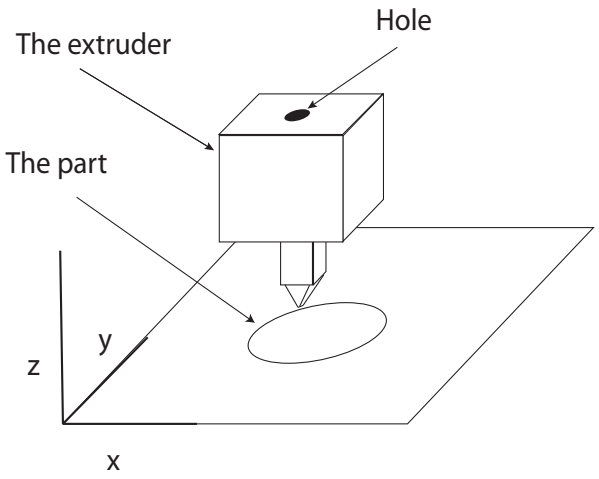


Figure 2.1: Illustration of a 3D printer

Figure 2.1 shows a simple illustration of what the extruder part and the board of the 3D printer could look like. The extruder is fed some kind of polymer through the hole on the top. There are different ways to do this. The polymer is then melted inside the extruder and fed out through the nozzle. The extruder and/or the board moves in the xyz-axes as shown on the figure. Not shown, are the rest of the manipulator, the mechanics around the manipulator (a fixture for example) and inside the extruder. These make it possible for the extruder and/or platform to move and the material to be fed through the extruder. The nozzle sits just above the last layer for each successive layer. The extruded polymer sticks to the last layer and solidify. In this way, the part is created layer by layer. Thinner layers equal the possibility of more details, but the mechanism that moves the extruder and/or the board has to be more accurate to do so.

To give a simplified explanation from start to end: First a 3D design of what is going to be printed is created in CAD-software (Computer Aided Design) and usually saved as an STL file. Then, programs that are designed for a specific 3D printer (for example Catalyst EX) process the file and send it to a microcontroller that is a part of the 3D printer. The microcontroller directly controls the motors and other mechanics of the 3D printer and the part is created according to the method described above. The reason for giving such a simplified explanation is that there is no standardized implementation of this procedure. Furthermore, there are many details and challenges in design and usage that is not discussed here.

2.3 CNC milling

CNC milling (computed numerically controlled milling [42]) is computer aided milling. It is traditionally less used for rapid prototyping than 3D printing and more as a manufacturing method. There are many variations of the CNC milling machine, but it is often a big box, where an object is put inside on a platform. A milling tool that is used to remove material from the object is located inside the machine. The tool spins a milling cutter that is somewhat similar to what is used in drills or dremel tools. The milling tool and/or the platform can typically move in the xyz-plane (similar to the 3D printer) in order to mill material away. Figure 2.2 is an illustration of the platform, part and milling tool. The platform and tool is attached to the machine by some mechanism in order to move. This is just a short overview of the CNC milling machines and there are many details and challenges in design and use that are not discussed here.

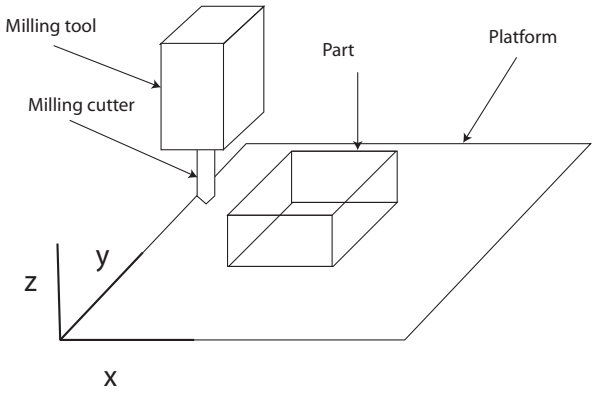


Figure 2.2: Illustration of a Milling Machine

Another type of milling machine is a robot arm that has the milling tool attached at its wrist. This technology has long been rated too inaccurate to be of any use in professional environments, but recent developments have given this type of milling a possible professional future.[2, 68] The robot arms usually has 6 degrees of freedom where the three last ones is called the wrist.[31] There are many different types of milling machines and all of them can not be covered here. The last on to be mentioned is the Tricepts by PKMtricept (PKM - Parallel Kinematics Machine)[50]. This is a combination of a parallel and serial manipulator and one of their models, the Tricept 9000, can be seen in figure 2.4.

The process from idea to finished part is quite similar to 3D printing. CAD-software is used to design the part and saved as a specific file type that suits the post processing software. This is typically some sort of CAM-software (Computer Aided Machining). The CAM-software creates a file that is either fed directly into the CNC milling machine or processed for a specific machine and then sent to the machine. There are many different implementations and no standard procedure.

Figure 2.3: A traditional commercial milling machine[49]

2.4 The RepRap project

As mentioned, this thesis is inspired by the RepRap, a project that started at Bath University by Adrian Bowyer. It consists of a self designed 3D printer that is designed in such a way that it is possible for the 3D printer to print many of its own parts. RepRap is short for replicating rapid prototyper. The project is based on the ideas of self-assembly and self-replication in biology. Living creatures produce themselves, given enough resources. The RepRap project is currently only studying self-replication. The interesting thing about self-replication is that it makes it possible for an entity to multiply exceptionally given the resources. This is unlike any other current manufacturing process, where a production growth in an exponential like manner is not possible over time. [6]

Behind the idea of a machine that can create all of its parts, there are some interesting characteristics. Since the machine can produce itself, only with the cost of the raw materials and assembly, typical rapid prototyping machines *can* become profitable for production and not only used for prototyping or other hobby related tasks. Also, the cost of the first machine is not as important as subsequent machines will (ideally) be quite cheap to produce.[6] This also makes it impossible for someone to sell the machine on a commercial basis. Another way to look at the RepRap is:

> ...a desktop manufacturing system that would enable the individual to manufacture many of the artifacts in everyday life [69]

2.4.1 The Present and Future of RepRap

The ideal goal for the whole project is to create what von Neuman describes as universal constructor.[51] Adrian Bowyer estimated that about 2500 RepRaps or RepRap deviates exists around the world (July 2009) compared to 4 at the

Figure 2.4: The Tricept 9000

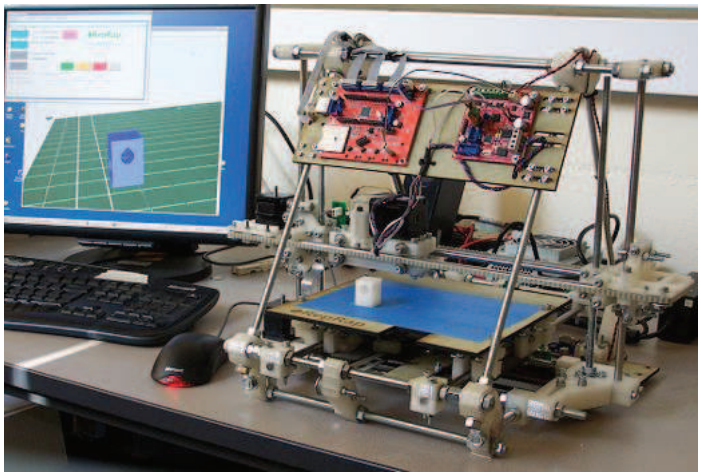


Figure 2.5: The latest generation RepRap, the Mendel[45]

start of 2008.[25] The use of open source design and code is central parts of the RepRap project. This allows for a large community of "RepRappers" that not only build the machine but also tries to improve it. The RepRap project has its own Wiki webpage where the RepRappers can describe their solutions or ideas on everything relating to the RepRap project. A vision for the project is to have hundreds of millions of RepRaps.[25] In other words, for many people to have a RepRap in their home to create different things they need or want.

One of the latest additions to the RepRap project is the possibility for electrical conductors to be directly built into the parts that the 3D printer creates. This makes the need for printed circuits marginal and thus the RepRap can create larger percentage of itself.[51] The RepRap is developing along two

Model	Mendel
Technology	FFF (Fused Filament Fabrication)/ Thermoplastic extrusion
Size	500 mm (W) x 400 mm (D) x 360 mm (H)
Weight	7.0 kg
Build Envelope	200 mm (W) x 200 mm (D) x 140 mm (H)
Materials	PLA, HDPE, ABS and more. Uses 3 mm filament
Speed	15.0 cm^3 per hour solid
Accuracy	Diameter of nozzle 0.5 mm, 2 mm min. feature size, 0.1 mm positioning accuracy, layer thickness 0.3 mm
Volume of printed parts to replicate	1110 cm^3

Table 2.1: Specifiactions for the Mendel RepRap [57]

lanes, Bath University and the independent RepRap community. As everything about the RepRap is made public, everyone (theoretically) can create their own RepRap machine at home and print their own things, but also help develop the machine.[44] This is somewhat related to artificial selection. A machine can make another machine that is of better design.[6]

As stated above, the RepRap still has to be assembled by hand, and it seems like there are no plans to make the machine able to assemble itself. Rather, the next (and current) steps in the project is designing a servo movement system for the machine and designing its material deposition (extruder) heads to for example be able to extrude solder.[51]

2.4.2 The Mendel RepRap

The Mendel is the latest of the two official RepRap designs created at Bath University (anno spring 2011). Figure 2.5 shows the Mendel manipulator. It is essentially made up of two serial manipulators, the one for the extruder and the one for the base. The one for the base moves the base back and forth along the y-axis. The serial manipulator for the extruder consists of two actuators. The first moves in the z-axis (up and down). The second holds the extruder, is moved by the first and moves in the x-axis (left and right). Specifications can be seen in table 2.1

2.4.3 RepRap Limitations

The RepRap is a very fascinating idea and also an affordable 3D printer that actually works. However it consists of many small parts and is quite tricky to put together. Thus a simplification of the design without sacrificing the amount it is capable to replicate would be a welcome step in the right direction. Some hobbyists have discussed whether a RepRap inspired design is capable of handling milling, something that would also be an interesting additional application for the machine. Both for creating PCBs and finishing printed parts.[53][52]

The machines that have been tested for this is often quite large compared to the RepRap Mendel and do not seem to embrace the idea of self-replication. An

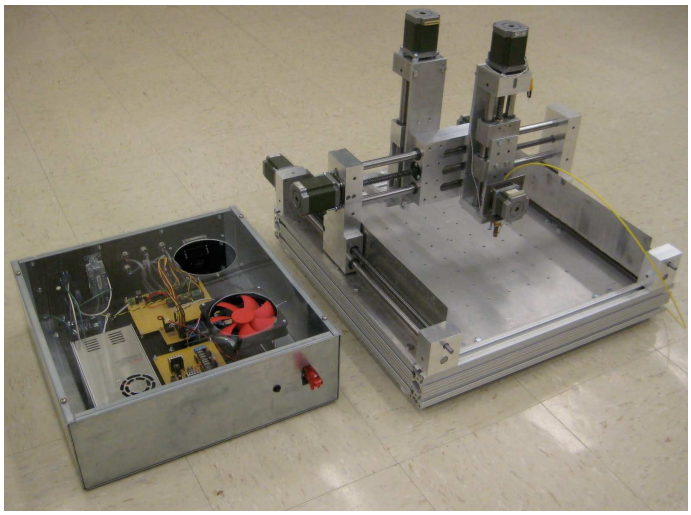


Figure 2.6: A pictute of the Hydra combined CNC milling machine and 3D printer [52]

example is shown in figure 2.6. Although there may be more limitations with the RepRap, the two mentioned issues are the ones that will be studied in this thesis.

2.5 Robotic Manipulator

One of the fundamental steps of this thesis is to decide on what kind of manipulator to study. What follows is a brief study of the different kinds of manipulators that are common. After that, arguments for the different kinds of manipulators to be used in this thesis are discussed, finishing with a choice of manipulator. There are several characteristics a manipulator can have. These include: [58, p.4-12]

- Configuration: complete specification of the location of every point on the manipulator
- Degrees of freedom (DOF): there are three for positioning (x, y and z coordinates) and three for orientation (pitch, roll and yaw). If a manipulator has six degrees of freedom it can reach a point with arbitrary orientation (albeit, often with some practical constraints). Manipulators can have more than six degrees of freedom.
- Workspace: the total volume swept out by the end-effector as the manipulator executes all possible motions. Some positions in the workspace reduce (serial manipulator) or increase (parallel manipulator) the degrees of freedom.
- Power source: hydraulically, pneumatically or electrically.
- Accuracy: the accuracy of the position and orientation of the end-effector, will affect the resulting parts the manipulator creates.

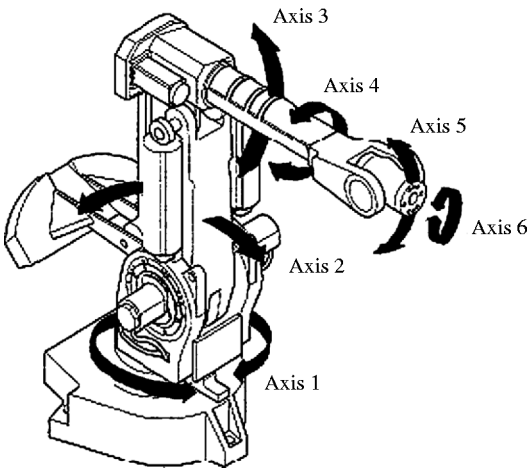


Figure 2.7: A serial manipulator with six revolute joints. There are many different possible designs for a parallel manipulator.[47]

- Repeatability: how well the manipulator is capable of doing the same task multiple times with the same accuracy
- Rigidity: withstand external forces. For example due to milling
- Joint type: revolute (R), prismatic (P), universal (U) or spherical (S). The first two can be actuated, the last two can not.
- Geometry: what type of joints the manipulator has and in what order. The geometry of the robot can be divided into two areas, the serial, kinematically open loop ones and the parallel, kinematically closed loop ones.

2.5.1 Serial Manipulator

The serial manipulator or robot arm is the most popular and well researched one. An illustration of a robot arm can be seen in figure 2.7. The joints connect links that starts at the base and ends at the end position. Typically, the manipulator has three joints and up to three joints are added as a wrist. They are categorized according to the first three joints. The most common ones are Articulate Manipulator (RRR), Spherical Manipulator (RRP), SCARA Manipulator (RRP), Cylindrical Manipulator (RPP) and the Cartesian Manipulator (PPP).[58, p 12-18] Their differences concerns (among other things) the size and form of the workspace, the rigidity and the practical design of the robot. The RepRap is a kind of Cartesian Manipulator, yielding a cubical workspace. Serial Manipulators can be compared to a human arm and has both its advantages and disadvantages.[14] Advantages are a sweeping workspace and dexterous maneuverability like the human arm. Disadvantages are limited load carrying capacity and precision positioning.[14]

2.5.2 Parallel Manipulator

Parallel manipulators are not as common and are usually treated in more advanced texts.[58, p 8] The kinematics and dynamics are more difficult to derive,

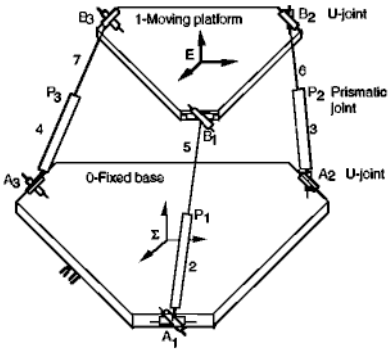


Figure 2.8: A 3 legged parallel manipulator. There are many different possible designs for a parallel manipulator [46]

but quite a lot of research has been done in since the late 1980s, especially on the Stewart platform.[14] Compared to the serial manipulator, the parallel manipulator exhibits a greater structural rigidity, which allows it to be more accurate.[58, p 19] On the other hand, the workspace size is often smaller than for serial manipulators.[12, p 243] One of the more common parallel manipulators is the Stewart platform. It was first publicized by D. Stewart in 1965 (however V. E. Gough built an operational version in 1954) and consists of two platforms that are connected with six legs.[14] One of the platforms acts as base and is fixed, while the other acts as end-effector and moves. The six legs are identical in their structure, consisting of 3 joints. The two joints connecting them to the platforms are spherical or universal, while the joint in the middle is a prismatic joint. Only the prismatic joint is actuated, the others are passive. The Stewart platform has six DOF. Removing three legs results in a tripod, this has 3 DOF and can not rotate the moving platform. An illustration of a tripod type of parallel manipulator is shown in figure 2.8. In the industry, some parallel manipulators have been developed for CNC milling, but these are not as simple as the Stewart platform alone. They sometimes have a serial wrist that makes it easier to access different areas of the milled part. Also, they are often of huge size and the base is at the top, attached to some kind of fixture. An example is shown in figure 2.4 at page 15.

2.5.3 Kinematics

The forward kinematics of a manipulator describes the position and orientation of the end-effector given the values of the actuated joint variables (angle or distance). This is performed mathematically with the use of matrixes and homogenous transformations. The opposite, inverse kinematics, is used to find the joint variables given the position and orientation of the end-effector.[58, p 73] For the serial manipulator, the forward kinematics has proven to be quite easy to derive, while the inverse kinematics is rather hard to derive. The opposite is true for the parallel manipulator.[12, p 243]

2.5.4 Dynamics

The dynamics is used to perform an in depth analysis of manipulator movement and includes a study of the forces and torques. For example, the friction in joints can be included in a dynamic model. The dynamics for parallel manipulators are very advanced, while for serial manipulators they are somewhat more simple.[14] However, dynamics are outside the scope of this thesis and will not be considered.

2.5.5 Discussion

One of the basic ideas behind this thesis is to improve the RepRap concept's capabilities to be alternatively equipped with a milling tool to perform milling and an extruder to print in 3D. To do this, the tool can be changed allowing the manipulator to both print and mill the same part. For this to be possible the manipulator must have some additional characteristics as opposed to be able to do only one of the things. An important issue is that milling requires much more structure rigidity than 3D printing.

To be able to perform 3D printing a manipulator should have 3 DOF. In order to mill the manipulator should have at least 3 DOF, preferably more. The workspace should be as large as possible. Electrical power source is preferable as pneumatic is too inaccurate and hydraulic require much maintenance, lot of peripheral equipment and is very noisy. Accuracy and repeatability is important both for 3D printing and milling, and rigidity is as mentioned especially important for milling.

Considering the greater structural rigidity of the parallel manipulator, it is more suitable for milling than a serial manipulator. On the other hand, the workspace is smaller, limiting the size of parts created. For both 3D printing and CNC milling the end-effector location and orientation is always known. This calls for the use of inverse kinematics that is simple for the Stewart platform.

An important aspect is that the robot should be an alternative to the RepRap. This means that the design limitations given by the RepRap concept should be paramount in the design process. The most important limitations are that the design can not be too advanced or too expensive as the robot is to be available to as many as possible. And that the robot should consist of as many as possible parts that it can create itself.

Taking these arguments into account and considering the interesting recent research in the field, the Stewart platform was deemed the most suitable platform for hobby oriented 3D printing and milling and thus chosen for this thesis.

2.6 The Stewart Platform

Bhaskar Dasgupta and T.S Mruthyunjaya have studied the research development of the Stewart platform per 1998.[14] They report that an increasing amount of research on the platform has been done in the 80s and 90s. The main focus of their article is the research areas and challenges of the Stewart platform, but by doing so they also give a characterization of the Stewart platform. Parallel manipulators like the Stewart platform are believed to have greater rigidity and positioning capability than serial manipulators. The kinematics (relation between length of legs and the position of the end-effector) is

opposite in difficulty to the serial manipulator. Inverse kinematics is simple (deciding the length of the legs given the position and orientation of the end-effector), while forward kinematics is complex and difficult. A similar duality is reported when singularities are discussed. The Stewart platform experience singularities as configurations where the machine gains one degree of freedom, but looses its controllability. The issue of singularities, which is tough to deal with, is not further investigated in this thesis. Another difficult domain in the Stewart platform research is analyzing and determining the workspace. The authors present some interesting research on this difficult topic. This thesis will however only suggest that the workspace of a parallel manipulator is smaller than the workspace of a serial manipulator of roughly the same size.

2.6.1 Design

Dasgupta and Mruthyunjaya present the generalized design of the Stewart platform as two platforms connected with six extensible legs. The legs are connected with spherical joints at both ends or spherical at one and universal at the other. The designs presented later in this thesis consist of universal joints at both ends. This is not entirely uncommon as the designs in these videos shows. [63] [62]

The shape of the platforms is quite arbitrary. The authors present among other designs, a design where both base and top are triangles where legs meet in pairs at the edges (3-3) and one where the base has six distinct connection points for the joints (6-3). The design presented later in this thesis use a 6-6 type of design, where in position zero, the two platforms are hexagons rotated 180 degrees in relation to each other.

2.6.2 Recent research

Much of the post 1998 research that was found dealt with the forward kinematics of the Stewart platform. The issue with this problem is that it is complex and time consuming to calculate. Several researchers have presented good solutions to the problem,[67, 41, 35, 23, 15] and some even suggesting forward kinematics applications to be used in real-time.[32, 29]

Ilian A. Bonev and Jeha Ryu have studied a new method to find a set of all attainable orientations of the platform about a fixed point.[5] Yunjiang Lou et al have studied the dynamic based trajectory planning for a Stewart platform.[36] Other studies of the dynamics has been performed as well. Denis Garagic and Krishnaswamy Srinivasan have studied friction compensation for the Stewart platform.[21] Shih-Ming Wang and Korner F. Ehmann et al have studied error and accuracy models and analysis of the Stewart platform.[66] Others have studied robots that are similar to the Stewart platform.[70, 24]

2.7 Research relating Rapid Prototyping, CNC milling and Biologically Inspired Computing

Some papers have discussed the use of Biologically Inspired Computing in relation to rapid prototyping, CNC milling or similar applications. Li Xueguang et al have established the Traveling Salesman Problem on the path-optimization problem and have applied the backtracking and genetic algorithm on the problem.[33]

Similarly, Ajay Joneja et al have in [27] studied tool path-optimization for the rapid prototyping process with a genetic algorithm. Pang King Wah et al have developed an enhanced genetic algorithm to solve the problem.[65] Z. Car et al have also used the genetic algorithm to optimize machining parameters in a turning process.[8] Similar research has been done for CNC rough machining by Agathocles A. Krimpenis et al.[30]

2.8 CAD and CAM

Mechanical CAD-software (Computer Aided Design) is a type of software where a 3D design of a physical object is created by the user. CAD-software also has many options to for example test and assemble the created objects. All or most of the characteristics of the object can be specified depending on the software.

Computer-aided manufacturing (CAM) is a set of techniques used in computer control for manufacturing.[13, p. 102] More concretely; it is the process of interpreting the design file from CAD software in order to create a file written in G-code or other similar control language. The file can be interpreted to control CNC milling machines and 3D printers. The G-code file sometimes needs to be post-processed in order to fit the specific manufacturing machine. With larger systems the post-processing can be integrated in the CAD and CAM software.

2.9 G-code

G-code is a very common language for CNC-programming, but it might also refer to a part of the CNC-programming, namely preparatory commands.[54, p47-48] In this thesis, G-code refer to the programming language and for example includes miscellaneous commands. The preparatory commands are used to prepare the control system to a certain state of operation. Following the preparatory commands (Gxx) are specific instructions for that type of command. An example is G01, which means Linear Interpolation and is followed by the position and orientation the end-effector is going to move to.

The miscellaneous functions (Mxx) is used to command the tool,[54, p 52-53] it might for example start a milling tool in the clockwise direction (M03) or stop the program (M00). Miscellaneous functions are divided into machine related functions and program related functions. Machine related functions controls various physical operations of the CNC machine while the program related functions control the execution of a CNC program (such as calling and ending a subprogram). For preparatory and miscellaneous commands implemented in this thesis see chapter 5.3, table 5.1.

2.10 Path Optimization

In this thesis, the path-optimization problem has been defined as the Travelling Salesman Problem (TSP) as described in further detail in chapter **??**. The parts of the path to be optimized are the ones where the tool is inactive. The TSP is the problem of finding the fastest round trip between n cities, where each city is visited only once.[3, p. 100-101 and103] In more formal terms, the goal is to find a Hamiltonian tour of minimal length on a fully connected graph.[18] There

$$\begin{pmatrix} 0 & 32 & 15 & 5 \\ 32 & 0 & 63 & 21 \\ 15 & 63 & 0 & 1 \\ 5 & 21 & 1 & 0 \end{pmatrix}$$

Figure 2.9: A cost matrix of a undirected complete graph with four nodes (cities).

are (n-1)! possible tours, therefore using the brute force algorithm for a graph with more than 8-9 cities is infeasible.[3]

The TSP is a hard or NP-hard problem depending on how the problem is presented. Currently, there exists no polynomial algorithm that solves the TSP and it is unlikely that such an algorithm ever will be found. The algorithms that solves the TSP are super polynomial (grows faster than any polynomial).[3] To solve the TSP, a cost matrix is used that presents the distances between all the cities. The algorithm searches this cost matrix to find the best solution. An example of a cost matrix is given in figure 2.9.

A thorough investigation of the research done in tool path-optimization with traditional algorithms has not been done for this research. Limited investigation suggests that the research has focused on making the CNC milling create better surface results on the parts that is milled. Two examples of this kind of research can be found in [39] and [4]. An article on optimizing the tool path length for turning CNC milling can be found in [65]. For studies of the problem done with biologically inspired algorithms, see chapter 2.7.

2.11 Genetic algorithm introduction

The genetic algorithm is one of the first types of evolutionary algorithms and was introduced by Holland in 1975.[19, p 2] It is inspired by the evolution of living organisms as seen in nature and by Charles Darwin's theory of evolution. The two cornerstones of evolutionary computing are competition based progress (the survival of the fittest) and combination of genes during reproduction. The combination of genes can include mutation.[19, p 2-4] The advantage of emulating this behavior on a computer is that the form and size of individuals is decided by the programmer. Likewise is the way genes combine during reconstruction, what individuals survive for another generation and so on. Evolutionary computing has proved to be suitable to solve several problems. Examples of real world applications are the timetabling of universities and design of a satellite dish holder boom. The boom proved to be 20 000% better than the traditional shape.[19, p 10]

2.11.1 How it works - general

Evolutionary algorithms are generate and test algorithms. A population of individuals with certain values for their genes is generated and their fitness is decided. From this population new individuals are created through recombination, mutation, both or simply survival of an individual from the old population.

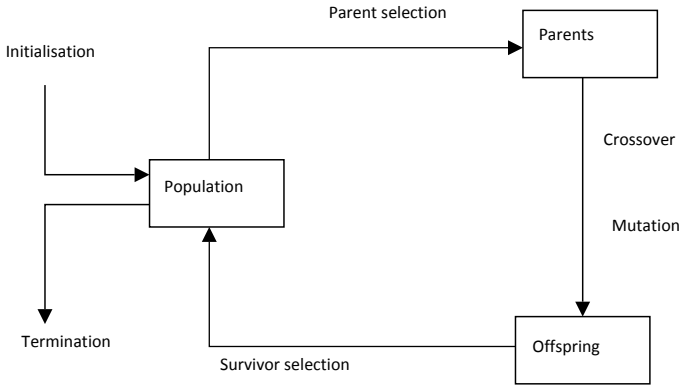


Figure 2.10: Flowchart of general evolutionary algorithm. This is heavily inspired by the flowchart at page 17 in [19]

Which of these that happens is randomly chosen according to some probability parameters. A new generation is chosen among the offspring and the current generation. Thus a cycle of generations is performed. Nine steps can be identified as seen in figure 2.10. The steps are now listed and described [19, p 16-24]

1. Initialization: The initial individuals of a population are often created randomly. This is because it saves computing power as evolutionary algorithms most often quickly moves from bad solutions to quite good solutions.[19, p 30] The first individuals are evaluated according to a fitness function and their fitness decided.

2. Population: Holds the individuals that live in the current generation. The number of individuals in a population has to be decided.

3. Parent selection: The parents are selected according to a parent selection mechanism with stochastic elements. The stochastic element is there to, among other things, avoid the algorithm getting stuck in a local optima.

4. Parents: The parents are the individuals that are chosen to have their genes transferred to the next generation. An individual in a population may be represented several times as a parent. Depending on a pseudo random mechanism, every parent is chosen to recombine with one (or more) of the other parents, with a possible mutation of the resulting offspring. Another option is that the original parent is mutated. A final alternative is that the parent survives for the next generation.

5. Recombination: Usually two parents split their genes and form new individuals called offspring. The exact method of crossover depends on the representation of the individuals. The recombination can also have stochastic elements that decide how many genes come from each parent. One of the interesting freedoms with evolutionary algorithms is that there is no restriction to the number of parents.

6. Mutation: Mutation is the random alteration of genes in a single individual. As with crossover this depends on the representation of the individuals.

7. Offspring: These are the indivdiuals that are created from the parents. The number of offspring has to be decided. Also, the offspring must have their fitness evaluated.

8. Survivor selection: Among the current generation and the offspring, who are the ones to survive for the next generation? This can be done in different ways, among which are: to only choose the offspring, the offspring and the best individual in the current generation or simply the fittest individuals.

9. Termination: The termination criteria decide when the algorithm is finished. This can for example be after a certain fitness has been reached. To be sure the algorithm terminates it is often smart to have a maximum number of generations as a safety.

As can be seen there are several steps where randomness plays a role. This often makes it hard to analyze the algorithm and the results it creates without experimental testing. However, the randomness is one of the strengths of the evolutionary algorithms as it makes it possible to avoid local optima.

