Saimaa University of Applied Sciences
Faculty of Technology, Lappeenranta
Degree Programme in Mechanical Engineering and Production Technology

Luka Ivanovskis

Four Axis Hot-Wire Foam Cutter Controlled by Mindstorms EV3

Thesis 2017

Abstract

Luka Ivanovskis
Four Axis Hot-Wire Foam Cutter Controlled by Mindstorms EV3, 47 pages, 3 appendices
Saimaa