2.12 Introduction to the Ant Colony Optimiazation algorithm

The background information presented here relies on [18]. Ant Colony Optimization (ACO) is a rather new form of optimization algorithm, being introduced in the early nineties. It models the foraging behavior of some ant species, in particular the pheromone deposition. ACOs have been successfully implemented on several types of optimizations problems, and specifically the Travelling Salesman Problem.

The ants of some species deposit a substance called pheromone as they move from the nest to a food source. Since the pheromone evaporates, places where the ants use more gets more and more popular. As ants are almost blind they orientate themselves with the help of the concentration of pheromone deposits.[18] This was proven by Deneubourgh and Goss with the double bridge experiment. [16, 22] However, in ACOs the ants are artificial; allowing the developers to give them features actual ants lack.

Chapter 3

Tools

In this chapter the tools used in this thesis is presented. Also, the type of motors and electronics that is suitable for the robot is suggested.

3.1 Python

Python is a programming language that is free, portable, powerful and remarkably easy and fun to use according to Mark Lutz.[37] He also stresses the language's software quality, high developer productivity and program portability.[37, p 3-4] Another important aspect is the 3rd party libraries, two of which are the free Numpy and Matplotlib. The 3rd party libraries make Python a very flexible language and covers topics from web programming to advanced mathematics.

3.1.1 Numpy and Matplotlib

Together with Scipy these libraries almost rivals Matlab in mathematical computation. More importantly, Numpy and Matplotlib make drawing in 3D possible. This makes it possible to create a 3D simulation of the Stewart platform.

3.2 Solid Works and Solid CAM

In this thesis SolidWorks has been used as CAD software and SolidCAM has been used as CAM software.[56, 55] SolidWorks has been extensively used to design and test the design of the Stewart platform while SolidCAM has only been used to verify that G-code can have a varied syntax.

There exist many different CAD and CAM software producers. [34] SolidWorks and SolidCAM was chosen simply because the University of Oslo has licenses for the software, I had earlier experience with it and a free student edition is available.

3.3 Stepper motors

The motor that is suggested to be used for further study of this project is a two phase hybrid stepper motor as can be seen in figure 3.1. It has high reliability,

Figure 3.1: A stepper motor

low cost and is easy to control.[59] The motor consists of a stator made of soft iron equipped with windings/coils and a permanent magnet rotor. The rotor has two sets of teeth that are out of alignment with each other by a tooth width. The number of teeth decides the accuracy of the motor. A driver is needed for the stepper motor between the microcontroller and the motor in order to use it.

In a two phase stepper motor, there are 4 windings. Two and two windings are positioned opposite to each other, and each pair is positioned 90 degrees to each other. The two opposite windings is applied a voltage at the same time so that a rotor tooth is magnetically attracted to each of these. After the rotor has moved the other two windings are applied a voltage to attract the teeth closest to them. This continues as one pair is applied a voltage while the other pair of windings is applied zero voltage.[9, p. 632-637] There are 3 typical step modes, full step, half step and microstep. When both windings are always on with alternating opposing currents, full step mode is used. Half step mode is when the motor alternates between energizing two windings and one winding. Half step gives a higher resolution for the motor. Microstepping allows for an even higher resolution by controlling the current in the motor windings. The resolution is limited by the mechanics of the motor[59]

The stepper motor usually has no feedback as the mechanical construction means that when the motor turns x steps it is possible to determine the position by counting the steps. However, steps might be skipped if the motor is under heavy load. This is very unlikely to be a problem in the current setting. The stepper motor is regarded as being accurate enough for the RepRap.

3.3.1 Alternatives

The main alternative for the stepper motor is the DC servomotor. As with stepper motors, the servomotor varies in size, complexity and price. The DC servomotor consists of a closed loop where a tachometer or other device provides feedback for the position of the motor. The inner workings of a DC motor are not explained here, but a thorough investigation of the DC motor can be found in [9].

A short comparison made by W. Voss [64, p 70-71] shows that the stepper

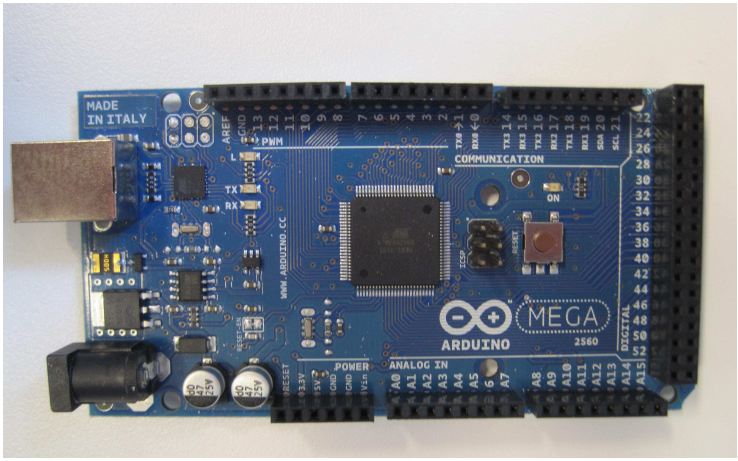


Figure 3.2: The Arduino Mega

motor is a better choice for low speed applications with higher torque. As the speed increases, the stepper motor looses its torque and at one point a servo motor should be used instead. A deeper analysis of which motor to use for the Stewart platform has not been done in this thesis. It is believed that both can be used, but the servomotor needs a more advanced control system (for example a PID regulator), therefore the stepper motor is suggested to be the preferred choice.

3.4 Arduino

Another useful tool that can be used in the future is the Arduino electronics prototyping platform.[1] It is open-source, has a large community and is continuously improved with new *shields* and other electronics that can be connected to it. The Arduino Mega has the possibility of controlling all the six motors with only one board. The Arduino Mega can be seen in figure 3.2

Chapter 4

Design

One of the most interesting aspects of this project is the physical implementation of the robot as this is where the research proves its usability. If the robot can't be built with the desired restrictions (see chapter 2.4 and 2.5 and especially 2.5.5), then there is little use in studying the rest of the system.

As stated in the introduction, one of the goals for this thesis is to make a suggestion for a different design than the current RepRap design that preferably would simplify the mechanical construction and allow a more diverse use by including CNC milling. The Stewart platform has been chosen because of its rigidity, however the actuators can be advanced and expensive or inaccurate. One of the reasons why the design is presented in such a detail is to keep with the idea of RepRap and make the design open-source. The designs and the ideas behind them will be discussed. After that, a comparison with the RepRap is done. Finally a concluding discussion sums up the experiences gathered from designing a Stewart platform.

Figure 4.1 shows a simple illustration of how the system is planned to function. A milling tool or an extruder is attached to the moving platform. The tool will work inside the Stewart platform, as shown on the figure. An interesting aspect is to first use an extruder to extrude a part in plastic, and then use a milling tool to create a smooth finish and to create cavities. Milling tool and extruder technologies have not been studied in this thesis.

4.1 Actuator

The idea behind the actuator was presented to me by Mats Høvin, an associate professor at the Robin research group at the University of Oslo. The actuator consist of a motor, in this case a stepper motor, a threaded rod and a nut with a nut housing as can be seen in figures 4.2 and 4.3. The nut is screwed on the threaded rod and the rod is connected to the motor shaft with the help of a plastic holder (and glue or epoxy). The motor is in a housing that is connected with a universal joint to the base-platform of the Stewart platform. Similarly, the nut is in a housing that denies it to spin freely. Because the housing is connected to the top-platform with a universal joint, the nut and the nut housing will not spin with the threaded rod when the motor is actuated. Thusly, the threaded rod will force the nut housing up and down and make the

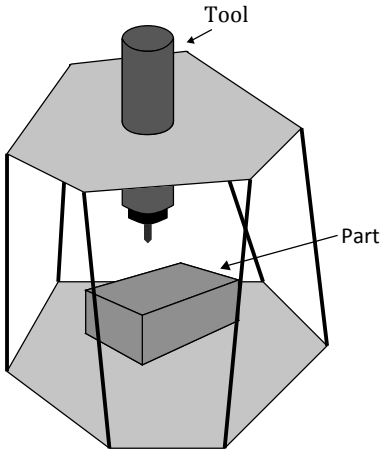


Figure 4.1: An illustration of the rapid prototyping system. A tool is attatched to the top platform and moved in order to mill or print. In this example a milling tool is attatched to the moving platform. A mechanism that can be used to keep the tool is not shown (and have not been studied in this thesis)

leg move in a prismatic manner.

Stepper motors come in many different sizes and qualities. A typical affordable stepper motor [60] has a step-size of 1.8 degress. This equals 200 steps for one revolution. A typical threaded rod will have a thread pitch of 1 mm. The result of this will be that one revolution will cause the actuator to move 1 mm. As one step on the stepper motor is 0.005 of one revolution the very high accuracy of 0.005 mm can be obtained. The referenced stepper motor has an accuracy of +-5% (0.00025 mm) and a RPM at 5 V of about 60. Hobbyists easily made the motor run at approximately 180 RPM and have even made it run as fast as 600 RPM. [60] This is done by exceeding the recommended maximum voltage, something that is not that dangerous using a current limiter (usually in the motor driver). Taking this into consideration, a safe low would be around 120 RPM, which yields only 2 mm/sec. The problem with slow speed can be overcome with the use of a higher thread pitch on the threaded rod. This will result in less accuracy, but since the accuracy of 1 mm pitch size is so high, this does not seem like a problem. 4 mm thread pitch will result in 8 mm/sec and an accuracy of 0.02 mm. Another possibility is to get a faster motor and thus sacrificing affordability or take the risk of running the motor at a higher voltage than prescribed. An issue with the accuracy that has to be taken into consideration is that there are six legs. Will they increase the error or mitigate it? More important is inaccuracies created in the nut and the universal joints. This has not been given focus in this thesis, see [66] for a discussion of errors caused by joints.

The conclusion on the actuator is that it is in fact very accurate, affordable and, in the current setting, slow. There is a tradeoff between the three that can be adjusted with the choice of design. Since it has not been possible to test the actuator a definite suggestion on the parameters can not be given. However, strong indications show that by using a motor that is a bit faster than the referenced one (for example 360 RPM) and using a thread pitch of 4 mm, the

Figure 4.2: A CAD model of the actuator

actuator will have a top speed of 2.4 cm/sec and an accuracy of 0.02 mm.

4.2 CAD Design of the Stewart platform

The design of the Stewart platform that is discussed here is the physical appearance of the different parts. It has been largely based on the actuator mechanism. Three types of design are proposed, one that was created as an initial design, one that was designed for a prototype print on a commercial 3D printer and three experimental type designs.

4.3 First design

The first design mainly deals with how the different housings (motor and nut) and the universal joints is going to look like. The arms consist in many ways of two parts, the bottom part that is connected to the base platform and holds the motor, and the top part that is connected to the top platform and hold the nut. The two parts are connected by the threaded rod. The threaded rod is connected to the motor with a simple mechanism that use glue or epoxy to hold them together.

Another factor is the idea that this design should be able to print and/or mill the part itself (can't be too complex). This is rather hard to decide before a working prototype has been created and tested. This area of study also relies

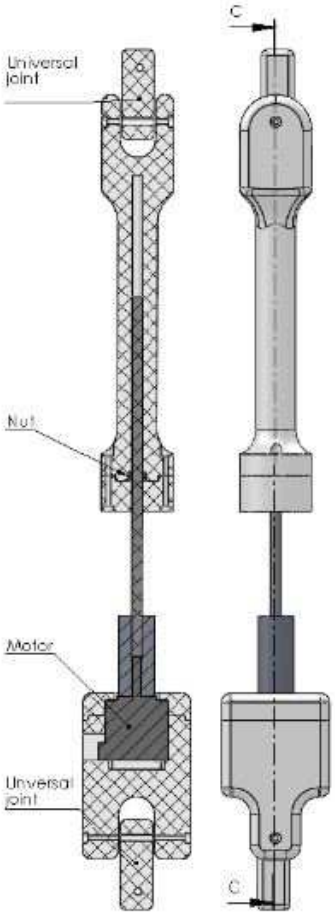


Figure 4.3: A CAD model of the actuator, whole and cut in half

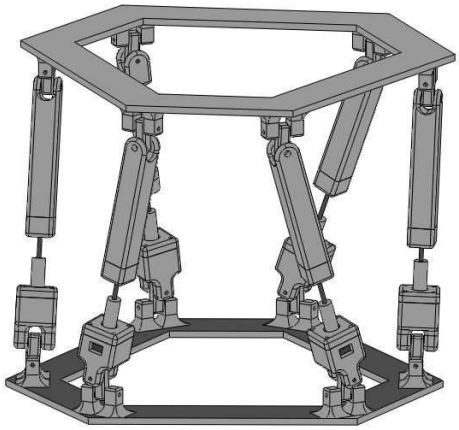


Figure 4.4: CAD model of the first design of the entire Stewart platform

much on practical experience in 3D printing and CNC milling, but is somewhat further investigated in chapter 4.4. The part has to be durable and have a long lifetime (an exact amount of hours is not specified at this point, but the idea is that a part should be thicker rather than thinner). At this stage in the development of the design the amount of material needed (cost) has not been given much focus. The entire first design of the Stewart platform can be seen in figure 4.4.

4.3.1 Motor housing

The motor is kept in place by a housing that completely surrounds the motor except the top part that is covered by a lid with a hole for the motor shaft. The lid also has holes for screws. A hole in the housing has been made for the motor wires. The design of the housing can be seen in figure 4.5. The housing is also part of the universal joint at one end. The universal joint is described in more detail later.

The presented design of the motor housing is not very clever, it needs a lot of material and has no lasting solution for taking the lid on and off easily. If the motor breaks down, the whole housing has to be replaced. A leaner design that also includes a lid that is fastened with bolts (instead of screws) is quite possible but is rather hard to implement without making the parts to complex.

4.3.2 Nut housing

To keep the nut in place a suitable room has to be made for it in the top arm. Also, a lid has to hold the nut in place. The lid and housing can be kept together with bolts. The length of the housing is paramount in the decision on how large the workspace is going to be. This is because as the threaded rod goes into the housing (the actuator moves downward) the housing in fact needs to be able to house the threaded rod. If not, the threaded rod will only crash with the top universal joint. This causes the nut housing to be quite large and thus use a large amount of material.

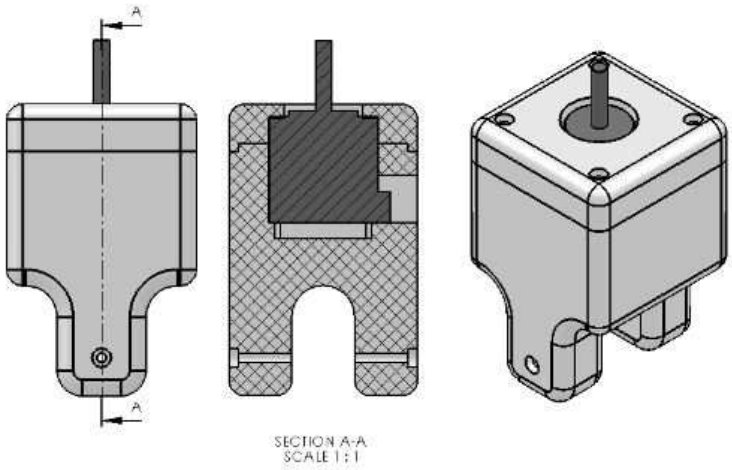


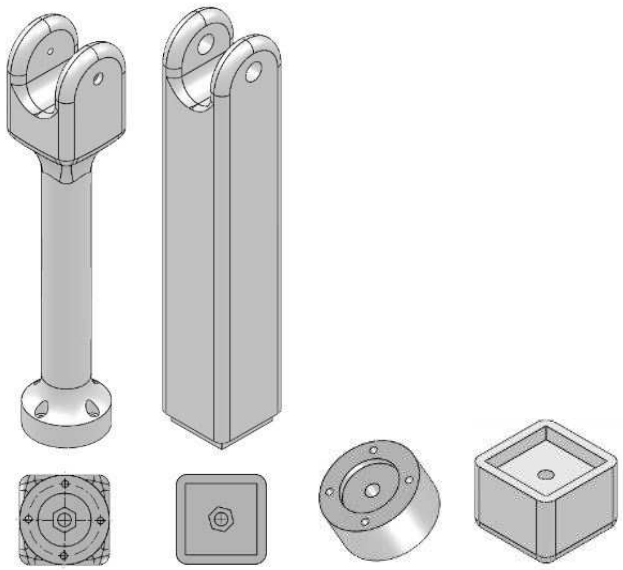
Figure 4.5: CAD model of the motorhousing

Two designs of the nut housing can be seen in figures 4.6a and 4.6b. One has a square shape and would probably be easier to print, but use more material than the circular shaped alternative. As can be seen, both have a connection point to the universal joint. A problem with the square alternative is that it does not have the possibility to use nuts and bolts with the lid in order to make the nut inside the nut housing easily replaceable. Also, as can be seen on the figure of the circular nut-housing, the connection mechanism with the universal joint might be a bit fragile.

An evident limitation with this design is the possible usable length of the threaded rods. This limits the workspace both in size and possible tilt and rotation. An example is given in figure 4.7. Of course the nut housing can be made much larger to allow a larger workspace, but this will come at the cost of much more material used. For alternatives for the nut housing see experimental design in chapter 4.5.

4.3.3 The universal joints

The universal joints are able to give the two arms (or sides of the universal joint) an almost arbitrary orientation towards each other. There are several ways to implement a universal joint. The presented here is similar to the one in figure 4.8. The idea of a universal joint is that a middle piece is connected to both parts by an offset of 90 degrees. Both parts have a rotational connection with the middle part. Figure 4.9a shows how the nut-hosing is connected to the top platform. Notice the middle piece. This probably will suffer a lot of stress when the platform moves and a more durable design might be needed. This could for example be metal reinforcements glued or otherwise fastened inside the holes. It also has an offset. This might cause errors with the inverse kinematics (see chapter 5.2) as this has not been accounted for there. Alternatives as universal joints can be seen in figure 4.10. Further analysis is needed before it is decided what sort of universal joint is preferable. Another type of joint that acts in a



(a) The nut housings (b) The lids

Figure 4.6: The nut housing (top part of the arm) for the Stewart platform, first designs

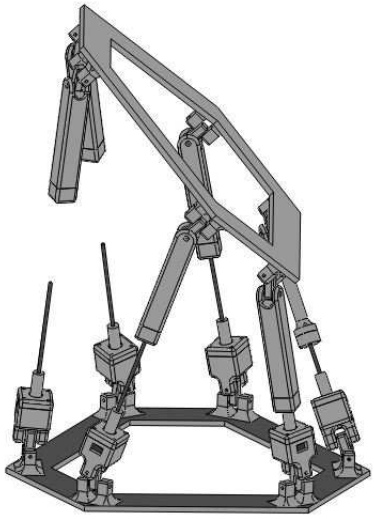
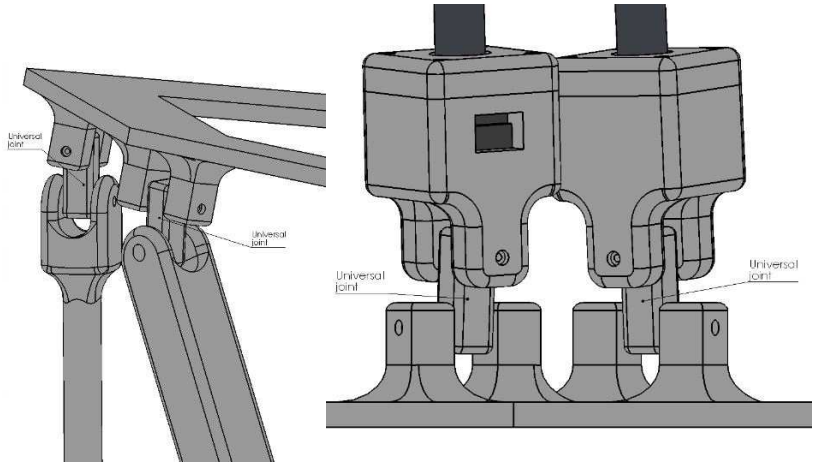


Figure 4.7: Limitation of tilt in the Stewart platform

Figure 4.8: A picture of a universal joint [48]



(a) The top universal joint (b) The base universal joint

Figure 4.9: The universal joints

similar manner is the spherical joint. This consist of a ball like part for the arm end and a container that house the ball.

A figure of the joint between the motor-housing and the base platform is shown in figure 4.9b. This joint is very similar to the top one. It is possible to see that there exists a trade off between the amount of material used (cost) and the lifetime of a given part. Stress tests can be performed in order to determine the best size and modifications of the universal joints. Unfortunately material engineering has not been included in the scope of this thesis.

4.3.4 The platform

The design of the platform is shown in figure 4.11. The size of the platform can be changed. However, the size of the motor, motor-housing, threaded rod and nut-housing have to be considered and possibly altered as well. At this stage in the development of the design, both the top and base platform has the exact same design. When an extruder or milling tool is to be attached to the top platform a different kind of top platform that can hold the tool must be created.

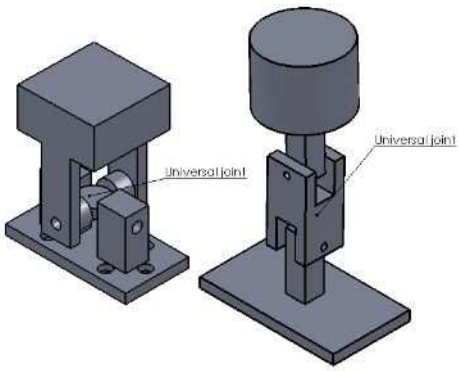


Figure 4.10: Alternatives for the universal joint

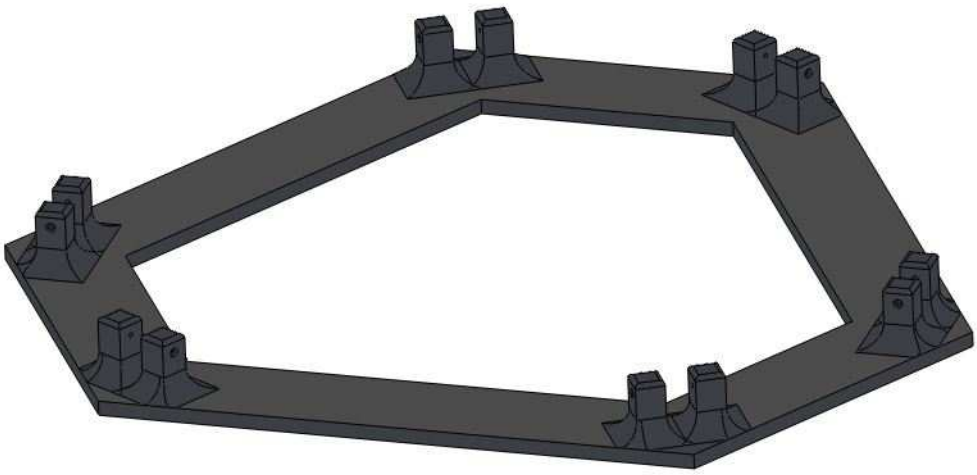


Figure 4.11: A CAD model of the platform

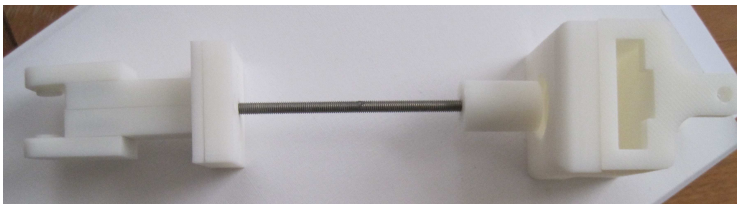


Figure 4.12: The prototype of an arm printed on a commercial 3D printer

Also in the platforms are parts of the universal joints. The size of these is quite arbitrary as the platforms have a lot of space to hold them. As discussed earlier, stress tests can reveal the ideal size and design of these parts.

4.4 Prototype design

The prototype design is a design for the arms that is based on the initial design and modified to be made up of as little material as possible and actually be printable on a commercial 3D printer (the Dimension SST 768 [10]). Since one of the central aspects of this thesis is self-replication, a printable design is of utmost importance. The design could furthermore be used for initial testing of platform control. One of the hardest issues in order to accomplish this is to design the parts in such a way that little or no support material is needed by the commercial 3D printer. Another problem is the size and location of holes and how the mechanical accuracy of the 3D printer works. This is due to the fact that if the hole is positioned too close to and edge, the area in between will not be properly printed. When the parts get small and have several holes in them for connecting with other parts, this must be given extra consideration or the part may not print right. A picture of the printed parts of one of the arms is shown in figures 4.12 and 4.13.

The main part of the motor housing was divided in half and all the walls have had their thickness reduced. The lid remains essentially the same. The three parts can be glued together with the engine inside to form a single part. Care has been taken to allow the parts to be printed with little or no support and that the lips that connect the three parts together are wide enough to be glued together and still fit neatly. The three parts of the motor housing can be seen in figure 4.14a. The nut housing has been made quite useless be reducing the amount of threaded rod it can house. This is to save material, as the current prototype is only for examining how the parts will print and possibly future (initial) testing. The top part has been divided into 4 parts to avoid unwanted support material and make the printing possible. This gives a total of 5 parts as can be seen in figure 4.15. Similar alteration has been done with the platform part of the universal joints as can be seen in figure 4.14b. Potential prototype platforms are wooden boards that can be cut to size.

Figure 4.16 shows layer detail of a single layer in the 3D printing software (Catalyst EX) for the 3D printer. Notice around the hole that it is not completely covered. Some space might be accepted, but if it becomes too large, the part can become unusable. A good conclusion to the challenges of creating a printable design is that it seems much easier than it actually is and a lot of

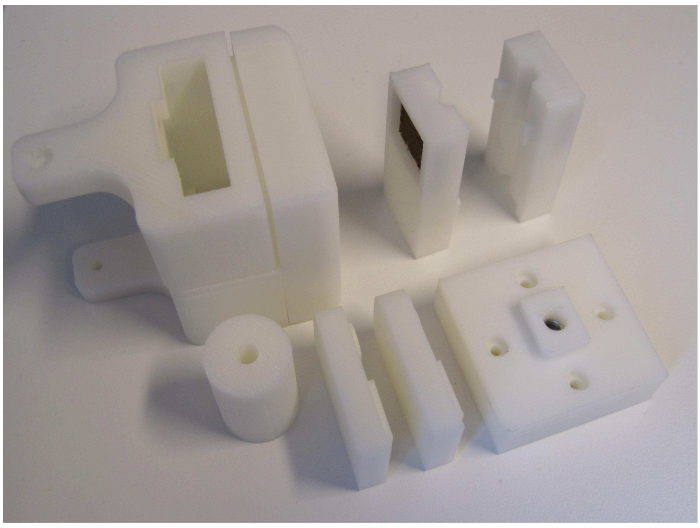


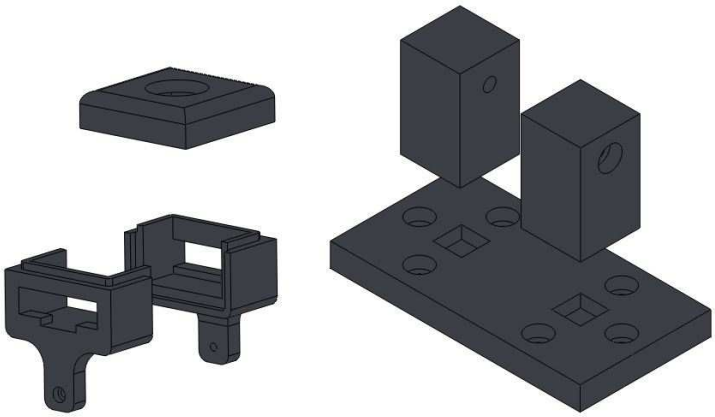
Figure 4.13: The parts of the prototype of an arm printed on a commercial 3D printer

issues have to be attended to at the same time. Especially if there are large holes involved. The possibility of a finish milling can really give some relief in the design challenges, especially considering holes.

4.5 Experimental design

The problem with the large nut-housing has been tried to be mitigated by designing the top universal joint in a different way. The idea is to allow for a small nut-housing and a larger workspace (longer threaded rods) at the same time. The trade off, however, might be that the rod can in some configurations crash with the universal joint, the top platform or a different threaded rod. Other issues are to model the offsets created by the design and controlling the manipulator. Three experimental designs are presented here, together with their shortcomings and advantages. The designs have been through simple testing as SolidWorks assemblies, where movements, size of workspace and collisions have been looked at. This testing is not thorough and has been done in order to remove the least possible designs. The design presented last seems to be the most promising one. The designs are presented with 400 mm long threaded rods in comparison to the 200 mm long threaded rods used in the initial design.

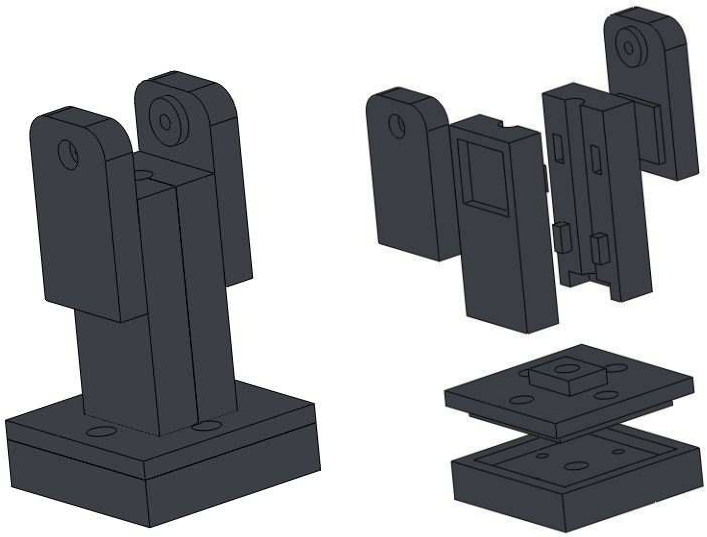
Version 1 can be seen in figure 4.17. A serious problem with this design is that at one side the universal joint is parallel with the threaded rod. This will not work very well when the motor spins the threaded rod, and can not be used. Version 2 can be seen in figure 4.18. The problem with version one is removed as both rotational parts are 90 degrees to the threaded rod rotational axis. As can be seen at the top universal joint, both the center part and the connection point to the arm both have large offsets. This makes the top joints of the manipulator behave in a quite unpredictable way, and a different type of inverse kinematic analysis has to be made. When tested as assemblies in SolidWorks both of these



(a) Exploded view of the motor-housing

(b) Exploded view of the platform connector

Figure 4.14: Exploded view of the motorhousing and of the platform connector for the prototype design



(a) Normal view

(b) Exploded view

Figure 4.15: CAD models of the nut housing for the prototype version

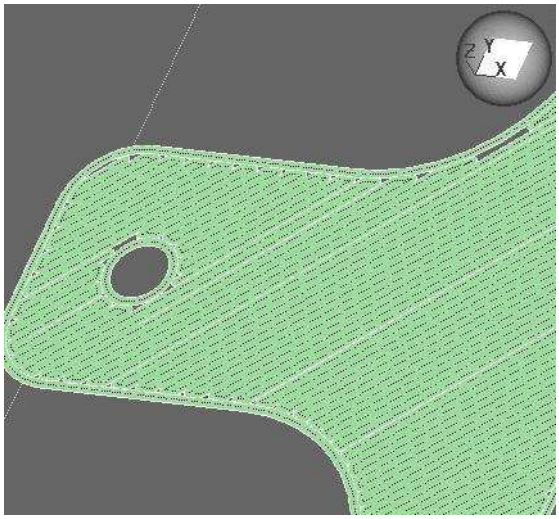


Figure 4.16: A layer of deposited material in the professional 3D printing software, Catalyst EX.

designs proved to collide often and were very uncontrollable.

The most promising experimental design is Version 3. Figures 4.19 and 4.20 show how this design is very similar to the original with only an offset in the nut housing. A simple study of the movements as a SolidWorks assembly reveals that this design has stable movements and should be investigated further.

4.5.1 Discussion on version 3

In theory, the experimental design allow for an infinite size of the workspace as the threaded rods can be of infinite length. However, there are many practical issues that have to be dealt with. The longer threaded rods make collisions between them quite possible, especially when the top platform is in a low position. This can be seen in figure 4.21 where 1000 mm long threaded rods are used. On the other hand, longer threaded rods and an offset in the top joints allow for a larger workspace, not only in the vertical direction, but also in the horizontal direction. Also, larger tilt angles are possible as shown in figure 4.22. Another issue is that too long threaded rods might cause them to flex, causing inaccuracies. What is needed to make version 3 work is to make the necessary adjustments to the inverse kinematics (see chapter 5.2 for typical inverse kinematics for the Stewart platform), find an ideal length of the threaded rods and implement a collision control system that includes the part of the threaded rods that is above the top platform. This is possible and would allow a great advantage in the Stewart platform design.

4.6 Comparison with the RepRap

Although a finished and working design is not practically implemented or tested in this thesis, the design has reached a state that allows it to be compared to the RepRap. The Mendel RepRap was presented in chapter 2.4.

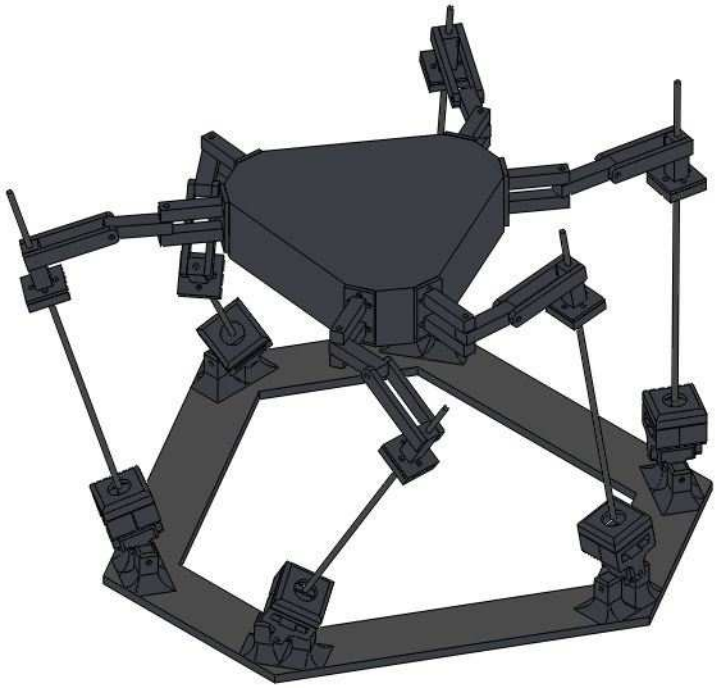


Figure 4.17: Experimental design, version 1

Figure 4.18: Experimental design, version 2

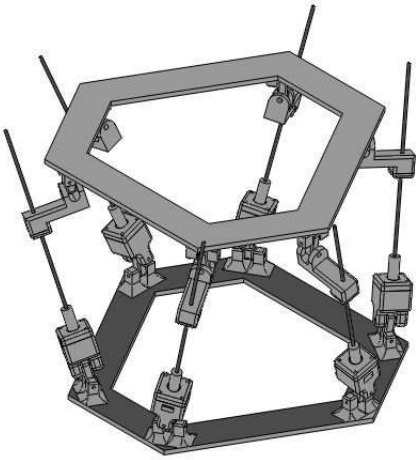


Figure 4.19: Experimental design, version 3

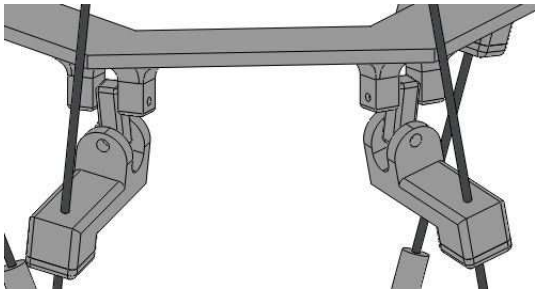


Figure 4.20: Close up of version 3

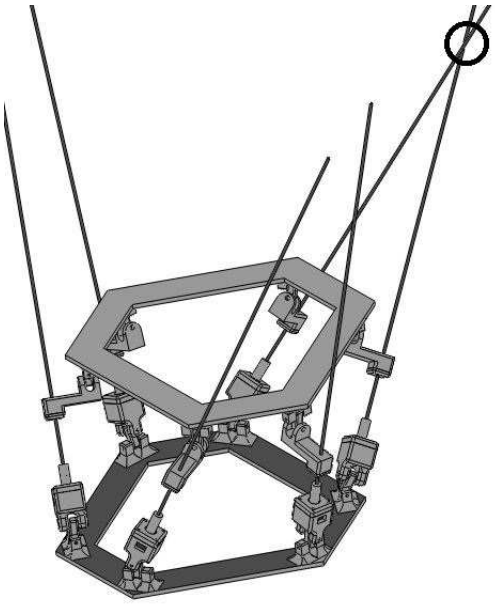


Figure 4.21: Crash in threaded rods for experimental design 3. 1000 mm threaded rods.

The Mendel RepRap consist of many parts, as figure 2.5 shows. As seen in the discussion about prototype design, the Stewart platform is also likely to need a rather large number of parts. However, this number might very well be substantially smaller than for the Mendel. The parts are also designed to be easy to assemble. The Mendel has several intricate parts that are not so easy to assemble. To what degree the Stewart platform would be easier to assemble is hard to tell before a working version has been created.

The mechanical design of the Mendel RepRap allows it to obtain a higher velocity than the presented Stewart platform. The manual for the Mendel reports a 3000 mm/min velocity [40] and a printing speed of 15 cm**3 per hour. [57] This is faster than the suggested 24 mm/sec (1440 mm/min) discussed above for the Stewart platform. Slower speed equals longer production time. However, max speed is not used when printing or milling, which means that the speed issue with the Stewart platform might not pose a serious problem.

The wiki-webpage for the Mendel also presents position accuracy for the Mendel of 0.1 mm, a nozzle diameter of 0.5 mm and a 2 mm minimum feature size.[57] With 24 mm/sec the Stewart platform has potentially 0.02 mm position accuracy. This is potential because of possible inaccuracies in joints and nuts that have not been investigated. It is questionable whether the 0.02 mm accuracy is needed for 3D printing and testing will reveal how accurate the Stewart platform really is. A better accuracy might be more important for CNC Milling. As mentioned in chapter 2.4.3 the RepRaps lack the stiffness to perform milling.

For a discussion on Stewart platform workspace see.[14] This might be increased when the experimental design is used. Anyway, the workspace for the Stewart platform is obviously smaller relative to its size than the Mendel. On

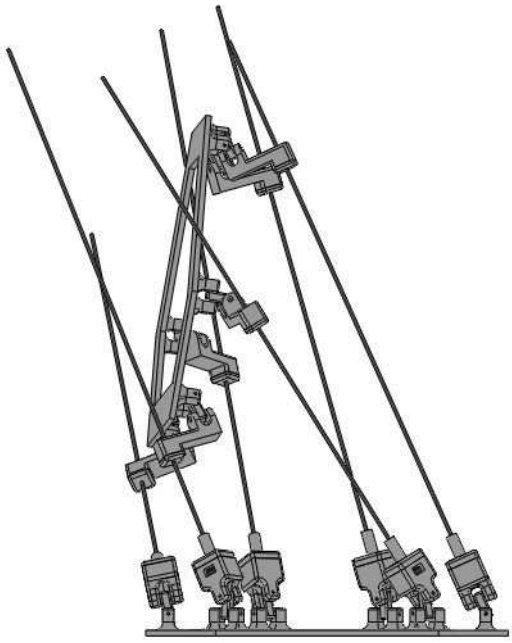


Figure 4.22: No crash with experimental design 3. Showing a large tilt in the moving platform. 1000 mm threaded rods.

the other hand, the Mendel has only 3 DOF while the Stewart platform has 6. Even though the orientation of the tool has clear limitations, this will allow the milling tool to perform a wider range of operations.

According to Sells, who has worked on the Mendel at Bath University, the Mendel can print between 48 % and 67 % of its own parts (excluding fasteners).[43] To calculate this number for the Stewart platform is impossible at this stage in the development, but comparing the designs it is easy to see that the Mendel has more nuts and threaded rods than the Stewart platform. On the other hand, the Stewart platform has 6 motors, while the Mendel has 4. Put together, this suggests a larger printable percentage for the Stewart platform. This is just a suggestion though, as nothing is sure before practical tests have been done.

4.7 Concluding discussion

Several design suggestions for the Stewart platform has been presented. An initial and somewhat standard design has been created and used for creating a prototype design and 3 experimental designs. Issues regarding mechanics and workspace have been discussed. The mechanics of the actuators allow a good accuracy, but inhibits speed. The orientation within the workspace is limited. The experimental designs tried to both increase the possible workspace and the orientation within the workspace. Assembly testing within a CAD-software environment suggests that the third experimental design might accomplish this in an implementable manner.

A simple prototype of a single arm was printed on a commercial 3D printer in order to suggest that the Stewart platform in the future will be capable to create the plastic parts it is made of. As self-replication is one of the central aspects in this thesis, a printable design is very important. The succesful print of the parts (figures 4.12 and 4.13) proved that this is very likely to be true. The next step in this regard could be to expand on the current prototype design to become more applicable as parts for a Stewart platform, in other words to increase the length of the nut housing, to put together and test an entire Stewart platform according to the prototype design or to use the experience from creating a prototype design to perform the same analysis and changes to the third experimental design.

The comparison with the RepRap Mendel suggests that the Stewart platform can in fact be an alternative to the current RepRap design. It is important to stress the *can* aspect of this possibility. Much more practical development and testing have to be performed before the *can* turns into a *is*. The above discussion has also showed that true 3D milling with a Stewart platform can be very difficult because even though it has 6 DOF, not all orientations are mechanically reachable. However, 2.5 D milling seems very possible and also to allow the milling tool some tilt. A solution to this problem has been done by PKMTricept [50] as can be seen in figure 2.4. Here, instead of a Stewart platform (or a *hexapod*), a tripod is used for 3 DOF positioning. The platform hangs from a structure and at the movable platform a small and rigid serial manipulator (robot arm) is positioned. Depending on the model, this has 2 - 3 DOF and allows a quite arbitrary orientation of the milling tool. This solution is a professional and industrial one and could not be easily implemented within the RepRap concept. Maybe if the manipulator is attached to the underside of a desk, a solution can be reached. However, these are just speculations.

This chapter shows more than anything that there are many areas to explore with the Stewart platform and research possibilities are abundant and varied. The CAD software was very useful for testing the motion of the Stewart platform in assembly mode where the different parts can be connected together with the help of mates and the actual movements of the manipulator can be studied.

One aspect that has not been studied is how the actual milling tool and extruder is to be attached to the moving platform. This is a practical problem that relies much on the kind of milling tool and extruder and should not be hard to overcome. A solution could be to have some kind of holder on the moving platform, allowing the milling tool and the extruder to be changed according to what procedure the robot is going to perform next. Another aspect that remanins untouched is the issue of rigidity. Practical testing or advanced dynamical testing is needed to verify whether the presented designs are rigid enough to mill.

Chapter 5

Simulator

A simulator was developed for the Stewart platform in order to be able to analyze the movements and actions of the platform and to further study it in the future. The simulator was written using the python language (see chapter 3.1) and developed with inverse kinematics that describes the position and orientation of the platforms and legs when the position and orientation of the end-effector is known. As mentioned earlier this is quite simple for a parallel manipulator, while the other way around, deciding where the end-effector is given the length of the legs, is mathematically complex. When working with 3D printing and CNC milling, the location and orientation of the end-effector is always given, thus this does not pose a serious problem.

5.1 Uses for the simulator

There are several possible uses for a simulator. However, the current simulator needs some improvements to allow all the listed functions to be possible. Circular interpolation is for example important to have implemented.

- Testing code to make sure no collision happens.
- Testing code to make sure no legs are to move beyond their limitations
- Make sure the tool path is correct
- Examine how many degrees a motor has spun
- Examine the length of the tool path
- Examine execution time
- Examine the size of the workspace

The functions that have been implemented are to make sure no collision happens, make sure no legs move beyond their limitations and make sure the tool path is correct. To examine the workspace is reportedly a complex matter and this problem has not been studied. The rest of the functions that have not been implemented are discussed in chapter 5.4.2.

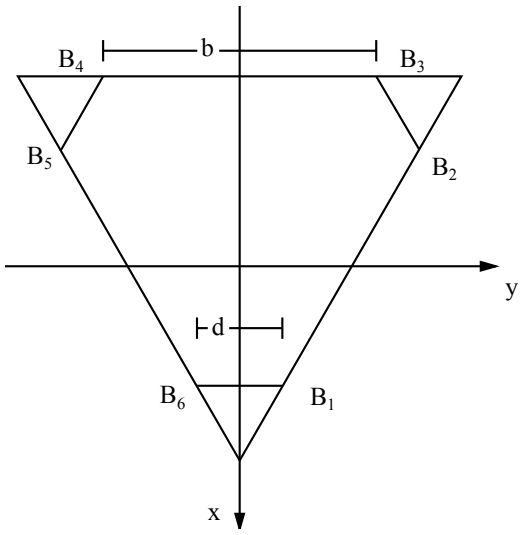


Figure 5.1: These are the dimensions for the base platform

5.2 Mathematics for the Stewart platform

The inverse kinematics used to relate the different parts of the Stewart platform is given here. As described earlier, the inverse kinematics uses the position and orientation of the end-effector and finds the position and orientation of the rest of the robot parts (two platforms and six legs). What is essentially done is to create three coordinate systems and assign them to each part, one for the base platform, one for the top platform and one for the end-effector. The positions of the six legs on each platform are then decided. This depends on how one wants the platforms to look like and can be quite arbitrary. By using homogenous transformation between the coordinate systems, the top platform and end-effector coordinate system can be described from the base coordinate system. In this way all the important points of the manipulator is known and it is possible to draw lines between these points in order draw the manipulator. The following inverse kinematics and platform dimensions are heavily inspired by [35].

$$B_1 = \begin{bmatrix} \frac{\sqrt{3}}{4}b \\ \frac{d}{2} \\ 0 \end{bmatrix} B_2 = \begin{bmatrix} -\frac{\sqrt{3}}{4}b \\ \frac{1}{2}(b+d) \\ 0 \end{bmatrix} B_3 = \begin{bmatrix} -\frac{\sqrt{3}}{2}\left(\frac{b}{2}+d\right) \\ \frac{b}{2} \\ 0 \end{bmatrix} \tag{5.1}$$

$$B_4 = \begin{bmatrix} -\frac{\sqrt{3}}{2}\left(\frac{b}{2}+d\right) \\ -\frac{b}{2} \\ 0 \end{bmatrix} B_5 = \begin{bmatrix} -\frac{\sqrt{3}}{4}b \\ -\frac{1}{2}(b+d) \\ 0 \end{bmatrix} B_6 = \begin{bmatrix} \frac{\sqrt{3}}{4}b \\ -\frac{d}{2} \\ 0 \end{bmatrix} \tag{5.2}$$

$$T_1 = \begin{bmatrix} \frac{\sqrt{3}}{4}a \\ \frac{a}{2} \\ 0 \end{bmatrix} T_2 = \begin{bmatrix} \frac{\sqrt{3}}{2}\left(\frac{a}{2}-c\right) \\ \frac{1}{2}(a+c) \\ 0 \end{bmatrix} T_3 = \begin{bmatrix} -\frac{\sqrt{3}}{2}\left(\frac{a}{2}+c\right) \\ \frac{c}{2} \\ 0 \end{bmatrix} \tag{5.3}$$

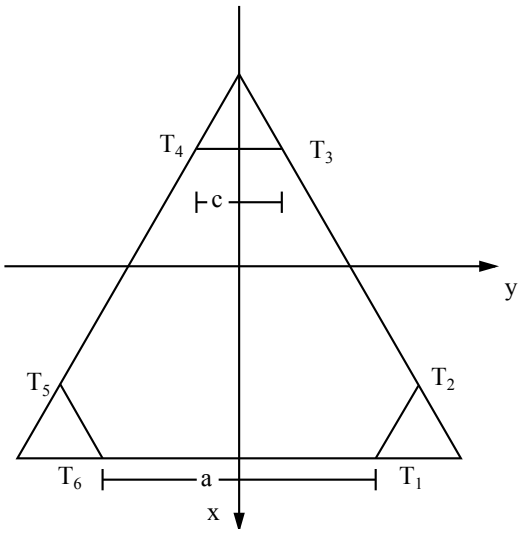


Figure 5.2: These are the dimensions for the top platform

$$T_4 = \begin{bmatrix} \frac{\sqrt{3}}{2}\left(\frac{a}{2} - c\right) \\ -\frac{c}{2} \\ 0 \end{bmatrix} T_5 = \begin{bmatrix} \frac{\sqrt{3}}{2}\left(\frac{a}{2} - c\right) \\ -\frac{1}{2}\left(a + c\right) \\ 0 \end{bmatrix} T_6 = \begin{bmatrix} \frac{\sqrt{3}}{4}a \\ -\frac{a}{2} \\ 0 \end{bmatrix} \tag{5.4}$$

The dimensions for the base platform are described in figure 5.1. The exact locations in relation to the base coordinate system are described in the equations (5.1) and (5.2). Similarly for the top platform in relation to the top coordinate system can be seen in figure 5.2 and equations (5.3) and (5.4). Legs 1-6 will be connected to the points B1-B6 and T1-T6. This mathematical representation is not identical to the designs that were presented in chapter 4.3.3 when regarding the universal joints, but is rather for a more general Stewart platform. A more design-specific simulator should be created when a certain design has been chosen, built and tested.

5.2.1 Homogenous Transformation

Homogenous transformation relates two coordinate systems using matrices. The normally used Euler angles is not used here as they cause the Jacobean matrix (if ever implemented for this system) to become singular even when not in a singular position for a Stewart platform.[35] Rather, to get from the moving platforms coordinate system to the base platform coordinate system, rotations, first about the x axis with α degrees, then about the y axis with β degrees, and finally about the z axis with γ degrees, are performed. The x and y axes are orientated as described in figure 5.2 and the z axis is normal to both of them. In other words, they are all normal to eachother. The resulting homogenous transformation matrix is described in equation (5.5). In the equation, c stands for cosine, and s for sine, while x, y and z is the location of the moving platform. For further details, see [35] and [23].

$$\begin{bmatrix} c(\beta)c(\gamma)+s(\alpha)s(\beta)s(\gamma) & -c(\beta)s(\gamma)+s(\alpha)s(\beta)c(\gamma) & c(\alpha)s(\beta) & x \\ c(\alpha)s(\beta) & c(\alpha)c(\gamma) & -s(\alpha) & y \\ -s(\beta)c(\gamma)+s(\alpha)c(\beta)s(\gamma) & s(\beta)s(\gamma)+s(\alpha)c(\beta)c(\gamma) & c(\alpha)c(\beta) & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.5}$$

Since the end-effector and the top platform are going to be normal to each other, it is quite easy to derive the homogeneous transformation between them. The matrix can be seen in equation (5.6).

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & toollength \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.6}$$

5.3 Parsing G-code

A G-code parser (interpreter) was created in order to allow the simulator to be controlled by the common machine controlling language, G-code. The interpreter creates lists of the command given and position and orientation for each frame in the resulting simulation video. G-code is commonly used for CNC milling machines, but also used by the RepRap system. It has several commands, and not all were implemented for this simulator. Those that were, are listed in table 5.1

Command	Name	Short description
G00	Rapid Positioning	Fast movement
G01	Linear Interpolation	Slow movement
G04	Dwell	Pause
G21	Programming in millimeters	The numbers for positioning etc are in millimeters
G28	Return to home position	Return end-effector to position zero
G43	Tool length compensation negative	Adds a negative length to the postion of the tool
G44	Tool length compensation positive	Adds a positive length to the position of the tool
G49	Tool length compensation cancel	Cancels the offset and use the standard position
G90	Absolute programming	The numbers for positioning and orientation are with refernce to position zero
M00	Compulsary stop	The machine will stop
M02	End of program	End of the entire program
M03	Spindle on (clockwise)	Turn on milling tool in the clockwise direction
M04	Spindle on (counterclockwise)	Turn on milling tool in the counterclockwise direction
M05	Spindle stop	Turn off milling tool

Table 5.1: Interpreted G-code [54]

Notable omissions are programming in inches, local coordinate system, incremental programming and circular interpolation. Also, subroutines have not been implemented. A complete G-code parser that would incorporate all the different variations of G-code is very time consuming and hard to create. However the most important commands for this simulator is the rapid and linear positioning (location and orientation) to be able to locate the end-effector, and from that draw the entire Stewart platform. In G-code X, Y and Z sets the position, A, B and C the orientation, H is positioned in front of tool length offsets and P, X and U is used in front of the value for dwell time.

The syntax or format of different G-code files has been found to be quite different. There is for example a difference in the use of lined numbers. The parser made for this thesis reads only G-code with numbered lines according to the "Nx standard" (for example N10, N20, N30 etc). This is how the syntax of G-code has been presented in a classic CNC machining book.[54] Because of these variations in syntax, some G-code files can not be read at all by the parser.

5.4 Results

The simulator has been tested in order to verify it capabilities. In order to be able to perform most of the functions listed in chapter 5.1, the simulator should be capable of

- Determining the current command
- Determining the specified speed
- Determining the calculated absolute speed of the end-effector
- Determining whether the tool is on or off.
- Determining how much the tool offset is specified to. This is necessary for milling where the bits might have different sizes.
- Determining the position of the tip of the end-effector
- Determining the orientation of the end-effector
- Determining the absolute speed for each of the six legs
- Create a moving image of the Stewart platform
- Show the end-effector path

The size of the platform in the simulation have been specified to be the same as the ones presented in chapter 7.1 about the design. The value for a and b is 400 mm and c and d are 250 mm. The tip of the end-effector is located 150 mm away from the top platform. The maximum speed of the platform is discussed in chapter 4.1. For the simulator the speed is sat to 30 mm/s for fast movements (when the tool is turned off) and 15 mm/s for slow movements (when the tool is turned on). The speed for slow movements can not in a milling application be decided like this, but have to be adjusted according to the material to be milled, spindle speed, the type of bit used by the milling tool and so on. The

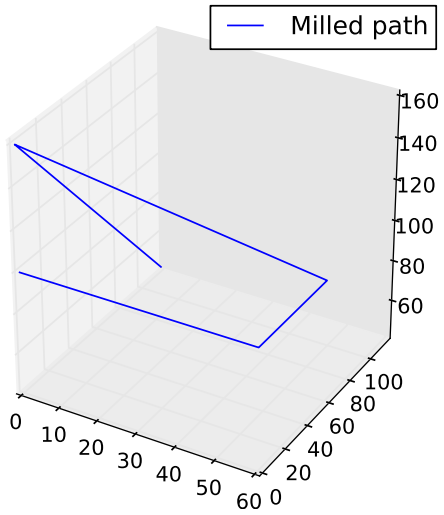


Figure 5.3: The path created for testing simple move

G-code can specify the desired milling speed with the feedrate (for example G95 - feedrate per revolution [54]), which is based on an analysis the CAM software have performed.

Two sets of data has been has been tested. The first set is just positions created in the software to make sure the platform moves as intended. The second set is the milling of a plaque reading "CNC" taken from a website with the owners consent. [20] The second set is used to verify that the simulator works reasonably well with G-code produced for milling. As discussed in chapter 5.3, not all the different types of G-code syntax can be read by the parser, and therefore (among other things) only one file of G-code has been tested. It is believed that this is sufficient to test the simulator as the contents of the files are essentially the same (the same at a very basic level: operate a tool). However, the file tested does not have advanced commands such as for example sub-routines. The G-code parser does not handle such commands. More knowledge is needed about G-code techniques, ideas and parsing to implement all of the possible commands given by a G-code file.

5.4.1 Simple move simulation

The path for the simple move test can be seen in figure 5.3. The test was successful as can be seen in figure 5.4.

5.4.2 G-code simulation

The path created from the G-code can be seen in figures 5.6 and 5.5. Creating a simulation of the entire path would take a long time (the G-code file has about

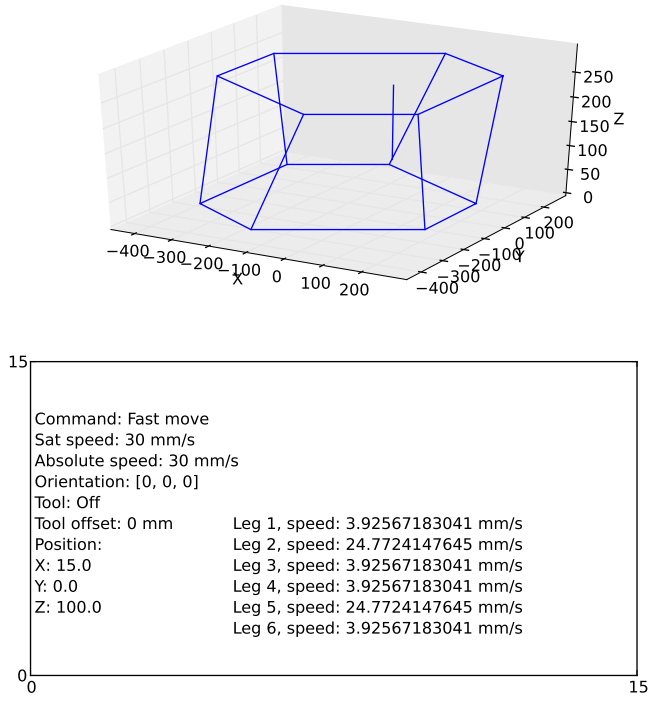


Figure 5.4: The simulation of the simple move

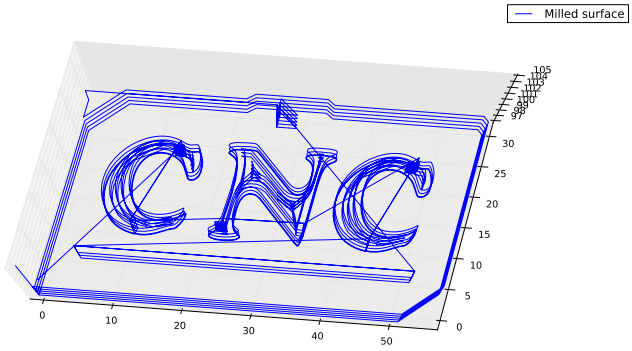


Figure 5.5: The path extracted from G-code, seen from above

4000 lines of code), therefore only a small portion of the path was used, see figure 5.7. A frame from the resulting animation can be seen in figure 5.8. The same figure also shows that all the listed functions minus drawing the path have been implemented. The drawing of an end-effector path has been programmed to be shown in a separate image.

The simulator is already capable of doing some of the uses listed in chapter 5.4. The simulator makes sure no movement of the end-effector makes the legs move faster than the desired speed and that the legs do not exceed a maximum and minimum length. Also, execution time is roughly equal to the length of the resulting animation file. Some simple additions can make the simulator calculate an exact execution time. Other things that are quite easy to implement are calculating the length of the tool path and how many degrees each motor has spun. To make sure the tool path is correct a visual inspection by the tool path can be done. However for a more in-depth and maybe even an automatic check, the size and shape of the milling tool have to be included and possibly a 3D surface of the resulting path created. Shape is also a keyword when making sure no collisions happens. One has to know the exact shape of the Stewart platform in order to accomplish a true collision control system. The current simulator can be improved to detect collision but would not cover every collision possibility since it does not include the shape of the Stewart platform.

5.4.3 Comment

The resolution of the end-effector path, the frames per second (FPS) of the movie and the speed of the end-effector are related. This is because the movie is created by animating *.png images. The default value for the FPS in the used software is 10 frames per second. In this way, each image represents 0.1 second of movie. Each move command in the G-code is fitted within the frames per second and speed. For example if a move command moves the end-effector 20 mm and the desired and max speed is 10 mm/s the move command have to be divided into 20 equal sized parts. Then each image represents a movement of 1

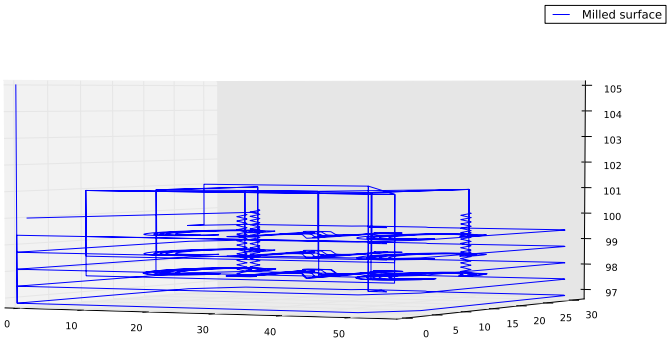


Figure 5.6: The path extracted from G-code, seen from the side

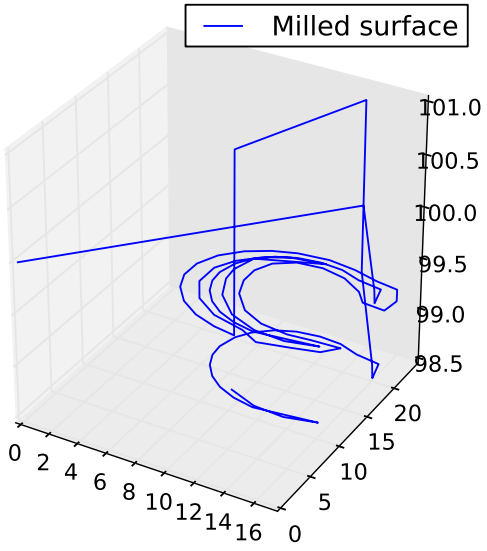


Figure 5.7: The part of the path used for simulation

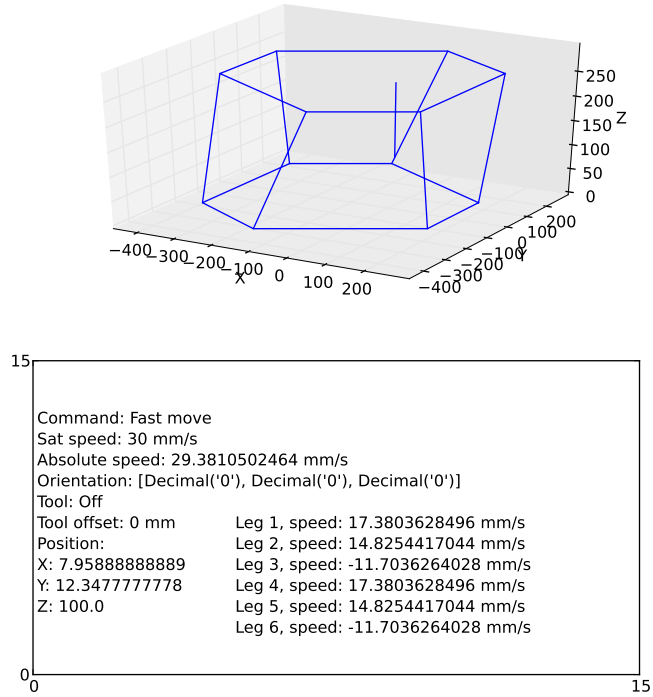


Figure 5.8: Simulation of the G-code with a fast speed of 30 mm/s and slow speed of 15 mm/s.

mm (which is a tenth of the maximum speed per second) and fits well with the fact that each frame is to be a tenth of a second. On the other hand if the move command only induces a movement of 0.1 mm, this will cause the simulated platform to move a tenth of the desired speed. Therefore, this movement is ignored for the next command that makes the end-effector move 1 mm or more. To sum up, a high speed needs a high FPS to keep the smallest move commands in the G-code. However, when these distances are very small (less then a tenth of a millimeter for example) this can for simulation be ignored. This is of course unless certain high detail simulation is wanted. Also worth noting, high resolution will cause the parsing and animating to take a very long time if the G-code file is long. What slows the program down is the saving of *.png images and making the move commands shorter.

5.5 Concluding discussion

A simulator has been created that is able to handle several simple G-code commands and create an animation that shows the Stewart platform together with

details of position, orientation, speed, tool offset, tool state (on/off) and current command. There are still several commands that have not been implemented. Whether the simulator needs to be able to handle all the commands in the G-code syntax depends on the application the simulator is used for. Another issue is that there exist no standard G-code. A command might mean different things when written for different applications. The simulator is capable of being used as an analyzing tool to discover collisions, to make sure the tool path and the length of the legs is correct and to examine speeds and time of operation. The two first have to be done manually, by studying the animation and a figure of the tool path. Other capabilities such as calculating the length of the tool path and calculating the number of degrees the motors have spun is not implemented, but can quite easily be so. A more sophisticated and accurate collision analysis and time calculation is possible. Workspace analysis is also possible, but has not been studied in this thesis due to its complexity. The study and creation of a simulator has successfully deepened my understanding not only of the Stewart platform, but also of the G-code language.

Chapter 6

Biologically Inspired Path-Optimization

The goal for this chapter is to present two possible solutions to improve the path of the end-effector, make a short comparison and discuss the applicability of the algorithms and the current approach to path-optimization (shortening the paths where the tool is inactive). The two algorithms presented are the genetic algorithm and the ant colony optimization algorithm. One of the reasons for doing this study is that some papers have looked at the possibility to do the similar things with positive results (see chapter 2.7 for details). Another, more obvious reason is that by optimizing the tool path, machining time can be reduced. The idea is to find shorter paths between the active paths, in other words, to find shorter inactive paths. I hope to take the research a step further by including actual G-code used for CNC-milling in the equation. This quickly makes things much more complicated as G-code does not have a standard and there are many practical problems that can occur. As discussed later, some of these practical problems have to some extent been ignored because they are outside the scope of this thesis.

6.1 Research method

Eiben and Smith discuss how to work with evolutionary algorithms and how to measure their performance.[19, p241-258] Working with evolutionary algorithms (and ant colony optimization) most often takes an experimental approach. The use of randomness in the algorithms demands this as it will not behave the same every time. An experimental approach is when the algorithm is implemented and run with specified parameters on a set of test data and results are recorded and analyzed. This is repeated for different parameters or perhaps different test data and the different results are compared with each other in order to get an impression of the algorithm's performance.

The first distinction they make is between design (one-off) models and repetitive problems. The path-optimization problem is a kind of hybrid. It is a repetitive problem in that it will be done several times. For repetitive problems all runs of the algorithm need to present a solution that is good enough. This is the opposite for design problems, where one really good solution is needed.

However, since the amount of time in this situation is not as important as for a typical repetitive problem, the algorithm can have an adequate speed or be run several times in order to locate a near optimal solution. If the solution is not good enough, the user can decide to use the original path or run the algorithm again. This do not mean that the algorithm can behave like a *design algorithm*, but has some flexibility when it comes to producing good enough results exceptionally fast versus producing optimal results. The input data is different from run to run so the algorithm has to handle a wide variety of different problem instances.

This study is on an application oriented situation and deals more with the problems of making the algorithm workable than pure academic research. Also, the focus has not been on proving that the genetic algorithm or ant colony optimization is better than existing solutions, but rather to show that the two algorithms in fact can be used on *exactly* this kind of path-optimization. Another focus is to see how the different parameters affect the solution. As only one test data set has been used, different parameters might be more suited to other problem instances.

There are several ways to measure the performance of an algorithm. Eiben and Smith suggests that success rate, effectiveness (solution quality), efficiency (speed) and progress curves are the most essential ones.[19] In practical application, as this is, success rate is the percentage of times the algorithm presents a sufficient solution. With the problem at hand, a sufficient solution is hard to determine, especially since the problem is not tested thoroughly in the literature. For example there has not been a testing of thousands of different CNC milling and 3D printing paths that concludes an optimization should at least reduce the path length to 50% of the original length. Another problem is that different paths have different qualities. Some might be written more or less by hand, while other might be created by professional and expensive CAM software. Other again might be created by open source beta software. For these reasons, no success percentage is used but rather the actual percentage of the original length is monitored.

Another measure is the mean best fitness (MBF). This is the best fitness over a number of runs with the same parameters, same test set and same algorithm. This measures how stable the algorithm is in a static environment. Two other measures are also included, best ever fitness and worst ever fitness. Worst ever fitness is the fitness over a number of runs that is best for a specific run but worst compared to the best of the other runs. Best ever fitness is the best fitness over a number of runs.

Dealing with algorithm efficiency, the average number of evaluations to a solution (AES) is used. When analyzing AES it is important to keep in mind that the results can be misleading. For example some evaluations use a longer time than others and the algorithm might use local search or other hidden labor. For the presented problem an evaluation is to calculate the length of the tour. The AES for the algorithms implemented for this thesis is not very misleading. For the ACO the number of evaluations is equal to the number of ants times the number of generations for a run. If the number of generations is the ending criteria, the AES is given. This is not true for the genetic algorithm where an individual might survive the crossover and mutations and do not need to be evaluated again.

Progress curves shows the best solution for each generation and can be used

Figure 6.1: A plaque reading CNC taken from [20]

to determine how many generations is needed to give a good result. Progress curve is for one run only

6.2 Parsing G-code

The G-code parser works in a similar fashion to the one presented for the simulator in chapter 5.3. The parser finds where different active parts starts and ends. The distances between different active paths and between the paths and position zero are calculated and put into a distance matrix.

6.3 Test data

The test data is for CNC milling. Note that several things have been left out. For example tool change and paths of circular interpolation. This is (among other things) because the G-code parser was created from scratch, and to allow time for testing etc, it was not created to include every common G-code command. For an overview of G-code parsing see chapter 5.3. The data set was found on a hobby website and used with the owner's permission (it is the same G-code as used in the design chapter).[20] The test data from the hobby-site is the milling of a plaque reading CNC, see figure 6.1. As described in chapter 5.3, G-code files from SolidCAM and other G-code files were found to have a different syntax and could not be tested. The downside of using only one file as test data is that the testing will not reveal whether the results are applicable to different instances. However, to study how applicable the algorithms are for different instances, detailed knowledge of the different *types* of typical paths used for CNC milling and 3D printing is needed. This has not been of focus in this thesis. The upside of studying only one file of test data is that the data can be tested and studied more thorough.

A similarity exists with the paths of 3D printing and CNC milling. Both have parts where the tool is used and parts where the tool is passive. This study focus on the parts where the tool is passive (inactive paths). When ignoring the possibilities of collision, the movement between the ends of the parts where the tool is active is still not quite arbitrary. For example starting to print at the top level of a part will have catastrophic results. Similarly, milling behind to-be-milled areas will not work. These problems (collision and the order of the paths

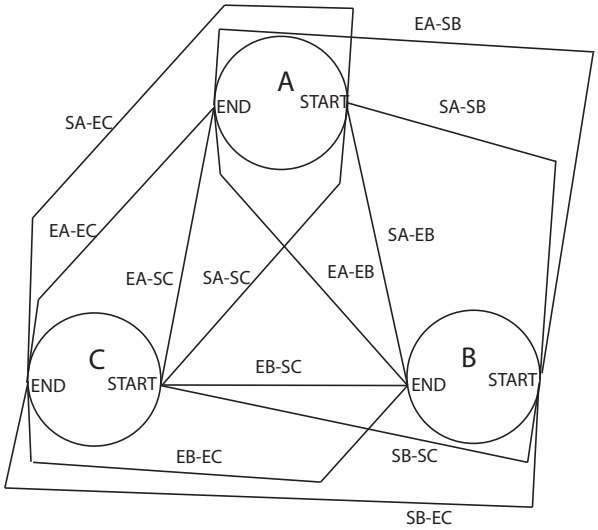


Figure 6.2: 3 cities and the paths between them

where the tool is active) have not been included in the algorithms. They are, however, discussed and dealt with in a final discussion for the chapter. Using this simplification, what is left is essentially the Traveling Salesman Problem (TSP) with an alteration. The TSP is a NP-hard optimization problem introduced in chapter 7.2. The goal of the TSP is to find the shortest round trip between N number of cities where every city is to be visited only once. The cities are connected with edges that describe the cost to move between them. In this application the cities are represented by a path where the tool is constantly active. The edges are the distance between these active paths, in other words paths where the tool is inactive. As mentioned, all the active paths are connected to the other active paths, ignoring practical problems. The alteration is that there are actually four ways of moving from one active path to another, start to start, start to end, end to end and end to start. All these distances are different and also decide what end of the path is available to connect to the next path. An example with 3 paths is given in figure 6.2. Here, the paths A, B and C have had their connecting routes shown. If the tool were to move along path SA-EB, the next moves available are only SB-SC and SB-EC. If SB-EC is chosen, the remaining move is given, SC-EA.

Because the cost matrix for this problem instance has 3 dimensions (from-city, to-city and what edge between the two cities) as opposite to the typical 2 dimensions, the cost matrix (the area to be searched) becomes larger and solving the problem becomes tougher. The graph is currently undirected, which means that travelling from i to j along path n is the same as travelling from j to i along path n. If collision control is implemented, the graph might become directed, something that will increase the search space even further.

6.4 Genetic Algorithm

For a general description of the genetic algorithm, see chapter 2.11. What follows is a mapping from some of the standard operators used in the genetic algorithm to this problem instance. The operators have been found in the book [19].

6.4.1 How it works - specific

Here, the details for the genetic algorithm used in this thesis are presented. Genetic algorithms are the most commonly known and most researched evolutionary algorithm.[19, p 37] The algorithm is created for each application with different operators. The operators to define are:

- Representation of individuals
- Initialization operator
- Parent selection operator
- Crossover (recombination) operator
- Mutation operator
- Survivor selection mechanism
- Termination operator

Representation

The representation defines much of how the other operators are going to look like and is therefore chosen first. For the travelling salesman problem, permutation representation is the most common choice. The permutation can for example be a number where each digit represents a city or a list where each item represents a city. For the path-optimization problem presented here, the parsing of G-code affect what kind of representation can be used. As described in chapter 5.3, the parsing creates a list of paths, describing the start position, end position and the type of movement. Each active path is given a number and the distance between them is stored in a cost matrix. Notice that between each path there are 4 possible distances, so the cost matrix will have 3 dimensions. Since there are four possible ways to move between two active paths and the selected option affect which way to move from the next active path, the path chosen between two cities have to be included in the representation. Therefore the representation of an individual consists of a list of lists with two elements. Each list inside the list is made up of a number representing an active path/city and a second number representing the edge used between the current and the next path/city. For the edges, 0 represents a move from start of current to start of next, 1 represents a move from start to end, 2 represents from end to start and 3 represents from end of the current path to the end of the next path. An example of an individual with the genes 1, 2 and 3: [[2, 1], [3, 0], [1, 3]]. Here the edge is from the start of path two to the end of path 3, from the start of path 3 to the start of path 1 and from the end of path 1 to the end of path 2.

Initalisation

The initialization creates a number of random permutations that is equal to the population size. Each individual is then given random edges between the active paths (or genes or cities).

Parent selection

$$p_i = q(1-q)^i \tag{6.1}$$

$$a_i = \sum_{1}^{i} P_{sel}(i) \tag{6.2}$$

The population is first ranked according to their fitness and the equation (6.1).[17] A larger value for the parameter q, equals larger selection pressure. The equation gives each individual a probability of being chosen as a parent. In this algorithm the number of individuals (μ) is equal to the number of parents (λ). The number of offspring is equal to the number of parents. After each individual has had their probability calculated they are ranked from one (least fit) to μ and given a value [a1, a2, ..., aμ] according to equation (6.2).[19, p 62]

Algorithm 1 **begin**
set *current_member*=i=1
Pick a random value r **uniformly from** $[0, 1/\mu]$
while *(current_member* $\leq \mu$ *)* ***do***
while *(* $r \leq$ a[i] *)* ***do***
set mating_pool[*current_member*] = **parents**[i]
set $r = r + 1/\mu$
set *current_member* = *current_member*+1
end while
set $i = i$+1
end while
end

Psueudocode fot the stochastic universal sampling (SUS) algorithm. Taken from [19, p 60-63]

Parents are then selected according to the stochastic universal sampling (SUS) method.[19, p 60-63] This is similar to spinning a roulette wheel one time with a number of equally spaced arms that is equal to the number of parents that are to be chosen. The pseudocode for SUS is presented in algorithm 1.

Crossover

There are mainly four crossover methods that are suitable for permutations, Partially Mapped Crossover (PMX), Edge Crossover, Order Crossover and Cycle Crossover.[19, p 52-56] Of these, Eiben and Smith reports that PMX is the most widely used for adjacency-type problems such as the TSP. A slightly modified PMX has therefore been chosen as crossover operator. The modification has been done because there are four possible edges between two genes and the edge used by the current gene is dependent on the edges used by the gene

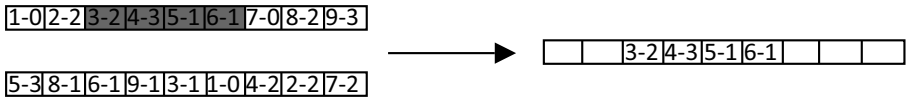


Figure 6.3: PMX 1: copy a segment from one of the parents to the offspring. [19, p 53]

Figure 6.4: PMX 2: add the same segment from the second parent. 6 and 3 are already added and are ignored. 9 is opposite to 4 in the first parent. Add 9 in the position 4 is in the second parent. 1 is opposite to 6 in the first parent. 6 in the second parent is inside the grey area, opposite to 6 is 3, 3 is also in gray area in the second parent. Opposite to 3 is 5, add 1 in the position 5 is in the second parent.[19, p 53]

before and the next gene. For illustration, see figures 6.3, 6.4 and 6.5. The issue with the four possible edges between two genes is solved as described in figure 6.6.

Mutation

Similarly to the crossover operation, Eiben and Smith present 4 common operators for mutating permutations, swap mutation, insert mutation, scramble mutation and inversion mutation.[19, p 45-47] A modified inversion mutation is chosen as mutation operator for this implementation. Similarly to the crossover operator, the modification is due to the four possible edges between two genes. Inversion mutation chooses a subset of the permutation and inverse the direction of this subset. This can be seen in figure 6.7 and the modification is seen in figure 6.8.

Survivior selection

The survivor selection mechanism can be very crucial in deciding whether the algorithm will explore the search space to a satisfactory extent. If only the fittest individuals are chosen for the next generation the algorithm can quickly end in a local optimum. The survivor selection for this implementation of the genetic algorithm is age-based replacement with elitism.[19, p 65-66] Age based replacement do not consider fitness when deciding which individuals that is to

Figure 6.5: PMX 3: Add remaining parts of the other parent. Switch positions of the parent and repeat to make another offspring.[19, p 53] The edges have to be adjusted, see figure 6.6

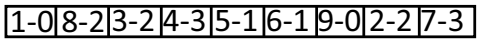


Figure 6.6: Correcting the edges. Starting with gene number two, look at the gene before, decide to use 0/1 (start - start/end) or 2/3 (end - start/end). Look at the next gene, this decides what number that is to be used. Continue for each gene until the last. The last use the gene before and the first gene to decided the edge.

Figure 6.7: Inversion mutation, based on [19, p 47]. The direction of the paths have to be adjusted, see figure 6.8

survive for the next generation, but says that each individual is to live for the same number of generations. Since the number of individuals in a population and the number of offspring created is the same, all individuals are replaced by the offspring. Elitism is to allow the fittest individual to survive and replace it with the least fit individual in the offspring. What this translates to, is that each individual in a population only lives for one generation except if they survive the crossover and mutation, or is the fittest individual.

Termination

As described in the chapter about methods for testing evolutionary algorithms, it is hard to know in advance how much to expect the algorithm will improve the inactive path length. Therefore the termination operator has been chosen to be a number of generations.

6.4.2 Results for the Genetic Algorithm

The test data used has already been discussed in chapter **??** and can be seen in figure 5.5 at page 57. The methods for testing have also been described in chapter 6.1. This genetic algorithm has 5 different parameters. Population size and number of generations that decides how many individuals that are in a population and how many generations the algorithm is going to run. The parameters p_c and p_m decides how often parents are going to be subject to crossover and mutation respectively. The last parameter, q, is used to rank the individuals according to equation (6.1). Larger q equals larger selection pressure. Different values for the parameters have been tested and are presented in tables 6.1 and 6.2. Progress curves for some parameter settings can be seen in figure 6.10 and 6.11. The total original length of the active and inactive paths is 2778.7

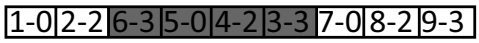


Figure 6.8: Correcting the edges. Starting with gene number two, look at the gene before, decide to use 0/1 (start - start/end) or 2/3 (end - start/end). Look at the next gene, this decides what number that is to be used. Continue for each gene until the last. The last use the gene before and the first gene to decided the edge.

mm. Of these is the tool active for 2508.9 mm and inactive for 269.8 mm. As can be seen, the inactive path is only 9-7% of the total lenght, something that really limits the path-length that can be saved.

Parameter setting	1	2	3	4	5	6	7	8	9	10	11
Population size	100	100	100	100	100	100	25	25	500	60	60
Generations	50	50	50	50	50	50	50	50	50	45	45
p_m	0.1	0.1	0.1	0.6	0.0	0.0	1.0	1.0	1.0	1.0	1.0
p_c	0.6	0.6	0.6	0.1	1.0	1.0	0.2	0.2	0.2	0.3	0.3
q	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Runs	10	100	200	10	10	50	10	100	10	10	100

Table 6.1: Parameter settings for the different tests. Test results can be seen in table 6.2.

Parameter setting	1	2	3	4	5	6	7	8	9	10	11
Best ever	123.7	104.1	102.1	106.0	100.8	100.3	100.4	101.0	97.4	105.5	98.3
Percentage of original inactive path	46.1	38.6	37.8	39.3	37.4	37.2	37.2	37.6	36.1	39.1	36.4
Total new length percentage of total original length	94.7	94.0	94.0	94.1	93.9	93.9	93.9	93.9	93.8	94.1	93.8
Worst ever run	196.9	213.3	211.3	213.3	180.8	173.2	159.2	199.1	167.8	148.3	195.0
Mean best	155.3	157.1	148.6	156.2	135.2	134.3	135.9	147.4	127.2	123.8	140.7
Average number of evaluations	3313	3302	3297	3295	5101	5101	1327	1327	25501	2761	2761

Table 6.2: Test results for different parameter settings for the genetic algorithm. The values of the parameters can be seen in table 6.1. The orignal length of the inactive path is 269.8 mm. Total original length with active and inactive paths is 2778.7 mm.

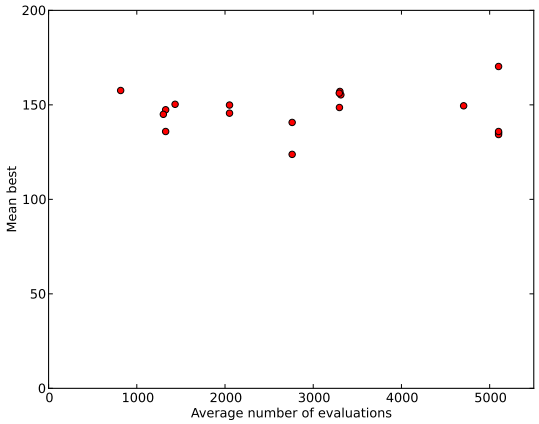


Figure 6.9: The Mean best vs the Average number of evaluations for the GA. Some of the test results presented here is not included in table 6.1 and 6.2

6.4.3 Discussion on GA test results

The goal of this discussion is to highlight different aspects of the genetic algorithm in the given context. Another goal is to discuss the applicability of genetic algorithm to the given problem. Because there has not been implemented any traditional form of optimization algorithm in this thesis to compare with, this have to be done in relation to how the genetic algorithm would perform within acceptable limits and how an increase in complexity (collision control for example) would affect the algorithm. The number of parameter settings has been limited to 11 to be able to have some sort of overview. However, this means that when comparing two types of settings, more than one parameter might be different. This is important to keep in mind when reading the discussion.

General

When taking the setting into consideration, the speed of the algorithm is not the most important characteristic. A speed of about 30 seconds is acceptable, while several minutes would be too slow. The time has in this testing been omitted and an average number of evaluations has instead been used. Still, while not measuring the time in detail, an approximate time could be observed. During testing, no single run took close to 30 seconds. According to the results a larger number of evaluations do not necessarily mean a much better mean best as can be seen in figure 6.9. On the other hand, none of the mean best are very convincing when compared to the best ever. This means that, in order to locate a very good solution, several runs are needed. After all, to be used in the 3D printing or CNC milling process, a very good solution is necessary.

Since this is just a single sample of the endless varieties of how G-code can describe active and inactive parts it is hard to generalize the parameters used with any success. An idea that can be used in conjunction to using several runs to find the best solution is to use different parameters for the different runs. In this way, the algorithm might prove to be applicable to many different kinds of

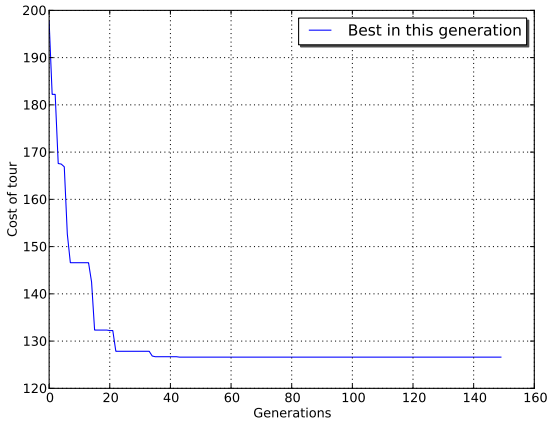


Figure 6.10: The best individual for each generation using parameter setting 4 in table 6.1 and 150 generations (genetic algorithm)

paths.

When considering the issue of the order of active parts (for example starting to print at the top level), the problem can be solved by analyzing the data correctly. In this way, edges in the search space can be removed in order to deny certain movements. Also, direction of the edges can be applied. The issue of collision control could prove much harder to solve however. Solutions for these problems do exist and they do not affect the algorithm specifically, but the preparation of data for the algorithm. This thesis will not look any deeper into the problems of collision control and the order of the active parts.

Number of runs

From the results of the first three parameter settings it can be seen that with more runs (10 and 20 times more, setting 1, 2 and 3), the best result improves, while the mean best result do not change that much. The first part is expected as more runs give a higher chance of getting a better result. The fact that the mean best is best with 200 runs is quite interesting. With different parameters (settings 5, 6, 7 and 8) the best results do not change much with 10 times the number of runs. While setting 10 and 11 show that an increase in the number of runs can result in an improved best result and an increase in the average best distance. As the results points to both sides, it is difficult to analyze the impact of the number of runs. However, it is fairly logical that with more runs, a higher chance of achieving a better best solution is gained at the cost of time. The numbers also shows that algorithm produces fairly similar results when ran several times. This suggests that there is little variance.

Mutation and Crossover rate

When looking at table 6.1 and 6.2 it is hard to make out any favorable combination of mutation and crossover rate. The mutation rate has been kept high in most of the settings while crossover rate has been kept quite low. This seems

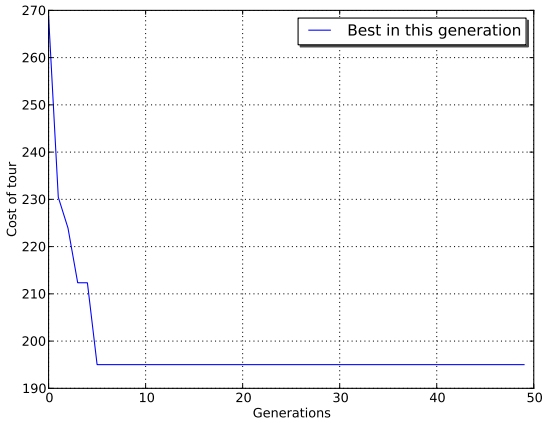


Figure 6.11: The best individual for each generation using parameter setting 5 in table 6.1 and 50 genereations (genetic algorithm)

to give good results, however the opposite is also possible (see setting number 5 and 6).

Population size and number of generations

The number of generations was set to be the ending criteria as it is impossible to say, from file to file, how much improvement can be expected. An additional ending criterion could be to monitor of fitness of the best individual change from generation to generation. If there is little or no improvement over a number of generations, the algorithm could terminate. According to the test results there exists no clear distinction between performance, generations and population size. For example, setting number 9 shows promising results for a large population size, while setting 10 and 11 seems to produce similar results with smaller population and fewer generations. What is certain is that population size and number of generations affect the average number of evaluations and the time the algorithm uses. This is also true for the crossover and mutation rate.

Progress Curves

Figure 6.10 and 6.11 shows progress curves for parameter setting 4 (with an increased number of generations) and 5 respectively. Since the genetic algorithm implemented here has elitism as part of the survival scheme, the current best individual in the algorithm will always be the same or better than the best individual in the last generation. For both graphs the best solution becomes stable quite early. This suggests that the number of generations can be kept at about 30-50 without sacrificing the solution quality.

6.5 Ant Colony Optimization

For an introduction to the ant colony optimization, see chapter 2.12. What follows is a mapping of the problem to the current problem instance. The article by Dorigo et al has been used for inspiration. [18]

6.5.1 Applying ACO to the TSP

As described earlier the tool path problem has in this thesis been considered as a TSP. The different parts of cuts or printing (active paths/cities) are considered as cities. Thus the cuts (cities) have four distances between each other (inactive paths).

The way ant colony optimization works is to allow artificial ants to traverse the graph, which is the cities and the distances between them, and when an ant is in a city it chooses the next edge based on uniform randomness. However, how likely an edge is to be chosen depends on the length of the edge and the amount of pheromone on the edge. After an iteration (generation) of ants, the pheromone values are updated. The edges that have been traversed will have more pheromone than those that were not. Thus, the edges traversed have a slightly higher chance of being chosen the next time.

To determine when to terminate the algorithm a specific number of iterations is often used. This can be used together with a length criterion to stop the algorithm when the current best solution is good enough. This, however, requires knowledge about the specific problem instance. Since the algorithm is initialized with all the edges having the same amount of pheromone, the initial runs are more random.

There are several ways of implementing the ant colony optimization algorithm. In this thesis, the Ant System has been implemented. It is one of the first implementations of the ant colony optimization algorithms presented by Dorigo in 1991.[18] More successful implementations have later been implemented. The details of the Ant System algorithm are now presented.

Pheromone Update

The pheromone is updated at the end of each generation according to equation 6.3.[18, p 5]

$$\tau_{ijn} \leftarrow (1-\rho)\cdot\tau_{ijn} + \sum_{k=1}^{m}\Delta\tau_{ijn}^{k} \tag{6.3}$$

The pheromone concentration between paths/cities i and j, along edge n (0, 1, 2 or 3 - start-start, start-end, end-start, end-end) is given by τ_{ijn}. In this equation, ρ is the evaporation rate, m is the number of ants and $\Delta\tau_{ijn}^{k}$ is the amount of pheromone deposited on the edge by ant k according to equation 6.4.[18, p 5]

$$\Delta\tau_{ijn}^{k} = \begin{cases} Q/L_k & \text{if ant } k \text{ used edge } (i,j) \text{ in its tour,} \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

Q is a parameter decided by the user and L_k is the length of ant k's tour.

Ants' decision rule

When an ant decides what edge to use, it does so according to the length of the edges and the relative pheromone concentration on them. However, this has a random element and the chance of choosing an edge is decided according to equation 6.5.[18, p 5]

$$p_{ijn}^{k} = \begin{cases} \frac{\tau_{ijn}^{\alpha} \cdot \eta_{ijn}^{\beta}}{\sum_{c_{iln} \in \mathbf{N}(s^p)} \tau_{iln}^{\alpha} \cdot \eta_{iln}^{\beta}} & \text{if } c_{il} \in \mathbf{N}(s^p), \\ 0 & \text{otherwise} \end{cases} \tag{6.5}$$

Here, p_{ijn}^{k} is the probability of ant k using the edge n from i to j, when the partial solution s^p has been made. $\mathbf{N}(s^p)$ is the set of feasible edges that have not been used by ant k and do not take ant k to a path/city it has been before, η_{ijn} is the pheromone concentration of the edge n from i to j and η_{ijn} is given by equation 6.6.[18, p 5] The length of the edge is represented by d_{ijn}. The importance of pheromone concentration versus the length of the edge is decided with the parameters α and β.

$$\eta_{ijn} = \frac{1}{d_{ijn}} \tag{6.6}$$

6.5.2 Results for the Ant System

While ACO is not strictly an evolutionary algorithm, it shares many of the characteristics and will be tested the same way as the genetic algorithm. The ACO algorithms was implemented as described above and tested with the presented test data. Table 6.4 shows the test results for different parameter settings and their results, the details of the settings can be found in table 6.3. The results consist of the best result, the best result as a percentage of the original path, the mean best fitness, worst result and the average number of evaluations to a solution. In this Ant System algorithm there are 6 parameters. This together with the stochastic nature of ACO algorithms makes it hard to determine what the ideal parameters are and how the parameters affect the final result. The testing presented here do not try to map the entire effect the parameters have on the performance of the algorithm, but rather to present some results with some different parameters and discuss them to get a better impression of how the algorithm work. Progress curves are presented for some of the tests, this can be seen in figures 6.13, 6.14 and 6.15.

Parameter setting	1	2	3	4	5	6	7
Ants	10	10	10	10	10	10	10
Generations	50	50	50	50	50	50	50
α	1.0	1.0	1.0	0.5	1.0	1.0	1.0
β	1.0	1.0	1.0	1.0	0.1	1.0	1.0
Q	1.0	1.0	1.0	1.0	1.0	1.0	1.0
ρ	0.7	0.7	0.7	0.7	0.7	0.2	0.9
Runs	10	100	200	10	10	10	10
	8	9	10	11	12	13	14
Ants	10	50	50	5	10	10	10
Generations	50	50	15	500	10	30	30
α	1.0	1.0	1.0	1.0	1.0	0.3	1.0
β	1.0	1.0	1.0	1.0	1.0	0.45	1.0
Q	0.5	1.0	1.0	1.0	1.0	0.64	1.0
ρ	0.7	0.7	0.7	0.7	0.5	0.7	0.7
Runs	10	10	10	10	10	10	50

Table 6.3: Parameter settings for the different tests. Test results can be seen in table 6.4.

Parameter setting	1	2	3	4	5	6	7
Best ever	99.7	88.9	88.9	99.0	115.0	100.4	99.4
Percentage of original inactive path	37.0	33.0	33.0	36.7	42.6	37.2	36.8
Total new length percentage of total original length	93.9	93.5	93.5	93.9	94.4	93.9	93.9
Worst ever run	119.0	139.0	132.2	122.5	192.7	112.6	122.8
Mean best	107.7	110.7	109.8	109.8	152.9	108.5	111.4
Average number of evaluations	500	500	500	500	500	500	500

	8	9	10	11	12	13	14
Best ever	99.2	99.1	99.8	100.5	109.1	100.2	99.1
Percentage of original inactive path	36.8	36.7	37.0	37.2	40.4	37.1	36.7
Total new length percentage of total original length	93.9	93.9	93.9	93.9	94.2	93.9	93.9
Worst ever run	126.0	107.8	112.1	135.1	151.6	116.7	132.3
Mean best	111.7	101.5	105.5	120.1	136.4	107.2	110.3
Average number of evaluations	500	2500	750	500	300	360	360

Table 6.4: Test results for different parameter settings for the ant colony optimization algorithm. The values of the parameter can be seen in table 6.3. The orignal length of the inactive path is 269.8 mm. Total original length with active and inactive paths is 2778.7 mm.

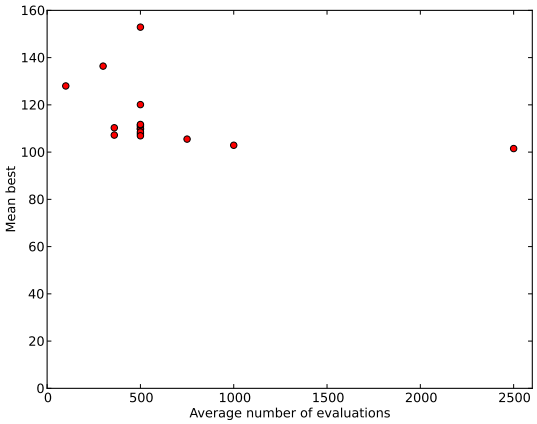


Figure 6.12: The Mean best vs the Average number of evaluations for the Ant System. Some of the results presented here is not included in table 6.3 and 6.4.

6.5.3 Discussion on ACO test results

Similarly to the discussion on the test results for the genetic algorithm, the goal of this discussion is to highlight different aspects of the Ant System ACO algorithm and discuss its applicability in the current context. The number of parameter settings has been limited to 14 to not drown in data. However, this means that not all aspects of changing parameters are highlighted. Another important aspect is that the algorithm is stochastic and can yield different results at different times. A large deviation of the test results is highly unlikely because the algorithm is run several times. A comparison with the genetic algorithm will be performed after Ant System has been discussed.

General

As noted earlier the speed of the algorithm is not the most important criterion. The end results should be as good as possible, preferably the global best. It seems that a global best is accomplished with parameter setting 2 and 3. However, these take a long time and require 50000 and 100000 evaluations. Longer runs or a larger population of ants do not seem to make any difference as figure 6.14 and 6.12 shows.

Similar to the genetic algorithm, the algorithm itself will not be affected by implementing collision control or by making sure the active paths are in the right order as this has to do with the data processing before the algorithm starts to work on the data.

Number of Runs

The test results for parameter setting 1, 2, 3, 13 and 14 in table 6.4 shows how the algorithm handles many runs. When increased 10 and 20 times, the best results improves considerably, about 10% better than the best result for 10 runs and 5% less percentage of the original length. For parameter setting 13 and 14

this is not as evident as the best result is hardly improved when the number of runs is increased 5 times. While it is expected that the best ever results improves with more runs, the mean best and the worst ever run is expected to be worse. It is interesting to see that for both settings the mean best result do not increase significantly and from 100 to 200 (setting 2 to setting 3), the mean best actually decreases. For setting 1 the mean best is about 8 mm more than the best ever result for 10 runs and abut 20 mm more for setting 2 and 3 (100 and 200 runs). For setting 13 and 14, the difference is 7 mm for 10 runs and about 11 mm for 50 runs. The worst ever run increases with about 15 mm to 20 mm from setting 1 to 2 and 3 and from 13 to 14. Yet again, the worst ever run was better for setting 3 than for 2. These results indicate that the Ant System algorithm will have a quite stable performance over time.

Alpha and beta

Parameter settings 4 and 5 show the effect of decreasing the importance of pheromone concentration and the length of the edge respectively. Parameter setting 12 shows how the algorithm is affected when α and β has been given random values together with Q. For all the other parameter settings, the two has been equally important. From the test results it is quite evident that a dramatic skew in the relation between the pheromone concentration and path length in the favor of pheromone concentration will deteriorate the results. This can be seen in the results for parameter setting 5 where pheromone concentration is 10 times more important that the length of the edge. The mean best result and the worst ever run will increase dramatically in length together with an increase of length for the best ever result. Similar results might be expected when the length is 10 times more important than pheromone concentration. In setting 4, the length of the edge is twice as important as the pheromone concentration. Setting 13 is used as a more random setting and is not so easy to compare to the other settings as it has two more parameters changed (the number of generations is decreased together with and Q). It shows a decrease in performance.

Rho

Rho (ρ) is the evaporation rate of the pheromone concentration. When the evaporation rate is close to 0, the algorithm will not remember the choices that ants have made so well and the algorithm will become more random. The generation before the current will have a much larger impact than the former generations. On the other hand, close to 1, the ants will quickly be more affected by the previous generations. The test results for setting 6 and 7 show that an increase and a drastic decrease in ρ from the standard (in these tests) 0.7 do not affect the end result significantly.

Q

The larger Q is, the more the length of an ant's tour affects the pheromone update of a used edge compared to the number of ants that have traversed the edge and the evaporation rate. Parameter setting 8 halves the value for Q without changing the results much.

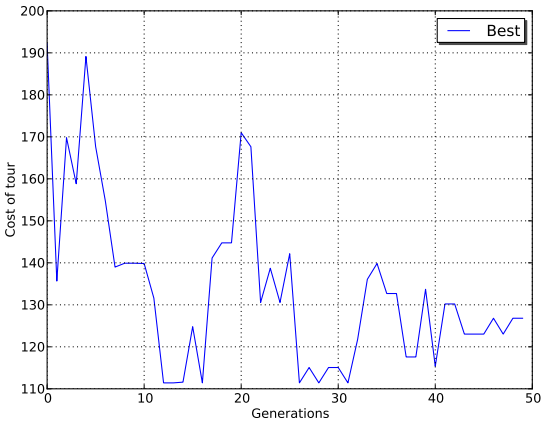


Figure 6.13: The best individual for each generation using parameter setting 1 in table 6.3(ant system)

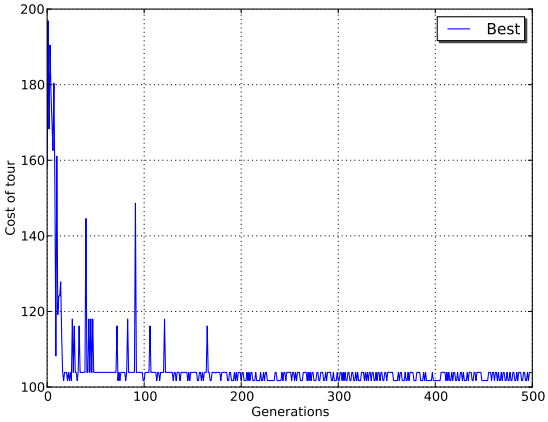


Figure 6.14: The best individual for each generation using parameter setting 1 in table 6.3 and 500 generations(ant system)

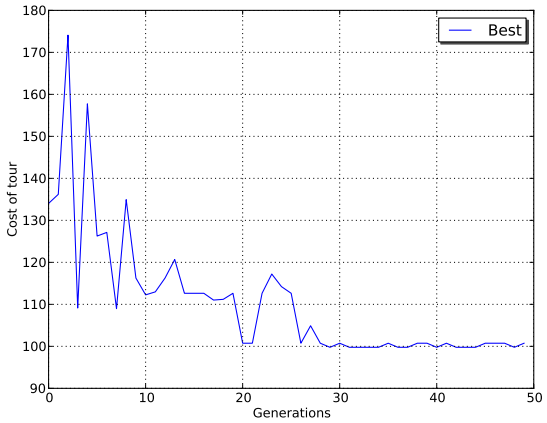


Figure 6.15: The best individual for each generation using parameter setting 1 in table 6.3 and 50 ants(ant system)

Ants and number of generations

The number of ants and the number of generations affect the number of evaluations as each ant is evaluated for each generation. Parameter setting 9 uses 5 times the number of ants without acquiring a better best solution than the *standard* parameter settings 2 and 3. However, the worst ever run and the mean best both becomes about 10 mm shorter. The same tendency is created by maintaining the number of ants and reducing the number of generations by 35, to 15 (setting 10). Parameter setting 11, 5 ants and 100 generations, gives a worse mean best and worst ever run than settings 1, 2 and 3.

Progress Curves

Two progress curves with parameter setting 1 are presented in figure 6.13 and 6.14. Figure 6.13 is more detailed, with 50 generations and figure 6.14 has 500 generations. Figure 6.14 shows that it is quite pointless to use more than 200 generations of ants. 50 to 100 generations seems like a more ideal number of generations for this parameter setting. Figure 6.13 shows that 50 generations might lead to less optimal results, so about 100 generations seems to be the most promising number. Increasing the number of ants to 50 (figure 6.15) will decrease the number of generations required for a good result, but will not do much to increase the efficiency of the algorithm.

6.6 Concuding discussion on Tool path-optimization

6.6.1 Comparing genetic algorithm and ant colony system

The test results clearly show that the ant system is a superior algorithm to the genetic algorithm for the TSP. The best result for the Ant System uses less evaluations, finds a shorter best solution, worst solution and mean solution. However, it is important to note that the number of evaluations, do not directly

correspond to the amount of time the algorithm use. In fact, the genetic algorithm was faster per evaluation than the Ant System algorithm under testing. Furthermore, the algorithms used are not the best in their categories and therefore should not be regarded as an absolute result. Having said this, the fact that the ant system found the shortest routes among the two, and it has been reported used in the industry in similar applications,[18, p 7] suggest that it is the preferred solution.

6.6.2 General discussion

The main point of this chapter is not to perform an in depth comparison of the two algorithms, but to show to what degree they are applicable to the problem instance. This has been briefly discussed for the two different algorithms, focusing on the required improvement needed to be used as a real application. The conclusion is that the algorithms and especially the ant system (or one of the other, better, versions) can be used for this instance. However, when adding the active path length and comparing the lengths to the original length the path do not become much shorter. The best result gives a saving of 6.5% of the original length, or a distance of 180 mm. With a speed of 24 mm/s this results in only an improvement in machining time of 7.5 seconds. This seems like a very low number and would suggest that the algorithm is unecessary. However, it is important to consider that the test data is an inscription of a design, which is a quite short path compared to the typical CNC milling path and 3D milling path. Even when considering this, the fact that the time saved in the test was so short it puts the algorithm in a defensive position, it needs to prove its applicability. It is important to stress that there is nothing wrong with the algorithm in itself, the issue is that it might not be worth applying it considering the results it can possibly give.

The two simplifications presented in the individual discussions need further investigation. The first, the order of the active parts, can be solved by investigating the data from the G-code before the path-optimization algorithm is activated. As path analysis and similar subjects has not been investigated in this thesis, it is not possible to confirm, but this can be most likely be solved quite easily. However, the length of the inactive path will most likely not be reduced as much as the results presented here shows. For example, for 3D printing, it is possibile that the length reduction might be very small as the order of the inactive paths is often very strict. The results might be better when dealing with CNC milling, as the order of the paths seems to be freer. For example by looking at the path in figure 5.5 and 5.6, it is possible to see that an improvement in path length is possible without violating the correct order of active paths. It is important to note that this analysis is done with limited knowledge of milling and 3D-prinitng paths and these should be studied further to verify what has been presented here.

The second improvement needed is collision avoidance. This is also a quite unexplored field in this thesis. Similarly to the order of the inactive paths, collision avoidance has to do with the data created from the G-code, and do not directly interfere with the path-optimization algorithm. Even with limited insight into the world of collision avoidance, it is possible to quite surely say that this might very well be more advanced and complex than deciding the possible appropriate orders of the active paths. Depending on the number of possible

inactive paths, it might prove too time consuming to analyze all the paths in order to deny collisions and calculate the lengths. However, it might be so that not all the possible inactive paths will be explored in the path-optimization algorithm. To exploit this, the length of collision free inactive paths might be calculated while running the path-optimization algorithm. A study of how many of the possible inactive paths explored by the algorithms can show whether this exploitation is possible. It seems that the inclusion of collision avoidance can be one of the central problems of further development.

An issue, which has been mentioned, is that when these simplifications are improved, the path found by the algorithm might very well not be shortened as much as the current test results suggests. This is also affected by the type of path to be improved (different G-code files, different types of tool path) and it is very hard to determine the total effect the algorithms has before testing on many different paths.

To conclude, the current problem do not so much lie with the algorithms as the preparation of data for the algorithm and the possible length that can be saved by analyzing the inactive paths. In this way, it is possible to say that the ant system algorithm and to some degree, the genetic algorithm, is applicable to the problem instance. However, the issue of preparing the data properly might prove to be too time consuming, making the algorithms efficiency futile. Further investigation of data preparation can reveal this. Another aspect is that the improvements might render the algorithm less efficient by denying much improvement in the inactive path length. There are still some hurdles that need to be jumped.

Chapter 7

Conclusion and proposals for further work

7.1 Conclusion

Working with this thesis has been an immense learning experience. The areas to study that are related to robotic manipulators are many and varied. This meant that not all the areas could be explored as much as desired. I have gained insight into the design issues for a Stewart platform, studied G-code and the mathematics behind the platform to create a simulator. These studies allowed me to analyze the properties of tool paths and discover how to improve the length of them. Other topics have also been to some extent studied, such as the travelling salesman problem and biologically inspired computing. The different approaches of design, simulation and path-optimization really allowed me to understand the problems and the Stewart platform better. The thesis allowed more than anything to get to know the Stewart platform in detail.

What follows is a short conclusion of the three aspects of this thesis. These are the design, the simulation and the path-optimization using biologically inspired computing. Then the experience from the three parts is put together to evaluate the possibility of using the Stewart platform as a rapid prototyper alternative to the current RepRap design (Mendel) that can both perform 3D printing and CNC milling. There are several aspects to this project, for example the issues regarding implementing the system to test it in a practical manner. Another aspect is studying areas of research that are related to the system, such as workspace analysis.

Design The design chapter explored how a rapid prototyper with 3D printing and CNC milling capabilities could be physically implemented as a Stewart platform, looked at issues such as workspace limitations and the design of an actuator and compared the design with the RepRap Mendel. Further studies and practical testing is needed to verify whether the presented design will contain the presented characteristics. The design that shows the most interesting capabilities is the third experimental design. A figure of this design can be seen at page 45 (figure 4.19). Goal number 1 as presented in chapter 1.2 has thus been achieved. No future problems were found too large or complex to be

solved, and thus further study and testing of the design is encouraged.

Simulator The simulator has capabilities to perform simple analysis of how the Stewart platform behaves according to a G-code file. The path can also be studied to be verified. Due to the fact that there is no standard G-code syntax and in order to make a working version of the manipulator, not all types of G-code files and commands were implemented. Further development in these fields, especially the amount of commands handled, is needed. To do this, the style of the G-code needs to be decided. The creation of the simulator enabled me to better understand the Stewart platform, CNC-milling and 3D printing. In this way, goal number 2 as presented in chapter 1.2 has been achieved.

Biologically inspired path-optimization Both the genetic algorithm and the ACO algorithm managed to improve the inactive parts of the tool path to less than half the length of the original inactive path lengths. The ACO algorithm gave consistently better results. Since there were made two very important simplifications, these have to be studied before the algorithm actually can be applied on the problem instance. The two simplifications lies in how the data is treated before entering the algorithm, therefore the current challenge is not in the algorithm, but in the preparation of data. Improving the simplifications can also make the algorithms less efficient by denying shorter paths. Maybe the most important aspect of this study is that the best result only decreases the machining time with 7.5 seconds when the fast speed is set to 24 mm/s. The tool path-optimization problem has thusly been investigated and algorithms have been implemented to solve it as described as goal 3 in chapter 1.2.

The Stewart Platofrm as a Rapid Prototyper The overlying goal was to investigate whether a robotic manipulator could be an alternative to the Mendel RepRap with improved functionality. At the present stage, the study of the Stewart platform can not dismiss its use as a rapid prototyper and RepRap alternative. The final conclusion is that there are several positive indications that the Stewart platform can be a very good alternative to the RepRap. But there are some problems (for example speed) that might deny such a possibility. The problems have to be solved and the positive indications tested further in order to confirm the possibility to use the Stewart platform as a rapid prototyper.

7.2 Further Work

Further work is suggestions for research topics that became evident during the work with this thesis. Since the area of study has been so wide, many different topics can be studied and therefore the number of topics presented here do not include everything that came up while working on this master thesis.

Avoid singularities and workspace Singularities for the Stewart platform are positions and orientations where the manipulator gains one degree of freedom and becomes uncontrollable. A complete description and characterization of the singularities would be to parameterize the entire singularity hyper-surface(s)

in the task-space (6D in the case of the Stewart platform).[14] Then, all the singularities can be analytically identified. This is however extremely difficult and Bhaskar Dasgupta and T.S. Mruthyunjaya had in 1998 not seen any work on work on this topic.

They had however found some practical work on the issue. Bhattacharya et al. [14] have looked at paths in vicinity of a singularity and Dasgupta and Mruthyunjaya themselves have studied singularity avoidance between two positions. Later researches have also been performed, such as Qimi Jiang and Clment M. Gosselin's study from 2009.[26] Singularity avoidance is important for controlling the Stewart platform to avoid uncontrollable situations. Further study of the theoretical and analytical solutions is needed. Another issue is to implement current solutions of singularity avoidance on the system presented in this thesis. Singularities are tightly connected with the workspace as singular positions are positions that limit the workspace. Studying the workspace of a Stewart platform is also a possible research topic.

Controlling the Stewart Platform In addition to avoiding singularities, avoiding collisions is important. There are two aspects of collision avoidance. The typical one is to avoid colliding with objects in the vicinity of the manipulator (such as the part that the machine is milling on). And the second on is to avoid colliding with the manipulators own parts as discussed in chapter 4.5.1. As collision avoidance is outside the scope of this thesis, it's details has not been studied. Studies relating to collision control of the Stewart platform include [11], while for a study of a more general collision control of robotics, see for example [28], [58, ch. 5] or [12, ch. 7]. Implementing this theory on the presented Stewart platform can be interesting. Taking this a step further would be to study the extra complexity in avoiding collisions due to experimental design number 3. Also needed for controlling the manipulator is a G-code parser that understand most of or all the G-code commands.

Path-optimization This has already been somewhat discussed in the final conclusion. Collision avoidance, singularity avoidance and correct order of path need to be treated in order for the path-optimization algorithms to be applicable. Other aspects that can be studied are the possibility of dividing active paths in order to shorten the total path length even further. Furthermore, testing the algorithms on more G-code files should be done to find out whether a considerable amount of time can be saved by using this type of path-optimization. In order to do this conscientiously, a study and identification of different types of G-code paths should be performed. Even more important is to study several G-code files to investigate the lengths of the inactive paths. This should be done to find out whether applicating path-optimization on the inactive paths would in any case save a considerable amount of machining time. Another topic that was not found in the literature is to compare the biologically inspired path-optimization algorithms with non-biological alternatives. Yet an even more interesting option could be to combine the studies done with traditional algorithms and biologically inspired algorithms.

Building, testing and simulation One of the long term goals initiated by this thesis is to create an alternative to the RepRap. In order to do this, the

Stewart platform needs to be built and tested. An interesting test would be to find out how many percentage of itself the manipulator can create. Another is to test the rigidity of the platform. Many of the issues that have been discussed above have to be implemented before this can be done, such as avoiding collisions and singularities and creating a control system that includes more G-commands. As the control system is closely related to the simulator, this also gives a chance to improve the simulator. Furthermore, the simulator can be improved in other ways, such as including the shape of the platform parts. Another two issues are testing the different parts for durability and finding out whether the experimental design is realizable.

Bibliography

[1] Arduino homepage. http://www.arduino.cc/(2011-04-28).

[2] Charles Bates. Move over machine tools here come robots. http://www.americanmachinist.com/304/Issue/Article/False/13386/(2010-03-26), 02 2006.

[3] K.A. Berman and J.L. Paul. *Algorithms: sequential, parallel, and distributed*. Thomson/Course Technology, 2005.

[4] E. Bohez, S.S. Makhanov, and K. Sonthipermpoon. Adaptive nonlinear tool path optimization for five-axis machining. *International Journal of Production Research*, 38(17):4329–4343, 2000.

[5] I.A. Bonev and J. Ryu. A new approach to orientation workspace analysis of 6-DOF parallel manipulators. *Mechanism and Machine Theory*, 36(1):15–28, 2001.

[6] Adrian Bowyer. A self-copying manufacturing process. http://www.reprap.org/pub/Main/WebHome/one-page.pdf(2010-03-24).

[7] A. W. (editor) Burks and John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, Illinois, USA, 1966.
Von Neumann's work on self-reproducing automata, completed and edited after his death by Arthur Burks. Also includes transcripts of von Nuemann's 1949 University of Illinois lectures on the "Theory and Organization of Complicated Automata".

[8] Z. Car, B. Barisic, and M. Ikonic. GA based CNC turning center exploitation process parameters optimization. *Metalurgija*, 48(1):47–50, 2009.

[9] J.N. Chiasson. *Modeling and high performance control of electric machines*. IEEE Press series on power engineering. John Wiley, 2005.

[10] Commercial 3d printer used: Dimension 768 series. http://www.dimensionprinting.com/3d-printers/printing-productspecs768series.aspx(2011-04-30).

[11] J. Cortes and T. Simeon. Probabilistic motion planning for parallel mechanisms. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 3, pages 4354–4359. IEEE, 2003.

[12] J.J. Craig. *Introduction to robotics: mechanics and control*. Addison-Wesley series in electrical and computer engineering: control engineering. Pearson-/Prentice Hall, 2005.

[13] J. Daintith and Oxford University Press. *Oxford dictionary of computing*. Oxford paperback reference. Oxford University Press, 2004.

[14] B. Dasgupta and TS Mruthyunjaya. The Stewart platform manipulator: a review. *Mechanism and Machine Theory*, 35(1):15–40, 2000.

[15] M. Dehghani, M. Ahmadi, A. Khayatian, M. Eghtesad, and M. Farid. Neural network solution for forward kinematics problem of HEXA parallel robot. In *American Control Conference, 2008*, pages 4214–4219. IEEE, 2008.

[16] J.L. Deneubourg, S. Aron, S. Goss, and J.M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3(2):159–168, 1990.

[17] Document including rank formula on page 17. `http://www.aero.caltech.edu/~tamer/GATutorial.pdf`(2011-04-28).

[18] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28–39, 2006.

[19] A.E. Eiben and J.E. Smith. *Introduction to evolutionary computing*. Natural computing series. Springer, 2003.

[20] G-code used for testing. `http://www.cuttingedgecnc.com/g-codes.htm`(2011-04-28).

[21] D. Garagic and K. Srinivasan. Contouring control of stewart platform based machine tools. In *American Control Conference, 2004. Proceedings of the 2004*, volume 4, pages 3831–3838. IEEE, 2002.

[22] S. Goss, S. Aron, J.L. Deneubourg, and J.M. Pasteels. Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76(12):579–581, 1989.

[23] K. Harib and K. Srinivasan. Kinematic and dynamic analysis of Stewart platform-based machine tool structures. *Robotica*, 21(05):541–554, 2003.

[24] M. Honegger, A. Codourey, and E. Burdet. Adaptive control of the hexaglide, a 6 dof parallel manipulator. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 1, pages 543–548. IEEE, 1997.

[25] Interview: Reprap. `http://www.openbusiness.cc/2009/07/24/reprap-2/`(2010-03-26), 07 2009.

[26] Q. Jiang and C.M. Gosselin. Determination of the maximal singularity-free orientation workspace for the Gough-Stewart platform. *Mechanism and Machine Theory*, 44(6):1281–1293, 2009.

[27] A. Joneja, KW Pang, K.G. Murty, DCC Lam, and MF Yuen. A Genetic Algorithm for Path planning in rapid prototyping. In *Proceedings of the ASME DETC-DFM Conference*.

[28] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90, 1986.

[29] A.V. Korobeinikov and V.E. Turlapov. Modeling and evaluation of the Stewart platforms. , 9(3):279–286, 2006.

[30] A. Krimpenis, PIK Liakopoulos, KC Giannakoglou, and GC Vosniakos. Multi-objective design of optimal sculptured surface rough machining through Pareto and Nash techniques. *CD-proceedings EUROGEN*, 2005.

[31] Kuka robot- component milling to cad specifications - youtube. `http://www.youtube.com/watch?v=0Zg7wRf6XEE`(2010-04-29).

[32] T.Y. Lee and J.K. Shim. Algebraic elimination-based real-time forward kinematics of the 6-6 Stewart platform with planar base and platform. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1301–1306. IEEE, 2001.

[33] X. Li, S. Zhang, L. Song, L. Miao, and D. Liu. Research on application of NC program optimization based on TSP. In *Mechatronics and Automation, 2009. ICMA 2009. International Conference on*, pages 1493–1498. IEEE, 2009.

[34] List of cad and cam software producers. `http://en.wikipedia.org/wiki/List_of_CAD_companies`(2011-04-28).

[35] K. Liu, J.M. Fitzgerald, and F.L. Lewis. Kinematic analysis of a Stewart platform manipulator. *Industrial Electronics, IEEE Transactions on*, 40(2):282–293, 1993.

[36] Y. Lou, F. Feng, and M.Y. Wang. Trajectory planning and control of parallel manipulators. In *Control and Automation, 2009. ICCA 2009. IEEE International Conference on*, pages 1013–1018. IEEE, 2009.

[37] M. Lutz. *Programming Python*. O'Reilly Series. O'Reilly Media, 2011.

[38] Makerbot industries. robots that make things. `http://www.makerbot.com/`(2011-04-29).

[39] S. S. Makhanov, D. Batanov, E. Bohez, K. Sonthipaumpoon, W. Anotaipaiboon, and M. Tabucanon. On the tool-path optimization of a milling robot. *Computers and Industrial Engineering*, 43(3):455 – 472, 2002.

[40] Manual page for mendel in the reprap wiki. `http://reprap.org/wiki/Mendel_User_Manual:_Host_Software`(2011-04-28).

[41] P. Nanua, K.J. Waldron, and V. Murthy. Direct kinematic solution of a Stewart platform. *Robotics and Automation, IEEE Transactions on*, 6(4):438–444, 1990.

[42] Numerical control - wikipedia. `http://en.wikipedia.org/wiki/Numerical_control`(2010-04-28).

[43] G.E. O'DONNELL and G. BENNETT. Open Design and the Reprap Project. 2010.

[44] Vik Olliver. Construction of rapid prototyping testbeds using meccano. http://staff.bath.ac.uk/ensab/replicator/Downloads/MeccanoFDMfinal.pdf(2010-03-24), 04 2005.

[45] Picture and information about mendel - reprap. http://www.reprap.org/wiki/Mendel(2010-04-27).

[46] Picture of a parallel manipulator. http://www.parallemic.org/Material//Tsai.gif(2011-04-28).

[47] Picture of a serial manipulator. http://www.emeraldinsight.com/fig/0490350503015.png(2011-04-28).

[48] Picture of an universal joint. http://www.adf-safetytools.com/image/16_cardan.jpg(2011-04-28).

[49] Picture of cnc-milling machine. http://image.made-in-china.com/2f0j00jeKQNlUqAabB/CNC-Milling-Machine-Machining-Center-VBZ-1000-.jpg(2010-04-27).

[50] Pkm tricept homepage. http://www.pkmtricept.com/(2011-04-28).

[51] Reprap - the replicating rapid prototyper project. www.bath.ac.uk/idmrc/themes/projects/amps/AMPS-Project-RepRap.pdf(2010-03-24).

[52] Reprap inspired design for milling. http://cpwebste.blogspot.com/2010/05/hydra-mmm-prototype-finished.html(2011-04-28).

[53] Reprap inspired design for milling. http://www.3dreplicators.com/cgi-bin/cblog/index.php?/categories/6-Tommelise(2011-04-28).

[54] P. Smid. *CNC programming handbook: a comprehensive guide to practical CNC programming*. EngineeringPro collection. Industrial Press, 2003.

[55] Solidcam. http://www.solidcam.com(2011-04-28).

[56] Solidworks - homepage. http://www.solidworks.com/(2010-04-29).

[57] Specifications for the mendel reprap. http://reprap.org/wiki/Mendel(2011-04-28).

[58] M.W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot modeling and control*. John Wiley & Sons, 2006.

[59] Stepper motor system basics (rev. 5/2010), 2010.

[60] Typical stepper motor. http://www.sparkfun.com/products/9238(2011-04-28).

[61] Up! a personal 3d printer. http://pp3dp.com/l(2011-04-29).

[62] Video of stewart platform. http://www.youtube.com/watch?v=wwKucXHto0w&feature=related(2011-04-28).

[63] Video of stewart platform 2. http://www.youtube.com/watch?v=WVQ1SSXAcOs&NR=1&feature=fvwp(2011-04-28).

[64] W. Voss. *A Comprehensible Guide to Servo Motor Sizing*. Copperhill Media Corporation, 2007.

[65] P.K. Wah, K.G. Murty, A. Joneja, and L.C. Chiu. Tool path optimization in layered manufacturing. *Iie Transactions*, 34(4):335–347, 2002.

[66] S.M. Wang and K.F. Ehmann. Error model and accuracy analysis of a six-DOF Stewart platform. *Journal of manufacturing science and engineering*, 124:286, 2002.

[67] Y. Wang. A direct numerical solution to forward kinematics of general Stewart–Gough platforms. *Robotica*, 25(01):121–128, 2007.

[68] Patrick Waurzyniak. Shop-floor productivity. *Manufacturing Engineering*, 135(1), 7 2005. http://www.sme.org/cgi-bin/find-articles.pl?&ME05ART38&ME&20050710&&SME&#article(2010-04-28).

[69] Eldho Wilson. Replicating rapid prototyper (reprap). http://dspace.sngce.ac.in/bitstream/123456789/1511/1/ELDHO%20WILSON.pdf(2010-03-24).

[70] L. Yi. Computer-aided geometric machining of a 3D free surface using a 3-UPU spatial parallel machine tool. *The International Journal of Advanced Manufacturing Technology*, 26(9):1018–1025, 2005.

Appendix A

Code attachment

A.1 Simulation software

As described in chapter 5. The code has three classes: Simulator, Inverse Kinematic and PostProcessor. The last one is really a G-code parser.

```
from decimal import *
import mpl_toolkits.mplot3d.axes3d as p3
import matplotlib.pyplot as plt
import numpy as np

#from pylab import *#this is the top module of scipy
import math as ma
import images2swf as mo
import os
from PIL import Image

class Simulator(object):
    '''
    The simulator can create a picture of a path or create an
        animation file
    '''


    def __init__(self, max = 500, min = 50):
        '''
        Constructor
        '''
        self.commandlist = []
        self.max = max
        self.min = min

    '''
    For simple generation of a path for the endpoint.
    '''
    def loadsimple(self):
        self.commandlist = self.generatesimplepath()

    '''The simple path generator'''
    def generatesimplepath(self, max = Decimal('3')):
        #Position
        x = [Decimal('0')]
        y = [Decimal('0')]
        z = [Decimal('100')]
```

```
        #Orientation
        pitch = [0]
        roll = [0]
        yaw = [0]
        command = ['Start']
        plat = [[0,0,250]]
        for element in range(99):
            if element < 20:
                x.append(x[element] + max)
                y.append(y[element])
                z.append(z[element])
                plat.append([plat[-1][0]+max, plat[-1][1], plat
                    [-1][2]])
                pitch.append(pitch[element])
                roll.append(roll[element])
                yaw.append(yaw[element])
                command.append('Fast move')
            elif element < 40:
                x.append(x[element])
                y.append(y[element] + max)#element*max)
                z.append(z[element])
                plat.append([plat[-1][0], plat[-1][1]+max, plat
                    [-1][2]])
                pitch.append(pitch[element])
                roll.append(roll[element])
                yaw.append(yaw[element])
                command.append('Fast move')
            elif element < 60:
                x.append(x[element] - max)
                y.append(y[element] - max)
                z.append(z[element] + max)
                plat.append([plat[-1][0]-max, plat[-1][1]-max, plat
                    [-1][2]+max])
                pitch.append(pitch[element])
                roll.append(roll[element])
                yaw.append(yaw[element])
                command.append('Fast move')
            else:
                x.append(x[element])
                y.append(y[element] + max)
                z.append(z[element]- max)
                plat.append([plat[-1][0], plat[-1][1]+max, plat
                    [-1][2]-max])
                pitch.append(pitch[element])
                roll.append(roll[element])
                yaw.append(yaw[element])
                command.append('Fast move')

        return [x,y,z,pitch,roll,yaw,command,plat]

    '''
    Load a path from the "post-processor"/g-code parser
    '''
    def loadpath(self, max=0.15):
        pp = PostProcessor()#have to include name of file
        self.commandlist = pp.createcommandlist()
    '''
    Finds the absolute speed between two coordinates
    '''
    def getabsolutespeed(self, co, nco, time):
        return Decimal(str(ma.sqrt((nco[0]-co[0])**2 + (nco[1] - co
            [1])**2 +
```

```
95          (nco[2] - co[2])**2)))/Decimal(str(time))
        '''
97      Draws a path
        '''
99      def drawpath(self):
            fig = plt.figure(figsize = (5,5))
101         ax = fig.gca(projection='3d')
            for i in range(len(self.commandlist[0])):
103             self.commandlist[0][i] = float(self.commandlist[0][i])
                self.commandlist[1][i] = float(self.commandlist[1][i])
105             self.commandlist[2][i] = float(self.commandlist[2][i])

107         ax.plot(self.commandlist[0], self.commandlist[1], self.
                commandlist[2],
                    label='Milled path')
109         ax.legend()

111         plt.show()


113


        '''
115     the coordinates lies in commandlist
        time: the time between two coordinates
117     Creates an animation from a list of commands
        '''
119     def  animation(self, time = 0.1):
            files = []
121         xrange = range(-450,300,1)
            zrange = range(0,300,1)
123         mill = 'Off'
            ik = InverseKinematic()
125         frames = 99
            for i in range(frames):#range(len(self.x)):
127             fig = plt.figure(figsize = (8,10))
                ax = fig.add_subplot(2,1,1, projection='3d')
129             cm = [self.commandlist[0][i],self.commandlist[1][i],
                      self.commandlist[2][i],self.commandlist[3][i],
131                   self.commandlist[4][i],self.commandlist[5][i],
                      self.commandlist[6][i],self.commandlist[7][i]]
133             plat = ik.getplatimageco(cm[0:6])
                base = ik.getbaseimageco()
135             legs = ik.getimagelegs(base, plat)
                for j in range(len(plat)):
137                 for k in range(len(plat[j])):
                        plat[j][k] = float(plat[j][k])
139             for j in range(len(base)):
                    for k in range(len(base[j])):
141                     base[j][k] = float(base[j][k])
                for j in range(len(legs)):
143                 for k in range(len(legs[j])):
                        for l in range(len(legs[j][k])):
145                         legs[j][k][l] = float(legs[j][k][l])
                co = [self.commandlist[0][i],self.commandlist[1][i],
147                 self.commandlist[2][i]]
                ori = [self.commandlist[3][i],self.commandlist[4][i],
                    self.commandlist[5][i]]
149             if(i < range(len(self.commandlist[0]))):
                    nco = [self.commandlist[0][i+1],self.commandlist
                        [1][i+1],
151                        self.commandlist[2][i+1]]
                    speed = ik.getlegvelocity(co, nco, time, self.max,
```

```
                    self.min)
153
                aspeed = self.getabsolutespeed(co,nco,time)
155             if self.commandlist[6][i] == 'Spindle on (clockwise)'
                    or \
                self.commandlist[6][i] == 'Spindle on (counterclockwise
                    )':
157                 mill = 'On'
                elif self.commandlist[6][i] == "Milling off":
159                 mill = 'Off'
                vel=0
161             if self.commandlist[6][i] == 'Fast move':
                    vel = 30
163             elif self.commandlist[6][i] == 'Slow move':
                    vel = 15
165
                ax.set_xlabel('X')
167             ax.set_ylabel('Y')
                ax.set_zlabel('Z')
169             ax.plot(plat[0],plat[1],plat[2], color = 'blue')
                ax.plot(base[0],base[1],base[2], color = 'blue')
171


173
                if type(self.commandlist[6][i]) == 'list':
175                 if self.command[6][i][0] == 'Tool height offset':
                        ik.toffs = self.command[6][i][1]
177
                ax.plot([float(self.commandlist[0][i]), float(self.
                    commandlist[7][i][0])],
179                             [float(self.commandlist[1][i]), float(
                                    self.commandlist[7][i][1])],
                                [float(self.commandlist[2][i]),
181                              float(self.commandlist[7][i][2])], color
                                    = 'blue')
                for j in range(len(legs[0])):
183                 ax.plot(legs[0][j],legs[1][j],legs[2][j], color = '
                        blue')
                ax.auto_scale_xyz(xrange,xrange,zrange)
185
                ax = fig.add_subplot(2,1,2)
187             ax.text(0.1, 12, 'Command: '        + str(self.
                    commandlist[6][i]))
                ax.text(0.1, 11,'Sat speed: '       + str(vel) + ' mm/s
                    ')
189             ax.text(0.1, 10, 'Absolute speed: ' + str(aspeed) + '
                    mm/s')
                ax.text(0.1, 9, 'Orientation: ' + str(ori))
191             ax.text(0.1, 8, 'Tool: '     + mill)
                ax.text(0.1, 7, 'Tool offset: ' + str(ik.toffs) + ' mm'
                    )
193             ax.text(0.1, 6, 'Position: ')
                ax.text(0.1, 5, 'X: '    + str(float(co[0])))
195             ax.text(0.1, 4, 'Y: '    + str(float(co[1])))
                ax.text(0.1, 3, 'Z: '    + str(float(co[2])))
197             ax.text(5, 7, 'Leg 1, speed: ' + str(speed[0]) +' mm/s'
                    )
                ax.text(5, 6, 'Leg 2, speed: ' + str(speed[1]) +' mm/s'
                    )
199             ax.text(5, 5, 'Leg 3, speed: ' + str(speed[2])+ ' mm/s'
                    )
                ax.text(5, 4, 'Leg 4, speed: '  + str(speed[3]) +' mm/s
```

```
                ')
201             ax.text(5, 3, 'Leg 5, speed: ' + str(speed[4]) +' mm/s
                    ')
                ax.text(5, 2, 'Leg 6, speed: ' + str(speed[5])+ ' mm/s
                    ')
203             plt.xticks(range(0,16,15))
                plt.yticks(range(0,16,15))
205
                fname = '_tmp%03d.png'%i
207             fig.savefig(fname)
                files.append(Image.open(fname))
209             plt.close('all')
                print i
211         mo.writeSwf('test.swf',files)#can add duration of each
                image (1/fps) to decide
            #fps: default is 0.1
213         path = os.getcwd()
            for i in range(frames):#range(len(self.x)):
215             fname = '\_tmp%03d.png'%i
                fdel = path + fname
217             os.remove(fdel)
            print 'Done!'
219

221 class InverseKinematic(object):
        '''
223     A toolbox for inverse kinematics
        '''
225     def __init__(self, tlength=150, a=400,c=250,b=400,d=250):
            self.a = a
227         self.c = c
            self.b = b
229         self.d = d
            self.toffs = 0
231         self.tlength = tlength
        '''
233     Returns the coordinates see from base
        '''
235     def getcoord(self, co, partzero):
            useco = co
237         useco[2] = useco[2]
            a1 = self.ht(useco)
239         a2 = self.ht(partzero)
            m = np.dot(a2,a1)#end-effector end seen from base
241         platcenter = co
            platcenter[2] = platcenter[2] + self.tlength + self.toffs
243         a3 = self.ht(platcenter)
            cfromb = np.dot(a2,a3)#center of platform seen from base
245

247         return [m[0][3], m[1][3], m[2][3]], [cfromb[0][3],cfromb
                [1][3],cfromb[2][3]]

249     '''
        Returns the center of the platform (given by local coordinates)
             seen from base.
251     '''
        def getplatcenterfrombaseco(self, co, m):
253         a1 = self.ht(co)
            nil = Decimal(0)
255         one = Decimal(1)
            a2 = [[one, nil, nil, nil],
```

```
257                 [nil, one, nil, nil],
                    [nil, nil, one, Decimal(str(self.tlength))],
259                 [nil, nil, nil, one]]
            trans = np.dot(a1,a2)
261         return [trans[0][3], trans[1][3], trans[2][3]]

263     '''
        From the tool tip to the top platform
265     '''
        def htbaseplat(self, co):
267         a1 = self.ht(co)
            nil = Decimal(0)
269         one = Decimal(1)
            a2 = [[one, nil, nil, nil],
271                 [nil, one, nil, nil],
                    [nil, nil, one, Decimal(str(self.tlength))],
273                 [nil, nil, nil, one]]

275         return np.dot(a1,a2)

277     #the location of the tool-tip with offset
        def gettooltipwoffco(self, co):
279         a1 = self.ht(co)
            nil = Decimal(0)
281         one = Decimal(1)
            a2 = [[one, nil, nil, nil],
283                 [nil, one, nil, nil],
                    [nil, nil, one, Decimal(str(-self.toffs))],
285                 [nil, nil, nil, one]]
            trans = np.dot(a1,a2)
287         return [trans[0][3], trans[1][3], trans[2][3]]

289     '''
        Gets the length the legs have to move given two coordinates.
            Intended to be used
291     for programming the Arduino
        '''
293     def getardlegdiff(self, co, nco, partzero, max, min):
            #bs,coor = self.getcoord(co, partzero)
295         nowlegs = self.getleglengths(co)
            bs,ncoor = self.getcoord(co, partzero, max, min)
297         nextlegs = self.getleglengths(ncoor)
            difference = []
299         absdiff = []
            for i in range(len(nowlegs)):
301             if nowlegs[i] == False or nextlegs[i] == False:
                    return False, False
303         for i in range(len(nowlegs)):
                difference.append(nextlegs[i] - nowlegs[i])
305             absdiff.append(ma.sqrt(difference[i]**2))
            maxdiffleg = 0
307         for i in range(1, len(nowlegs)):
                if absdiff[i] > absdiff[maxdiffleg]:
309                 maxdiffleg = i
            return difference, maxdiffleg
311
        '''
313     Finds the velocity for the legs
        '''
315     def getlegvelocity(self, co, nco, time, max, min):
            #self.setcoordinates(co)
317         #next = InverseKinematic(nco[0],nco[1],nco[2],0,0,0)
```

```
        leglengthsnow  = self.getleglengths(co, max, min)#a list
            with six entries,
        #each entry has one value
        leglengthsnext = self.getleglengths(nco, max, min)
        speeds = []
        for i in range(6):  #six legs
            difference = leglengthsnext[i] - leglengthsnow[i]
            speeds.append(difference/time)
        return speeds

    '''
    Checks the velocity for the legs
    '''

    def checkvelocity(self, co, nco, maxdist, max, min):
        #self.setcoordinates(co)
        #next = InverseKinematic(nco[0],nco[1],nco[2],0,0,0)
        leglengthsnow = self.getleglengths(co, max, min)#a list
            with six entries,
        #each entry has one value
        leglengthsnext = self.getleglengths(nco, max, min)
        for i in range(len(leglengthsnow)):
            if leglengthsnow[i] == False or leglengthsnext[i] ==
                False:
                return False
        error = []
        for i in range(6):
            difference = ma.sqrt((leglengthsnext[i] - leglengthsnow
                [i])**2)
            if difference > maxdist:
                error.append(i+1)
        if len(error) > 0:
            print 'uiuiuiuiui too fast!!'
            return error
        else:
            return [0]
        #return speeds

    '''
    This and getplatimageco has the first location added to the
        back
    Returns the position of the base (/platform) seen from the base
        . In each sublist there
    are the x,y and z coordinates for the six legs.
    '''
    def getbaseimageco(self):
        Bx = [Decimal(str(ma.sqrt(3)/4*self.b)),Decimal(str(-ma.
            sqrt(3)/4*self.b)),
              Decimal(str(-ma.sqrt(3)/2*(self.b/2+self.d))),
              Decimal(str(-ma.sqrt(3)/2*(self.b/2+self.d))),Decimal
                  (str(-ma.sqrt(3)/4*self.b)),
              Decimal(str(ma.sqrt(3)/4*self.b)),Decimal(str(ma.sqrt
                  (3)/4*self.b))]
        By = [Decimal(str(self.d/2)),Decimal(str((self.b+self.d)/2)
            ),Decimal(str(self.b/2)),
              Decimal(str(-self.b/2)),Decimal(str(-(self.b+self.d)
                  /2)),Decimal(str(-self.d/2)),
              Decimal(str(self.d/2))]
        Bz = [0,0,0,0,0,0]
        return [Bx,By,Bz]

    '''
```

```
    '''Get the coordinstes for the platform to be used by drawn
    '''
    def getplatimageco(self, co):
        platco = self.getplatjointco()
        ht = self.htbaseplat(co)
        imgco = [[],[],[]]

        #for each corner of the platform
        for element in platco:
            transpo = np.dot(ht,element)
            imgco[0].append(transpo[0][0])#x
            imgco[1].append(transpo[1][0])#y
            imgco[2].append(transpo[2][0])#z
        imgco[0].append(imgco[0][0])
        imgco[1].append(imgco[1][0])
        imgco[2].append(imgco[2][0])
        return imgco
    '''
    Gives the locations of the attachment points from and for the
    center of the base (/platform)
    '''
    def getbasejointco(self):
        B1 = [[ma.sqrt(3)/4*self.b],[self.d/2],[0]]
        B2 = [[-ma.sqrt(3)/4*self.b],[(self.b+self.d)/2],[0]]
        B3 = [[-ma.sqrt(3)/2*(self.b/2+self.d)],[self.b/2],[0]]
        B4 = [[-ma.sqrt(3)/2*(self.b/2+self.d)],[-self.b/2],[0]]
        B5 = [[-ma.sqrt(3)/4*self.b],[-(self.b+self.d)/2],[0]]
        B6 = [[ma.sqrt(3)/4*self.b],[-self.d/2],[0]]
        return [B1,B2,B3,B4,B5,B6]

    def getlocplatjointco(self):
        T1 = np.array([[ma.sqrt(3)/2*self.a/2],[self.a/2],[0]])
        T2 = np.array([[ma.sqrt(3)/2*(self.a/2-self.c)],[(self.a+
            self.c)/2],[0]])
        T3 = np.array([[-ma.sqrt(3)/2*(self.a/2+self.c)],[self.c
            /2],[0]])
        T4 = np.array([[-ma.sqrt(3)/2*(self.a/2+self.c)],[-self.c
            /2],[0]])
        T5 = np.array([[ma.sqrt(3)/2*(self.a/2-self.c)],[-(self.a+
            self.c)/2],[0]])
        T6 = np.array([[ma.sqrt(3)/2*self.a/2],[-self.a/2],[0]])
        return np.array([T1,T2,T3,T4,T5,T6])

    #Local coordinates of the corners
    def getplatjointco(self):
        T1 = [[Decimal(str(ma.sqrt(3)/2*self.a/2))],[Decimal(str(
            self.a/2))],[0],[1]]
        T2 = [[Decimal(str(ma.sqrt(3)/2*(self.a/2-self.c)))],
              [Decimal(str((self.a+self.c)/2))],[0],[1]]
        T3 = [[Decimal(str(-ma.sqrt(3)/2*(self.a/2+self.c)))],
              [Decimal(str(self.c/2))],[0],[1]]
        T4 = [[Decimal(str(-ma.sqrt(3)/2*(self.a/2+self.c)))],
              [Decimal(str(-self.c/2))],[0],[1]]
        T5 = [[Decimal(str(ma.sqrt(3)/2*(self.a/2-self.c)))],
              [Decimal(str(-(self.a+self.c)/2))],[0],[1]]
        T6 = [[Decimal(str(ma.sqrt(3)/2*self.a/2))],[Decimal(str(-
            self.a/2))],[0],[1]]
        return [T1,T2,T3,T4,T5,T6]

    '''
    Coordinates for legs used in drawing images
    '''
```

```
425     def getimagelegs(self, base=0,plat=0):
            if base ==0 or plat ==0:
427             base = self.getbaseimageco()
                plat = self.getplatimageco()
429         imagelegs = [[],[],[]]
            for i in range(6):
431             imagelegs[0].append([base[0][i],plat[0][i]])
                imagelegs[1].append([base[1][i],plat[1][i]])
433             imagelegs[2].append([base[2][i],plat[2][i]])
            return imagelegs
435
        '''
437     The length of the leg
        '''
439     def getleglengths(self,co, max, min):
            base = self.getbasejointco()
441         plat = self.getplatjointco()
            leglengths = []
443         for i in range(len(base)):
                leglengths.append(self.calcleglengths(plat[i], base[i],
                    co, max, min))
445
            return leglengths
447     '''
        Finds the length of the legs
449     '''
        def calcleglengths(self, vectop, vecbase, co, max, min):
451         homog = self.ht([co[0],co[1],co[2],0,0,0])
            for i in range(len(vectop)):
453             vectop[i] = Decimal(str(vectop[i][0]))
                if i < len(vecbase):
455                 vecbase[i] = Decimal(str(vecbase[i][0]))
            mid = np.dot(homog,vectop)
457         #dott = [[mid[0]],[mid[1]],[mid[2]]]
            t=[[],[],[]]
459         for i in range(3):
                t[i] = mid[i] - vecbase[i]
461         length = ma.sqrt(t[0]**2+t[1]**2+t[2]**2)
            if length > max or length < min:
463             print 'The length of the leg is too small or too large!
                    Aborting.'
                return False
465         else:
                return length
467
        #Homogenous transformation from platform to base
469     def ht(self, co):
            x = co[0]
471         y = co[1]
            z = co[2]
473         a = co[3]#alpha, beta, gamma
            b = co[4]
475         g = co[5]
            nil = Decimal(0)
477         return [[Decimal(str(ma.cos(b)*ma.cos(g)+ma.sin(a)*ma.sin(b
                )*ma.sin(g))),
                    Decimal(str(-ma.cos(b)*ma.sin(g)+ma.sin(a)*ma.sin(
                        b)*ma.cos(g))),
479                 Decimal(str(ma.cos(a)*ma.sin(b))), Decimal(str(x))
                        ],
                    [Decimal(str(ma.cos(a)*ma.sin(g))), Decimal(str(ma.
                        cos(a)*ma.cos(g))),
```

```
481                     Decimal(str(-ma.sin(a))), Decimal(str(y))],
                    [Decimal(str(-ma.sin(b)*ma.cos(g)+ma.sin(a)*ma.cos(
                        b)*ma.sin(g))),
483                     Decimal(str(ma.sin(b)*ma.sin(g) + ma.sin(a)*ma.cos
                        (b)*ma.cos(g))),
                      Decimal(str(ma.cos(a)*ma.cos(b))), Decimal(str(z)
                          )],
485                 [nil,nil,nil,Decimal(1)]]

487
'''
489 Notes:
ONLY MILLING MODE HAS BEEN IMPLEMENTED NOT 3D PRINTING MODE
491 Output format is what? A list with positions and orientations
of the endpoint with current speed.
493 ABC: orientation, defaults to degrees
'''
495 class PostProcessor(object):
    '''
497     or G-code parser
    '''
499 #type could be 'sim' 'ard' or 'dis' (distance calculation
    def __init__(self, partzero = [0,0,100, 0, 0, 0], gcode='M:/
        gcode.txt',
501                  type = 'sim', time =0.1, max = 500, min = 50):
        self.partzero = partzero#location of "part zero" the top
            and middle of the
503         #part that is being milled
        go = True
505         try:
            self.gcode = open(gcode, 'r')#Filename
507         except:
            print 'Could not find file, exiting'
509             go = False
        if go == True:
511             self.type = type
            self.speed = 1
513             self.absolute = True
            self.run = True
515             self.max = max
            self.min = min
517             self.skipline = False
            self.ik = InverseKinematic()
519             self.lastposition = [0,0,0,0,0,0]
            self.time = time #time in seconds between two entries
                in the
521             #commandlist for simulation time depends on fps in the
            #software that transforms pictures into a movie
523             if type == 'sim':
                start,platstart = self.ik.getcoord([0,0,0,0,0,0],
                    self.partzero)
525                 self.commandlist = [[start[0]],[start[1]],[start
                    [2]],[0],[0],[0],
                                    ['Start'],[platstart]]
527                                     #List to be returned to
                                        simulator
                #self.createcommandlist()
529                 d = 0
            elif type == 'ard':
531                 self.commandlist = []
            elif type == 'dis':
533                 self.commandlist = []
```

```
                self.fastdist = 0.0
                self.slowdist = 0.0
                self.milldist = 0.0
                self.nonmilldist = 0.0
                self.mill = False
                #self.createcommandlist()
        '''
        Checks with the user whether he/she wants to continue
        '''
    def checkcontinue(self):
        answer = False
        while answer == False:
            c = raw_input('> ');
            if c == 'y' or c == 'Y':
                answer = True
            elif c == 'n' or c == 'N':
                self.run = False
                answer = True
        '''
        Creates a commandlist from the imported G-code file.
        A number of commands are not implemented
        '''
    def createcommandlist(self):
        commands = self.gcode.readlines()
        movelist = [False, False, False, False, False, False]
        movenow = False

        for i in range(len(commands)):
            if len(self.commandlist[0]) > 200:
                break
            self.skipline = False
            commandline = commands[i]
            if self.run == True and (commands[i][0] == 'N' or
                commands[i][0] == 'n'):
                for j in range(len(commands[i])):
                    one = commands[i][j]
                    try:
                        one = int(one)
                    except:pass

                    if self.skipline == False and type(one) == str
                        and one != '.' \
                    and one != 'N':
                        if one == 'G' or one == 'g':
                            self.gcommand(commands[i], j, i)
                        elif one == 'M' or one == 'm':
                            self.mcommand(commands[i],j, i)
                        elif one == 'X':
                            movelist[0], movenow = self.xyzabc(
                                commands[i], j, i)
                        elif one == 'Y':
                            movelist[1], movenow = self.xyzabc(
                                commands[i], j, i)
                        elif one == 'Z':
                            movelist[2], movenow = self.xyzabc(
                                commands[i], j, i)
                        elif one == 'A':
                            movelist[3], movenow = self.xyzabc(
                                commands[i], j, i)
                        elif one == 'B':
                            movelist[4], movenow = self.xyzabc(
                                commands[i], j, i)
```

```
589                         elif one == 'C':
                                movelist[5], movenow = self.xyzabc(
                                    commands[i], j, i)
591
                            elif one == 'F':
593                             print 'Warning, command ', one, ' in
                                    line ', i+1,
                                'could not be implemented.'
595
597                         if len(commands[i])-1 == j and movenow ==
                                True:
                                movenow = False
599                             now = [self.commandlist[0][-1], self.
                                    commandlist[1][-1],\
                                        self.commandlist[2][-1],self.
                                            commandlist[3][-1],\
601                                     self.commandlist[4][-1],self.
                                            commandlist[5][-1]]
                                platnow = self.commandlist[7][-1]
603                             for k in range(len(movelist)):
                                    if movelist[k] == False:
605                                     movelist[k] = self.lastposition
                                            [k]
                                if self.type == 'sim':
607                                 self.addmovement(now,platnow,[
                                        Decimal(str(movelist[0])),
                                    Decimal(str(movelist[1])),Decimal(
                                        str(movelist[2])),
609                                 Decimal(str(movelist[3])),Decimal(
                                        str(movelist[4])),
                                    Decimal(str(movelist[5]))],self.
                                        speed, self.time)
611                             elif self.type == 'ard':
                                    self.addardmove(now,platnow,[
                                        Decimal(str(movelist[0])),
613                                 Decimal(str(movelist[1])),Decimal(
                                        str(movelist[2])),
                                    Decimal(str(movelist[3])),Decimal(
                                        str(movelist[4])),
615                                 Decimal(str(movelist[5]))],self.
                                        speed)
                                elif self.type == 'dis':
617                                 self.adddistance(self.lastposition,
                                        movelist)
                                self.lastposition = movelist
619                             movelist = [False, False, False, False,
                                    False, False]
        return self.commandlist
621
    '''
623 Finds out what type of G-command
    '''
625 def gcommand(self, commandline, j, i):
        infolist = self.getonetwothree(commandline,j, i)
627     if infolist[1] != False and infolist[2] != False:
            two = infolist[1]
629         three = infolist[2]
            if two == '0':
631             self.gzero(commandline, j, i, three)
            elif two == '2':
633             self.gtwo(commandline, j, i, three)
```

```
            elif two == '4':
635             self.gfour(commandline, j, i, three)
            elif two == '9':
637             if three == '0':#absolute programming
                    self.absolute = True
639             else:
                    print 'Warning, command G' , two, three, '
                        could not be implemented.'
641
            else:
643             print 'Warning, command G' , two, three, ' could
                    not be implemented.'

645     else:
            print 'Warning, command G in line ' , i+1, ' could not
                be implemented.'
647
    '''
649 Finds what type of G0x command
    '''
651 def gzero(self, commandline, j, i, three):
        if three == '0':
653         self.speed = 'fast'
        elif three == '1':
655         self.speed = 'slow'
        elif three == '2' or three == '3':#Circular interpolation
            interpolation
657         print 'Circular interpolation not implemented. Line ',
                i+1, ' will be skipped.'
            self.skipline = True
659     elif three == '4':#dwell
            try:
661             if commandline[j+4] == 'P' or commandline[j+4] == '
                    U' or commandline[j+4] == 'X':
                    dwelltime = 0
663                 dwellstring = ''
                    for k in range(j+4, len(commandline)):
665                     if commandline[k] == ' ':
                            break
667                     else:
                            dwellstring = dwellstring + (
                                commandline[k])
669                 dwelltime = int(dwellstring)
                    if self.type == 'sim':
671                     self.addlast()
                        for mm in range(int(dwelltime)*10):
673                         self.commandlist[6].append(['Dwell for
                                some time ', dwelltime])
                    elif self.type == 'ard':
675                     self.commandlist.append([3, dwelltime])
            except:
677             print 'Warning, could not implement dwell in line '
                    , i+1, '.'
                #print 'Could not implement dwell in line ', i, '.
                    Continue? [y/n]'
679             #self.checkcontinue()
        else:
681         print 'Warning, could not implement G0' , three, '.'
            print 'Command, G0' , three, ' could not be implemented
                , continue? [y/n]'
683         self.checkcontinue()
    '''
```

```
685         Finds type of G2x command
            '''
687         def gtwo(self, commandline, j, i, three):
                if three == '0':#programming in inches
689                 print 'Inches as metric unit not implemented. Terminate
                        ? [y/n] '
                    print 'If no, the program will use the values as
                        millimeters.'
691                 print '(High risk of crash etc.)'
                    self.checkcontinue()
693             elif three == '1':#programming in millimeters
                    if self.type == 'sim':
695                     self.addlast()
                        self.commandlist[6].append('Programming in
                            millimeters')
697                 elif self.type == 'ard':
                        self.commandlist.append(0)
699             elif three == '8':#return to home position
                    now = [self.commandlist[-1][0],
701                        self.commandlist[-1][1],
                           self.commandlist[-1][2]]
703                 if self.type == 'sim':
                        self.addmovement(now,[0,0,0], 'fast', self.time)
705                 elif self.type == 'ard':
                        self.addardmovement(now,[0,0,0], 'fast')
707             else:
                    print 'Warning, command G2', three, ' could not be
                        implemented.'
709                 #print 'Command, G2' , three, ' could not be
                        implemented, continue? [y/n]'
                    #self.checkcontinue()
711
            '''
713         Finds type of G4x command
            '''
715         def gfour(self, commandline, j, i, three):
                hcheck = False
717             try:
                    hcheck = commandline[j+4]
719             except:pass
                if hcheck != False:
721                 if three == '3':#Tool height offset compensation
                        negative
                        if hcheck == 'H':#try
723                         offsetstring = ''
                            for k in range(j+4, len(commandline)):
725                             if commandline[k] == ' ':
                                    break
727                             else:
                                    offsetstring = offsetstring + (
                                        commandline[k])
729                         offset = -float(offsetstring)
                            self.ik.settooloffset(offset)
731                         if self.type == 'sim':
                                self.addlast()
733                             self.commandlist[6].append(['Tool height
                                    offset', offset])
                            elif self.type == 'ard':
735                             self.commandlist.append([2, offset])
                    elif three == '4':#Tool height offset compensation
                        positive
737                     if commandline[j+4] == 'H':#try
```

```
                    offsetstring = ''
                    for k in range(j+4, len(commandline)):
                        if commandline[k] == ' ':
                            break
                        else:
                            offsetstring = offsetstring + (
                                commandline[k])
                    offset = float(offsetstring)
                    self.ik.settooloffset(offset)
                    if self.type == 'sim':
                        self.addlast()
                        self.commandlist[6].append(['Tool height
                            offset', offset])
                    elif self.type == 'ard':
                        self.commandlist.append([2, offset])
            elif three == '9':#Tool length offset compensation
                cancel
                self.ik.settooloffset(0)
                if self.type == 'sim':
                    self.addlast()
                    self.commandlist[6].append(['Tool height offset
                        ', 0])
                elif self.type == 'ard':
                    self.commandlist.append([2, 0])
            else:
                print 'Warning, command G4', three, 'could not be
                    implemented.'
        else:
            print 'Warning, command G4', three, 'could not be
                implemented. No value given'
    '''
    Adds the positon of the last command, used for non-moving
        commands
    '''
    def addlast(self):
        for i in range(len(self.commandlist)):
            if i != 6:
                self.commandlist[i].append(self.commandlist[i][-1])
    '''
    Decide type of Mxx command
    '''
    def mcommand(self, commandline, j, i):
        infolist = self.getonetwothree(commandline,j, i)
        if infolist[1] != False and infolist[2] != False:
            two = infolist[1]
            three = infolist[2]
            if two == '0':
                if self.type == 'sim':
                    if three == '0':#compulsory stop
                        self.addlast()
                        self.commandlist[6].append('Stopping
                            machine!')
                        self.run =False
                    elif three == '1':#optional stop
                        self.addlast()
                        self.commandlist[6].append('Push button to
                            stop machine!')
                    elif three == '2':#end of program
                        self.addlast()
                        self.commandlist[6].append('End program')
                        self.run = False
                    elif three == '3':#spindle on (clockwise)
```

```
791                         self.addlast()
                            self.commandlist[6].append('Spindle on (
                                clockwise)')
793                     elif three == '4':#spindle on (counterclockwise
                            )
                            self.addlast()
795                         self.commandlist[6].append('Spindle on (
                                counterclockwise)')
                        elif three == '5':#spindle stop
797                         self.addlast()
                            self.commandlist[6].append('Stop spindle')
799                     else:
                            print 'Warning, command, M' , two, three,
801                         ' could not be implemented'
                    elif self.type == 'ard':
803                     if three == '0':#compulsory stop
                            self.commandlist.append(1)
805                     elif three == '1':#optional stop
                            self.commandlist.append(2)
807                     elif three == '2':#end of program
                            self.commandlist.append(3)
809                     elif three == '3':#spindle on (clockwise)
                            self.commandlist.append(4)
811                     elif three == '4':#spindle on (counterclockwise
                            )
                            self.commandlist.append(5)
813                     elif three == '5':#spindle stop
                            self.commandlist.append(6)
815                     else:
                            print 'Warning, command, M' , two, three, \
817                         ' could not be implemented'
                    elif self.type == 'dis':
819                     if three == '3' or three == '4':
                            self.mill = True
821                     elif three == '5':
                            self.mill = False
823             else:
                    print 'Waring, command, M' , two, three, ' could
                        not be implemented.'
825                 #print 'Command, M' , two, three, ' could not be
                        implemented, continue? [y/n]'
                    #self.checkcontinue()
827     '''
        Tries to find the numbers after a command
829     '''
        def getonetwothree(self, commandline, j, i):
831         one = commandline[j]
            two = False
833         three = False
            try:
835             two = commandline[j+1]
            except:
837             print 'Warning, command ', one, ' in line ' , i+1, '
                    could not be implemented.'
            if two != False:
839             try:
                    three = commandline[j+2]
841             except:
                    print 'Warning, command ', one, two, ' in line ' ,
                        i+1,
843                 ' could not be implemented.'
```

```
845             return [one, two, three]

847         '''
            Tries to find x, y, z, a, b and c at the current line for move
                commands
849         '''
            def xyzabc(self, commandline, j, i):
851             movenow = False
                if commandline[j] == 'X':
853                 searchlist = ['Y', 'Z', 'A', 'B', 'C']
                elif commandline[j] == 'Y':
855                 searchlist = ['X', 'Z', 'A', 'B', 'C']
                elif commandline[j] == 'Z':
857                 searchlist = ['Y', 'X', 'A', 'B', 'C']
                elif commandline[j] == 'A':
859                 searchlist = ['Y', 'Z', 'X', 'B', 'C']
                elif commandline[j] == 'B':
861                 searchlist = ['Y', 'Z', 'A', 'X', 'C']
                elif commandline[j] == 'C':
863                 searchlist = ['Y', 'Z', 'A', 'B', 'X']
                movestring = ''
865             for k in range(j+1, len(commandline)):
                    if commandline[k] == ' ' or commandline[k] == '\n':
867                     movenow = True
                        for m in range(k+1, len(commandline)):
869                         if commandline[m] in searchlist:
                                movenow = False
871                             break
                        break
873                 else:
                        movestring = movestring + commandline[k]
875             move = float(movestring)
                return move, movenow
877         '''
            Add movement to be used to communicate with the Arduino
879         '''
            def addardmove(self, now, platnow, next, speed):
881             v = 1
                if speed == 'fast':
883                 v = 2
                elif speed == 'slow':
885                 v = 1
                difference, max = self.ik.getardlegdiff(now, next, self.
                    partzero, self.max,
887                                                         self.min)
                if difference == False:
889                 self.run = False
                else:
891                 self.commandlist.append([1,difference, max, v])

893         '''
            Add items to the commandlist list given speed and start and end
                 positions
895         Fast: 30 mm/s (subject to change)
            Slow: 15 mm/s (subject to change)
897         10 'Hz' gives 10 pictures per second
            start: the starting point of the movement
899         end: the ending point of the movement
            Fast: each picture can move 0.30 mm
901         Slow: each picture can move 0.15 mm
            '''
903         def addmovement(self, start, platstart, end, speed, sec=0.1):
```

```
            v = 15
905         if speed == 'fast':
                v = 30
907         elif speed == 'slow':
                v = 15
909         pointend,platend = self.ik.getcoord(end, self.partzero)

911         #Find the legnth between the centrepoint of the platform at
                end and at start
            length = Decimal(str(ma.sqrt((platend[0]-platstart[0])**2 +
                (platend[1] -
913                             platstart[1])**2 + (platend[2] -
                                platstart[2])**2)))
            less = v*sec
915         if float(length) < less:
                print 'Too short between commands, will use next, the
                    length: ',
917             float(length)
            else:
919             listentries = int(ma.ceil(length/Decimal(str(v*sec))))
                '''the sec at the end says how many
921             seconds between each command/list entry/frame
                what it does: finds absolute length between start and
                    end and divides it
923             with the maximum length for the sat speed and time
                    between
                two entries. This equals the number of entries is
                    needed between
925             start and end with the given speed and time between two
                     entries.
                Analysis is for the endpoint.  '''
927
                declist = Decimal(listentries) #To use decimal to
                    perform mathematical operations
929             #print (end[0]-start[0])/0.3
                xsteppo = Decimal(str(pointend[0]-start[0]))/declist
931             ysteppo = Decimal(str(pointend[1]-start[1]))/declist
                zsteppo = Decimal(str(pointend[2]-start[2]))/declist
933
                astep   = Decimal(str(end[3]-start[3]))/declist
935             bstep   = Decimal(str(end[4]-start[4]))/declist
                cstep   = Decimal(str(end[5]-start[5]))/declist
937
                xsteppl = Decimal(str(platend[0]-platstart[0]))/declist
939             ysteppl = Decimal(str(platend[1]-platstart[1]))/declist
                zsteppl = Decimal(str(platend[2]-platstart[2]))/declist
941
                pos = start
943             platpos = platstart
                addangle = [start[3],start[4],start[5]]
945             for i in range(listentries):
                    pos = [pos[0]+xsteppo, pos[1]+ysteppo, pos[2]+
                        zsteppo]
947                 addangle = [addangle[0]+astep, addangle[1]+bstep,
                        addangle[2]+cstep]
                    platpos = [platpos[0]+xsteppl, platpos[1]+ysteppl,
                        platpos[2]+zsteppl]
949                 platnpos = [platpos[0]+xsteppl, platpos[1]+ysteppl,
                         platpos[2]+zsteppl]
                    addplatpos = self.checklegspeed(platpos, platnpos,
                        sec*v)
951
```

```
                if len(addplatpos) > 3:
                    divided = len(addplatpos)/3
                    addpos =[]
                    addangle = []
                    newsteps = [xsteppo/divided, ysteppo/divided,
                        zsteppo/divided]
                    newanglesteps = [astep/divided, bstep/divided,
                        cstep/divided]
                    for j in range(divided):
                        addpos.append(newsteps[0]*(j+1))
                        addpos.append(newsteps[1]*(j+1))
                        addpos.append(newsteps[2]*(j+1))
                        addangle.append(newanglesteps[0]*(j+1))
                        addangle.append(newanglesteps[1]*(j+1))
                        addangle.append(newanglesteps[2]*(j+1))
                else:
                    addpos = pos

                for j in range(0,len(addpos),3):
                    self.commandlist[0].append(addpos[j])
                    self.commandlist[1].append(addpos[j+1])
                    self.commandlist[2].append(addpos[j+2])
                    self.commandlist[3].append(addangle[j])#if no
                        orientation is given
                    self.commandlist[4].append(addangle[j+1])
                    self.commandlist[5].append(addangle[j+2])
                    self.commandlist[7].append([addplatpos[j],
                        addplatpos[j+1],
                                                addplatpos[j+2]])
                    if speed == 'fast':
                        self.commandlist[6].append('Fast move')
                    elif speed == 'slow':
                        self.commandlist[6].append('Slow move')

    '''
    Recursion:
    Checks the speed of the legs, if the speed for one of
    the legs is too fast, the positions are broken
    up in two etc. This function/method is for simulation.
    '''
    def checklegspeed(self, pos, nextpos, maxdist):
        if self.run == True:
            for i in range(len(pos)):
                pos[i] = Decimal(pos[i])
                nextpos[i] = Decimal(nextpos[i])
                #0 success, 1-6 leg that failed
            speedcheck = self.ik.checkvelocity(pos, nextpos,
                maxdist, self.max,
                                                self. min)
            returnpos = []
            if speedcheck == False:
                self.run == False
                #return False
            elif 0 not in speedcheck:
                print "Speed for leg(s) " , speedcheck ,
                " is too fast. Movement is slowed down."
                #add a point in between
                midpos = [0,0,0]
                for i in range(len(nextpos)):
                    midpos[i] = Decimal(str(nextpos[i]-pos[i]))/2+
                        pos[i]
                one = self.checklegspeed(pos,midpos, maxdist)
```

```
                    for i in range(len(one)):
1009                    returnpos.append(one[i])
                    one = self.checklegspeed(midpos, nextpos,maxdist)
1011                for i in range(len(one)):
                        returnpos.append(one[i])
1013                return returnpos
                else:
1015                returnpos = pos
                    return returnpos
1017        else:
                return False
1019
    #Used for testing:
1021 pp = PostProcessor(gcode='c:/gcode.txt')
    #com = pp.createcommandlist()
1023 sim = Simulator()
    #sim.commandlist = com
1025 sim.loadsimple()
    sim.animation()
```

Simulator.py

A.2 Biologically Inspired Path-optimization

This code is the implementation of what has been discussed in chapter 6. It has 5 classes: GCodeReader, TestGA, GeneticAlgorithm, TestAnt and AntSystem.

```
#packeges used
import math as ma
import numpy as np
from decimal import *
import random
import matplotlib.pyplot as plt


class GcodeReader(object):
    '''
    Parse G-code and the data to be used by the optimization
    algorithms. Can also find lengths of paths where the tool
    is inactive (fast)'''
    def __init__(self, gcode='M:/gcode2.txt'):
        '''
        Constructor
        '''
        go = True
        try:
            self.gcode = open(gcode, 'r')#Filename
        except:
            print 'Could not find file, exiting'
            return False
            go = False
        if go == True:
            self.slowdistance = 0
            self.speed = False
            self.run = True
            self.skipline = False
            self.lastposition = [0,0,0]
            self.commandlist = []
            self.fastdist = 0.0
            self.slowdist = 0.0
            self.milldist = 0.0
            self.nonmilldist = 0.0
            self.lastmovetype = False
            self.mill = False
            self.tooloffset = 0
            self.firstelement = True
    '''
    Start to read the code, similar to "Post processor"/G-code
        parser
    '''
    def readgcode(self):
        commands = self.gcode.readlines()
        self.movelist = [False, False, False]
        movenow = False
        for i in range(len(commands)):
            skipline = False
            commandline = commands[i]
            self.movelist = [False, False, False]
            if self.run == True and (commands[i][0] == 'N' or
                commands[i][0] == 'n'):
                for j in range(len(commands[i])):
                    one = commands[i][j]
                    if skipline == False and type(one) == str and
                        one != '.' and one != 'N':
```

```
                if one == 'G' or one == 'g':
56                  infolist = self.getonetwothree(
                        commandline,j, i)
                    if infolist[1] != False and infolist[2]
                         != False:
58                      two = infolist[1]
                        three = infolist[2]
60                      if two == '0':
                            if three == '0' or three == ' '
                                :
62                              self.speed = 2
                            elif three == '1':
64                              self.speed = 1
                            elif three == '2' or three == '
                                3':
66                              #circular interpolation not
                                     implemented
                                skipline = True
68                              self.speed = False
                                print 'Circular
                                    interpolation not
                                    implemented,',
70                              'skipping line ', i+1
                        elif two == '1' and three == ' ' :
72                          self.speed = 1
                        elif two == '2' and three == ' ' :
74                          self.speed = False
                            skipline = True
76                      elif two == '3' and three == ' ' :
                            self.speed = False
78                          skipline = True
                        elif two == '2' and three == '8':
80                          self.movelist = [0,0,0]
                            self.speed = 2
82                          self.movenow = True
                        elif two == '4':#tool offset 43-
                            44+ 49 0
84                          hcheck = False
                            try:
86                              hcheck = commandline[j+4]
                            except:pass
88
                            if hcheck == 'H':
90                              offsetstring = ''
                                for k in range(j+4, len(
                                    commandline)):
92                                  if commandline[k] == '
                                        ':
                                        break
94                                  else:
                                        offsetstring =
                                            offsetstring +
                                            \
96                                      (commandline[k])
                                if three == '3':#Tool
                                    height offset
98                                  #compensation negative
                                    self.tooloffset = -
                                        float(offsetstring)
100                             elif three == '4':#Tool
                                    height offset
                                    #compensation positive
```

```
102                                         self.tooloffset = float
                                                (offsetstring)
                                        elif three == '9':#Tool
                                            length offset
104                                         #compensation cancel
                                            self.tooloffset = 0
106                                     else:
                                            print 'Warning, command
                                                G4', three,
108                                         'could not be
                                                implemented.'
                                    else:
110                                     print 'Warning, command G4'
                                            , three,
                                        'could not be implemented.
                                            No value given'
112                         elif one == 'M' or one == 'm':
                                infolist = self.getonetwothree(
                                    commandline,j, i)
114                             if infolist[1] != False and infolist[2]
                                     != False:
                                    two = infolist[1]
116                                 three = infolist[2]
                                    if two == '0':
118                                     if three == '3' or three == '4'
                                            :
                                        self.mill = True
120                                     elif three == '5':
                                        self.mill = False
122                         elif one == 'X' or one == 'x':
                                movenow = self.findvalue(commands[i], j
                                    , 'X')
124                         elif one == 'Y' or one == 'y':
                                movenow = self.findvalue(commands[i], j
                                    , 'Y')
126                         elif one == 'Z' or one == 'z':
                                movenow = self.findvalue(commands[i], j
                                    , 'Z')
128                         if movenow == True:
                                if self.speed != False:
130                                 self.addmovement()
                                    movenow = False
132         print self.slowdistance
    '''
134 Finds the x, y and z value in a line of code
    '''
136 def findvalue(self, line, j, axis):
        if axis == 'X':
138         self.movelist[0] = Decimal(str(self.getvalue(line, j)))
        elif axis == 'Y':
140         self.movelist[1] = Decimal(str(self.getvalue(line, j)))
        elif axis == 'Z':
142         self.movelist[2] = Decimal(str(self.getvalue(line, j) +
                self.tooloffset))
        movenow = True
144
        for i in range(j+1, len(line)):
146         if line[i] == 'X' or line[i] == 'Y' or line[i] == 'Z':
                movenow = False
148     return movenow

150 '''
```

```
    add movement information to self.commandlist
152 '''
    def addmovement(self):
154     for i in range(len(self.movelist)):
            if self.movelist[i] == False:
156             if self.firstelement == True:
                    self.movelist[i] = 0
158             else:
                    self.movelist[i] = self.commandlist[-1][1][i]
160     if self.firstelement == True:
            lastpos = [0,0,0]
162         self.firstelement = False
        else:
164         lastpos = self.commandlist[-1][1]
        distance = Decimal(str(ma.sqrt((self.movelist[0]- lastpos
            [0])**2+
166                        (self.movelist[1]- lastpos[1])**2+
                           (self.movelist[2]- lastpos[2])**2)))
168     if self.speed == 1:
            self.slowdistance = self.slowdistance + distance
170     if self.lastmovetype == self.speed:#same type of movement
            self.commandlist[-1][1] = self.movelist[:]
172         self.commandlist[-1][2] = self.commandlist[-1][2] +
                distance#increase distance
        else:
174         self.commandlist.append([lastpos[:],self.movelist[:],
                distance,self.speed])
            self.lastmovetype = self.speed
176
    '''Finds a number after a letter'''
178 def getvalue(self, line, j):
        value = ''
180     for i in range(j+1, len(line)):
            if line[i] == ' ':
182             break
            else:
184             value = value + line[i]
        debug = float(value)
186     print debug
        return float(value)
188
    '''    Tries to find the numbers after a command'''
190 def getonetwothree(self, commandline, j, i):
        one = commandline[j]
192     two = False
        three = False
194     try:
            two = commandline[j+1]
196     except:
            print 'Warning, command ', one, ' in line ', i+1, '
                could not be implemented.'
198     if two != False:
            try:
200             three = commandline[j+2]
            except:
202             print 'Warning, command ', one, two, ' in line ',
                    i+1,
                ' could not be implemented.'
204     return [one, two, three]

206 '''Returns a list of all the slow movements'''
    def getslowdata(self):
```

```
208         slowlist = []
            slowlist.append([[0,0,0],[0,0,0]])
210         for i in range(len(self.commandlist)):
                if self.commandlist[i][3] == 1:
212                 slowlist.append(self.commandlist[i][0:2])
            return slowlist
214
        #creates a list of distances (two between each entry) and
            returns it
216     def preparefortsp(self, data):
            distancematrix = [[0 for x in data] for x in data]
218         for i in range(len(data)):
                for j in range(i+1, len(data)):
220                 #from start of i to start of j
                    dis1 = ma.sqrt((data[i][0][0]-data[j][0][0])**2+(
                        data[i][0][1]- \
222                             data[j][0][1])**2+(data[i][0][2]-data[j
                                ][0][2])**2)
                    dis2 = ma.sqrt((data[i][1][0]-data[j][0][0])**2+(
                        data[i][1][1]- \
224                             data[j][0][1])**2+(data[i][1][2]-data[j
                                ][0][2])**2)
                    dis3 = ma.sqrt((data[i][0][0]-data[j][1][0])**2+(
                        data[i][0][1]- \
226                             data[j][1][1])**2+(data[i][0][2]-data[j
                                ][1][2])**2)
                    dis4 = ma.sqrt((data[i][1][0]-data[j][1][0])**2+(
                        data[i][1][1]- \
228                             data[j][1][1])**2+(data[i][1][2]-data[j
                                ][1][2])**2)
                    distancematrix[i][j] = [dis1,dis2,dis3,dis4]
230                 distancematrix[j][i] = [dis1,dis2,dis3,dis4]
            return distancematrix
232     ''' Returns the fast distance '''
        def getfastdistance(self, data):
234         distance = 0
            for i in range(1,len(data)):
236             dis = ma.sqrt((data[i-1][1][0] - data[i-1][0][0])**2+(
                    data[i-1][1][1] -
                            data[i-1][0][1])**2 +(data[i-1][1][2] -
                                data[i-1][0][2])**2)
238             distance = distance + dis
            dis = ma.sqrt((data[-1][1][0] - data[0][0][0])**2+(data
                [-1][1][1] - \
240                     data[0][0][1])**2 +(data[-1][1][2] - data
                        [0][0][2])**2)
            distance = distance + dis
242         print 'Initial distance: ', distance
            return distance
244
    '''Used for testing the genetic algorithm'''
246 class TestGA(object):
        def __init__(self, runs, file = 'c:/gcode.txt', populationsize
            = 100,
248                  generations = 50, pc = 0.6, pm = 0.1, ranktype =
                        1, q = 0.5):
            originallength = 0
250         evals = 0
            best = False
252         worst = False
            sumruns = 0
254         ga = GeneticAlgorithm(file, populationsize, generations, pc
```

```
            , pm, ranktype, q)
        originallength = ga.getstuff()
        for i in range(runs):
            runbest = ga.master()
            if best == False:
                best = runbest[:]
            else:
                if runbest[0] < best[0]:
                    best = runbest[:]
            if worst == False:
                worst = runbest[:]
            else:
                if runbest[0] > worst[0]:
                    worst = runbest[:]
            evals = evals + ga.evaluations
            sumruns = sumruns + runbest[0]
        meanbest = float(sumruns)/runs
        percentage = float(best[0])/float(originallength)*100
        eva = evals/runs
        print 'Best ever: ', best[0]
        print 'Best run ever: ', best[1:]
        print 'Original length', originallength
        print 'Best ever percentage of original length', percentage
        print 'Worst ever: ', worst[0]
        print 'Mean best fitness over ', runs, ' runs: ', meanbest
        print 'Average number of evaluations: ', eva

'''
Path-optimization as TSP solved by the genetic algorithm
PMX crossover
Inversion mutation
SUSampling
ranking selection
generational replacement scheme with elitism
No of Generetions termination criterion

'''
class GeneticAlgorithm(object):
    def __init__(self, file = 'c:/gcode', populationsize = 100,
        generations = 50,
                 pc = 0.6, pm = 0.1, ranktype = 1, q = 0.5):
        self.q = q
        self.file = file
        self.evaluations = 0 #number of evaluations
        self.ranktype = ranktype #1= exponetial, 0 = linear
        self.pc = pc
        self.pm = pm
        if populationsize % 2 != 0:
            print 'This algorithm do not accept an odd number of',
            ' individuals in the population.'
            print 'Change to ', populationsize + 1,' [y] or quit [q
                ]?'
            populationsize = populationsize + 1
        #self.offspringsize = offspringsize
        self.gener = generations
        self.popsize = populationsize
        self.population = []

    '''Get the data from the G-code reader'''
    def getstuff(self):
        self.gcr=0
        try:
```

```
314             self.gcr = GcodeReader(self.file)
        except:
316             pass
        if self.gcr != False:
318             self.gcr.readgcode()
                self.data = self.gcr.getslowdata()
320             #self.gcr.printfastdistance(self.data)
                self.findslowlength()
322             self.amount = len(self.data)#size of individual
                self.individual = [[x] for x in range(self.amount)]
324             #distance matrix
                self.dismat = self.gcr.preparefortsp(self.data)
326         return self.gcr.getfastdistance(self.data)

328     '''Get simple data for initial testing'''
    def getsimpledata(self):
330         self.dismat = [[0 for x in range(5)]for x in range(5)]
            self.amount = len(self.dismat)
332         for i in range(1,self.amount):
                dis1 = random.randint(0,100)
334             dis2 = random.randint(0,100)
                self.dismat[0][i] = [dis1,dis1,dis2,dis2]
336             self.dismat[i][0] = [dis1,dis1,dis2,dis2]
            for i in range(1, len(self.dismat)):
338             for j in range(i,len(self.dismat)):
                    if i != j:
340                     distance = [random.randint(0,100),random.
                            randint(0,100),
                                    random.randint(0,100),random.
                                        randint(0,100)]
342                     self.dismat[i][j] = distance
                        self.dismat[j][i] = distance
344         self.individual = [[x] for x in range(self.amount)]

346     '''Main function, controls the other functions'''
    def master(self):
348         self.population = []
            self.evaluations = 0
350         self.createinitial()
            itbesttour = []
352         for i in range(self.gener):
                #if best individual = superbra: break
354             offspring = []
                rankedind = self.rankindividuals()
356             matingpool = self.selectparent(rankedind)
                for j in range(0,self.popsize,2):
358                 child1=[]
                    child2=[]
360                 parent1=[]
                    parent2=[]
362                 #Select parents
                    parent1 = matingpool.pop(random.randint(0,self.
                        popsize-1-j))
364                 parent2 = matingpool.pop(random.randint(0,self.
                        popsize-2-j))
                    usepar1 = []
366                 usepar2 = []
                    for gf in range(1, len(parent1)):
368                     gene1 = []
                        gene2 = []
370                     for gg in range(2):
                            try:
```

```
372                         gene1.append(parent1[gf][gg])
                            gene2.append(parent2[gf][gg])
374                     except:
                            print 'what now'
376                 usepar1.append(gene1)
                    usepar2.append(gene2)
378             #Is there a crossover?
                p = random.uniform(0,1)
380             if p < self.pc:
                    child1,child2 = self.crossover(usepar1,usepar2)
382                 #Is there a mutation of the offspring?
                    p = random.uniform(0,1)
384                 if p < self.pm:
                        child1 = self.mutate(child1[:])
386                     child2 = self.mutate(child2[:])
                    #Add the children with their tour length to the
                         offspring-pool
388                 #child1.insert(0, self.evaluateindividual(
                        child1))
                    #child2.insert(0, self.evaluateindividual(
                        child2))
390                 offspring.append(child1[:])
                    offspring.append(child2[:])
392             else:
                    #if no crossover, is there a mutation of the
                        parents?
394                 p = random.uniform(0,1)
                    if p < self.pm:
396                     mut1 = self.mutate(usepar1)
                        mut2 = self.mutate(usepar2)
398                     #Add the mutated parents with their length
                            to the offspring-pool
                        #mut1.insert(0, self.evaluateindividual(
                            mut1))
400                     #mut2.insert(0, self.evaluateindividual(
                            mut2))
                        offspring.append(mut1[:])
402                     offspring.append(mut2[:])
                    else:
404                     #The parents survived for another
                            generation!
                        offspring.append(parent1[:])
406                     offspring.append(parent2[:])

408         for o in range(self.popsize):
                if type(offspring[o][0])== list:
410                 jepp=[]
                    for gf in range(len(offspring[o])):
412                     gene1 = []

414                     for gg in range(2):
                            try:
416                             gene1.append(offspring[o][gf][gg])

418                         except:
                                print 'what now'
420                 jepp.append(gene1)
                    tepp = self.checkedges(jepp)
422                 cost = self.evaluateindividual(tepp)
                    ny = [cost]
424                 for p in tepp:
                        ny.append(p)
```

```
426                     offspring[o] = ny
            #Sort the offspring
428             offspring.sort()
                offspring[-1] = self.population[0][:]#elitism
430             itbesttour.append(offspring[-1][0])
                #when generational, all offspring replace the parents
432             self.population = []
                self.population = offspring[:]
434         #Uncomment to allow plotting.
            '''ax = plt.subplot(1,1,1)
436         x = range(0,self.gener)
            #x = range(self.ants)
438         #ax.plot(x, listmeantour, label = "Mean")
            ax.plot(x, itbesttour, label = "Best in this generation")
440         plt.ylabel('Cost of tour')
            plt.grid(True)
442         #plt.xlabel('Ant')
            plt.xlabel('Generations')
444         ax.legend(loc=1, ncol=3, shadow=True)
            plt.show()      '''
446         cost = self.evaluateindividual(self.population[0][1:])
            print self.population[0][0]#shortest trip
448         #print nr_eval #number of evaluations of tour
            return self.population[0]
450
    ''' Create initial population randomly '''
452     def createinitial(self):
            for i in range(self.popsize):
454             ind = random.sample([[x] for x in range(self.amount)],
                    self.amount)
                individual = self.addrandomedges(ind)
456             individual.insert(0, self.evaluateindividual(individual
                    ))
                self.population.append(individual)
458         self.population.sort()
        def addrandomedges(self, individual):
460         for j in range(len(individual)):
                if j == 0:
462                 p = random.random()
                    if p <= 0.25:
464                     individual[j].append(0)
                    elif 0.25 < p <= 0.5:
466                     individual[j].append(1)
                    elif 0.5 < p <= 0.75:
468                     individual[j].append(2)
                    else:
470                     individual[j].append(3)
                else:
472                 entry = individual[j-1][1]
                    if j == (len(individual)-1):#last element
474                     into = individual[0][1]
                        if entry == 0 or entry == 2:
476                         if into == 0 or into == 1:
                                individual[j].append(3)
478                         else:
                                individual[j].append(2)
480                     else:
                            if into == 0 or into == 1:
482                             individual[j].append(1)
                            else:
484                             individual[j].append(0)
                    else:
```

```
                    p = random.random()
                    if entry == 0 or entry == 2:#enters in 0
                        if p <= 0.5:
                            individual[j].append(2)
                        else:
                            individual[j].append(3)
                    elif entry == 1 or entry == 3:#enters in 1
                        if p <= 0.5:
                            individual[j].append(0)
                        else:
                            individual[j].append(1)
        individual = self.checkedges(individual)
        return individual

    #Ranks the individuals
    def rankindividuals(self):
        #Total p (percentage)
        totp = 0
        #Sort the population based on fitness
        self.population.sort()
        rankedindividuals = []

        #Linear. For more details, see page 60-61 of Introduction
            to
        #Evolutionary Computing (Eiben and Smith 2007).
        if self.ranktype == 0:
            self.population.reverse()#least fit first
            #parameterization value 1.0 < s <= 2.0:
            s = 1.5
            i = 0
            for individual in self.population:
                #Selection probability
                #reversed
                psel = (2-s)/self.popsize + 2*i*(s-1)/(self.popsize
                    *(self.popsize-1))
                i += 1
                #Should be 1 when finished
                totp += psel
                if(i<0):
                    print 'Error'
                parent = [individual[0],individual[1], psel, totp]
                rankedindividuals.append(parent)
        #Exponential. For details, see (Michalewicz, 1994) or
        #http://www.aero.caltech.edu/~tamer/GATutorial.pdf
        elif self.ranktype == 1:
            totp=1
            q = self.q #0.5
            for i in range(self.popsize):
                pexp = q*(1-q)**(i) #not reversed population
                parent = [self.population[i][0]]
                for j in range(1, len(self.population[i])):
                    parent.append(self.population[i][j])
                parent.append(pexp)
                parent.append(totp)
                rankedindividuals.append(parent)
                totp -= pexp
            rankedindividuals.reverse()
        else:
            print 'error, wrong input for type of ranking'
        return rankedindividuals

    '''Evaluates an individual, finds length of path'''
```

```
546     def evaluateindividual(self, individual):
            #The total number of evaluations
548         self.evaluations += 1
            fitness = 0
550         #Look through all the gene, add the cost moving between
                them
            for i in range(len(individual)):
552             if i != (len(individual)-1):#not last element
                    try:
554                     distance = self.dismat[individual[i][0]][
                            individual[i+1][0]][individual[i][1]]
                    except:
556                     print 'Something is wrong with this individual'

558             else: #last element, move to first
                    distance = self.dismat[individual[i][0]][individual
                        [0][0]][individual[i][1]]
560
                fitness = fitness + distance
562         return fitness

564     '''Select a parent based on ranking - SUS'''
        def selectparent(self, rankedind):
566         #current_member: the position in the mating_pool
            #i: the individual in the ranked_par list
568         currentmember = i = 0
            p = random.uniform(0,1.0/len(rankedind))
570         #The pool of parents
            matingpool = []
572         while(currentmember < len(rankedind)):
                while (p <= rankedind[i][-1]):
574                 dork = rankedind[i][0:-2]
                    matingpool.append(dork[:])
576                 p = p + (1.0/self.popsize)
                    currentmember = currentmember + 1
578             i = i + 1
            return matingpool
580
        ''' Inversion mutation '''
582     def mutate(self, par):
            parent = par[:]
584         #The number of genes to have their order inverted
            length = random.randint(2,len(parent)-2)
586         #The first gene to have its order inverted
            position = random.randint(0, len(parent)-length)
588         child = parent[:]
            reversedbit = parent[position:(position+length)][:]
590         reversedbit.reverse()
            for i in range(len(reversedbit)):
592             child[i+position] = reversedbit[i]
            #Fix the two broken links
594         #child = self.addedges(child, position)
            #child = self.addedges(child, position+length-1)
596         #child = self.checkedges(child[:])
            #child = self.control(child, True)
598         return child

600     '''Check an individual for faults'''
        def control(self, individual, mut):
602         unused = self.individual[:]
            wronglist = []
604         for i in range(len(individual)):
```

```
            try:
606             unused.remove([individual[i][0]])
            except:
608             print 'Wrong with individual'
                wronglist.append(i)
610     for i in wronglist:
            individual[i] = [unused.pop(),0]
612         self.addedges(individual, i)
        individual = self.checkedges(individual)
614     return individual

616 '''PMX crossover by using two parents'''
    def crossover(self, par1, par2):
618     parent1 = par1[:]
        parent2 = par2[:]
620     #The number of genes from parent 1 to be added to the child
        length = random.randint(1,len(self.individual)-2)
622     #The position of the first gene from parent 1
        try:
624         position = random.randint(0, len(parent1)-length)
        except:
626         print 'randerror'
        child1 = [False]*len(parent1)
628     child2 = [False]*len(parent1)
        map = [[],[]]
630     #i = position
        #Add genes from parent 1 to child 1 and genes from parent 2
             to child 2
632     for i in range(position,(position+length)):
            child1[i] = parent1[i]
634         child2[i] = parent2[i]
            map[0].append(parent1[i])
636         map[1].append(parent2[i])

638     pos = position
        #Add genes from parent 2 to child 1 and from parent 1 to
            child 2
640     for i in range(len(parent1)):
            if child1[i] == False:
642             exist = False
                for gene in child1[position:position+length]:
644                 if gene[0] == parent2[i][0]:
                        exist = True
646             if exist == False:
                    child1[i] = parent2[i]
648             else:
                    child1[i] = self.findgene(child1, map, parent2[
                        i], 1)
650             exist = False
                for gene in child2[position:position+length]:
652                 if gene[0] == parent1[i][0]:
                        exist = True
654             if exist == False:
                    child2[i] = parent1[i]
656             else:
                    child2[i] = self.findgene(child2, map, parent1[
                        i], 2)
658
        return child1, child2
660
    '''Check edges for correctness and fix them!'''
662 def checkedges(self, individual):
```

```
        for i in range(1,len(individual)):
            if i == 0:
                before = individual[-1][1]
            else:
                before = individual[i-1][1]
            if i == (len(individual)-1):
                next = individual[0][1]
            else:
                next = individual[i+1][1]

            if before == 0 or before == 2: #enters at front, 0 and
                1 can not be used
                if next == 0 or next == 1: #next use front, have to
                     use 3
                    individual[i][1] = 3
                else: #have to use 2
                    individual[i][1] = 2
            else: #enters at back, 2 and 3 cannot be used
                if next == 0 or next == 1: #next use front, have to
                     use 1-go in back
                    individual[i][1] = 1
                else:#have 0
                    individual[i][1] = 0
        return individual

    '''Add edges to an individual'''
    def addedges(self, child, edge):
        if edge >= (len(child)-1):#end of individual
            nextedge= child[0][1]
        else:
            try:
                nextedge = child[edge+1][1]
            except:
                print 'wut?'
        if edge == 0:
                beforeedge = child[-1][1]
        else:
            beforeedge = child[edge-1][1]
        if nextedge == 0 or nextedge == 1:#can not use 0 or 2
            if beforeedge == 0 or beforeedge == 2:#can not use 0 or
                 1
                child[edge][1] = 3
            elif beforeedge == 1 or beforeedge == 3:#can not use 2
                or 3
                child[edge][1] = 1
        elif nextedge == 2 or nextedge == 3:#can not use 1 or 3
            if beforeedge == 0 or beforeedge == 2:#can not use 0 or
                 1
                child[edge][1] = 2
            elif beforeedge == 1 or beforeedge == 3:#can not use 2
                or 3
                child[edge][1] = 0
        return child

    '''Used for PMX crossover, finds the right gene'''
    def findgene(self, child, map, gene, chnr):
        index = 0
        for i in range(len(map[chnr-1])):
            if map[chnr-1][i][0] == gene[0]:
                index = i
        if chnr == 1:
            ok = True
```

```
718             for i in range(len(child)):
                    if type(child[i]) == list:
720                     if child[i][0] == map[1][index][0]:
                            ok = False
722             if ok == True:
                    return map[1][index]
724             else:
                    return self.findgene(child, map, map[1][index],
                        chnr)
726         elif chnr == 2:
                ok = True
728             for i in range(len(child)):
                    if type(child[i]) == list:
730                     if child[i][0] == map[0][index][0]:
                            ok = False
732             if ok == True:
                    return map[0][index]
734             else:
                    return self.findgene(child, map, map[0][index],
                        chnr)
736
    '''Used for testing AntSystem'''
738 class TestAnt(object):
        def __init__(self, runs, file = 'm:/25D.txt', ants = 10, alpha
            = 1.0, beta = 1.0,
740                  Q = 1.0, rho = 0.7, generations = 50):
            originallength = 0
742         best = False
            worst = False
744         sumruns = 0
            acs = AntSystem(file, ants, alpha, beta, Q, rho,
                generations)
746         originallength = acs.getstuff()
            for i in range(runs):
748             acs.addpheromone()
                runbest = acs.start()
750             if best == False:
                    best = runbest[:]
752             else:
                    if runbest[0] < best[0]:
754                     best = runbest[:]
                if worst == False:
756                 worst = runbest[:]
                else:
758                 if runbest[0] > worst[0]:
                        worst = runbest[:]
760             evals = acs.evaluations
                sumruns = sumruns + runbest[0]
762         meanbest = float(sumruns)/runs
            percentage = float(best[0])/float(originallength)*100
764         print 'Best ever: ', best[0]
            print 'Path: ', best[1:]
766         print 'Original length', originallength
            print 'Best ever percentage of original length', percentage
768         print 'Worst ever: ', worst[0]
            print 'Mean best fitness over ', runs, ' runs: ', meanbest
770         print 'Average number of evaluations: ', evals

772 '''Solve the TSP problem with Ant System '''
    class AntSystem(object):
774     def __init__(self, file = 'm:/gcode2.txt', ants = 10, alpha =
            1.0, beta = 1.0,
```

```
                 Q = 1.0, rho = 0.7, generations = 50):
776         self.Q = Q#constant: delta-tau=Q/length
            self.rho = rho#evaporation rate
778         self.ants = ants
            self.alpha = alpha
780         self.beta = beta
            self.file = file
782         self.generations = generations
            self.evaluations = ants * generations
784
        '''Get data from Gcode reader'''
786     def getstuff(self):
            self.gcr=False
788         try:
                self.gcr = GcodeReader(self.file)
790             #self.gcr = True
            except:
792             print 'Could not retrieve data'
            if self.gcr != False:
794             self.gcr.readgcode()
                self.data = self.gcr.getslowdata()
796             #self.gcr.printfastdistance(self.data)
                self.amount = len(self.data)#size of individual
798             self.individual = [[x] for x in range(self.amount)]
                #distance matrix
800             self.dismat = self.gcr.preparefortsp(self.data)
            return self.gcr.getfastdistance(self.data)
802
        '''Get simple data for testing'''
804     def getsimpledata(self):
            self.dismat = [[0 for x in range(5)]for x in range(5)]
806         self.amount = len(self.dismat)#length of individual
            for i in range(1,self.amount):
808             dis1 = random.randint(0,100)
                dis2 = random.randint(0,100)
810             self.dismat[0][i] = [dis1,dis1,dis2,dis2]
                self.dismat[i][0] = [dis1,dis1,dis2,dis2]
812         for i in range(1, len(self.dismat)):
                for j in range(i,len(self.dismat)):
814                 if i != j:
                        distance = [random.randint(0,100),random.
                            randint(0,100),
816                                 random.randint(0,100),random.
                                     randint(0,100)]
                        self.dismat[i][j] = distance[:]
818                     self.dismat[j][i] = distance[:]

820
        '''Runs the algorithm, runs for a number of generations and
            finds the total
822     best run.'''
        def start(self):
824         #Average
            listmeantour = []
826         #Best for one run (iteration best tour)
            itbesttour = []
828         #Total shortest tour (best so far)
            sfbesttour = [0]
830         #Add initial pheromone concentration if nonexistent.
            if type(self.dismat[0][1][0]) != list:
832             self.addpheromone()
            termination = 0
```

```
834             self.numbind = len(self.dismat[0])#the number of genes
                generations = 0
836             #Run algorithm
                while termination == 0:
838                 generations += 1
                    routelist = [] #list containing the route of all the
                        ants of this generation
840                 meantour = 0
                    besttour = 0
842                 for ant in range(0,self.ants):
                        length = 0
844                     startgene = random.randint(0,self.numbind-1)
                        tabulist =[startgene] #the cuts the ant has visited
846                     unvisited = range(0, self.numbind) #the cuts the
                            ant has not yet visited
                        try:
848                         unvisited.remove(startgene)
                        except:
850                         print 'Could not remove ' ,startgene ,' from
                                unvisited individuals.'
                        #Add the rest of the cuts
852                     for i in range(0, self.numbind-1):
                            next,edge = self.choosenext(tabulist, unvisited
                                )
854                         tabulist[-1] = [tabulist[-1],edge]
                            length = length + self.dismat[tabulist[-1][0]][
                                next][edge][0]
856                         tabulist.append(next)
                            try:
858                             unvisited.remove(next)
                            except:
860                             print 'Something is wrong with the program'
                        #Add length from last to first city, completing the
                             tour
862                     self.addlastedge(tabulist)
                        length += self.dismat[tabulist[-1][0]][tabulist
                            [0][0]][tabulist[-1][1]][0]
864                     tabulist.append(length)
                        meantour += length
866
                        if length < besttour or besttour == 0:
868                         besttour = length
                        if length < sfbesttour[0] or sfbesttour[0] == 0:
870                         sfbesttour = [length, tabulist[:]]
                            sfbesttour[1].pop()#??
872                     routelist.append(tabulist)

874                 #Update
                    meantour = meantour/self.ants
876                 listmeantour.append(meantour)
                    itbesttour.append(besttour)
878                 self.updatepheromonevalues(routelist)

880                 #the termination condition is:
                    if generations == self.generations:
882                     #Uncomment to allow plotting.
                        #ax = plt.subplot(1,1,1)
884                     #x = range(0,generations)
                        #x = range(self.ants)
886                     #ax.plot(x, listmeantour, label = "Mean")
                        #ax.plot(x, itbesttour, label = "Best")
888                     #ax.plot(x, routelist[:][-1])
```

```
                #plt.ylabel('Cost of tour')
890             #plt.grid(True)
                #plt.xlabel('Ant')
892             #plt.xlabel('Generations')
                #ax.legend(loc=1, ncol=3, shadow=True)
894             #plt.show()
                #print 'The best tour traversed:' , sfbesttour
896             #final_city_order = final_run(self.dismat, self.
                    alpha, self.beta)
                #print 'The last run gives a tour of:' ,
                    final_city_order
898             #print '(Obviously the large number is the length)'
                termination = 1
900         return sfbesttour
        '''
902     Add the last edge of the trip, this is predetermined'''
        def addlastedge(self, tabulist):
904         #enters last in front, can use 2or3
            if tabulist[-2][1] == 0 or tabulist[-2][1] == 2:
906             if tabulist[0][1] == 0 or tabulist[0][1] == 1: #have to
                    enter in back: 3
                    tabulist[-1] = [tabulist[-1],3]
908             elif tabulist[0][1] == 2 or tabulist[0][1] == 3:#have
                    to enter in front: 2
                    tabulist[-1] = [tabulist[-1],2]
910             #enters last in back, can use 0 or1
            elif tabulist[-2][1] == 1 or tabulist[-2][1] == 3:
912             if tabulist[0][1] == 0 or tabulist[0][1] == 1: #have to
                    enter in back: 1
                    tabulist[-1] = [tabulist[-1],1]
914             elif tabulist[0][1] == 2 or tabulist[0][1] == 3:#have
                    to enter in front: 0
                    tabulist[-1] = [tabulist[-1],0]
916
        '''Updates the pheromone for all the edges'''
918     def updatepheromonevalues(self, antpermutations):
            #the sums of tau
920         sums = [[[0,0,0,0] for x in range(self.numbind)] for x in
                range(self.numbind)]
            #First add together the taus in the the sums list
922         for perm in antpermutations:
                length = perm.pop()
924             for i in range(len(perm)-1):
                    sums[perm[i][0]][perm[i+1][0]][perm[i][1]] += self.
                        Q/length
926         #Then calculate the new pheromone values
            for i in range(len(sums)):
928             for j in range(len(sums[i])):
                    if type(self.dismat[i][j]) == list:
930                     for k in range(len(sums[i][j])):
                            self.dismat[i][j][k][1] = (1-self.rho)*self
                                .dismat[i][j][k][1]
932                         + sums[i][j][k]

934     '''Choose the next cut for an ant.'''
        def choosenext(self, tabulist, unvisited):
936         sum = 0.0
            done = 0
938         current = tabulist[-1]
            probabilitylist = []
940         next = 0
            while done == 0:
```

```
942             #calculate probabilities based on pheromone and draw
                    next cut!
                sum = self.findsumeq4(unvisited, current)
944             #this is to depend on pheromone values in the self.
                    dismat
                #and the length of the edge
946             pj = 0.0 #probability of choosing edge j
                #for i in unvisited:#pj+= because has
948             for j in unvisited:#self.dismat[current]
                    for k in self.dismat[current][j]:
950                     try:
                            if k[0] == 0:
952                             pj += k[1]**self.alpha * (1.0/0.01)**
                                    self.beta/sum
                            else:
954                             pj += k[1]**self.alpha * (1.0/k[0])**
                                    self.beta/sum
                        except:
956                         print 'gd'
                        probabilitylist.append(pj)
958             p = random.uniform(0,1)
                i = 0
960             #find the next city

962             while done == 0:
                    if i == 0:
964                     if p <= probabilitylist[i]:
                            next = [unvisited[i],0]
966                         done = 1
                        elif p <= probabilitylist[i+1]:
968                         next = [unvisited[i],1]
                            done = 1
970                     elif p <= probabilitylist[i+2]:
                            next = [unvisited[i],2]
972                         done = 1
                        elif p <= probabilitylist[i+3]:
974                         next = [unvisited[i],3]
                            done = 1
976                 elif i == 1:
                        if p <= probabilitylist[i+3]:
978                         next = [unvisited[i],0]
                            done = 1
980                     elif p <= probabilitylist[i+4]:
                            next = [unvisited[i],1]
982                         done = 1
                        elif p <= probabilitylist[i+5]:
984                         next = [unvisited[i],2]
                            done = 1
986                     elif p <= probabilitylist[i+6]:
                            next = [unvisited[i],3]
988                         done = 1
                    elif probabilitylist[i*4-1] < p <= probabilitylist[
                        i*4]:
990                     next = [unvisited[i],0]
                        done = 1
992                 elif probabilitylist[i*4] < p <= probabilitylist[i
                        *4+1]:
                        next = [unvisited[i],1]
994                     done = 1
                    elif probabilitylist[i*4+1] < p <= probabilitylist[
                        i*4+2]:
996                     next = [unvisited[i],2]
```

```
                    done = 1
                elif probabilitylist[i*4+2] < p <= probabilitylist[
                    i*4+3]:
                    next = [unvisited[i],3]
                    done = 1

                i += 1
            if len(unvisited) <= (self.numbind-2):
                if type(tabulist[-2]) == list:
                    #enters in front, have to use 2or3
                    if tabulist[-2][1] == 0 or tabulist[-2][1] ==
                        2:
                        if next[1] == 0:
                            next[1] = 2
                        elif next[1] == 1:
                            next[1] = 3
                            #enters in back, have to use 0or1
                    elif tabulist[-2][1] == 1 or tabulist[-2][1] ==
                         3:
                        if next[1] == 2:
                            next[1] = 0
                        elif next[1] == 3:
                            next[1] = 1

        if type(next) == int:
            print 'problem'
        return next[0],next[1]

    '''Find the sum of equation used to determine the probability
        of choosing an edge'''
    def findsumeq4(self, unvisited, current):
        sum = 0.0
        #find the sum (equation 4, page 32 Dorigo 2006 Ant Colony
            Optimization)
        for i in unvisited:
            for j in range(len(self.dismat[current][i])):
                if current != i or type(self.dismat[current][i]) ==
                     list:
                    try:
                        if self.dismat[current][i][j][0] == 0:
                            eta = (1.0/0.01)
                        else:
                            eta = (1.0/self.dismat[current][i][j
                                ][0])
                    except:
                        print 'Error in findsumeq4 method'
                    try:
                        tau = self.dismat[current][i][j][1]
                    except:
                        print '-...'
                    sum = sum + tau**self.alpha*eta**self.beta
        if sum == 0:
            print 'stop!'
        return sum

    '''Add pheromone concentration of 1 to all edges, used during
    initialisation.'''
    def addpheromone(self):
        for i in range(len(self.dismat)):
            for j in range(len(self.dismat[i])):
                if type(self.dismat[i][j]) == list:
                    for k in range(len(self.dismat[i][j])):
```

```
1052                            if type(self.dismat[i][j][k]) == list:
                                    self.dismat[i][j][k] = [self.dismat[i][
                                        j][k][0], 1]
1054                            else:
                                    self.dismat[i][j][k] = [self.dismat[i][
                                        j][k], 1]
1056 #Used for testing
     #ga = GeneticAlgorithm('c:\gcode.txt')
1058 #ga.getstuff()
     #ga.getsimpledata()
1060 #ga.master()
     #ga.mutate([0,1,2,3,4,5,6])
1062 #acs = AntColonySystem()
     #acs.getstuff()
1064 #acs.start()
     #TestAnt(100, 'C:\gcode.txt', ants=10, alpha = 1.0, beta = 1.0, Q =
         1.0, rho = 0.7,
1066 #generations = 50)
     TestGA(1, 'C:\gcode.txt',pc=0.6,pm=0.1 , populationsize = 200,
         generations = 500, q = 0.5)
```

Biological.py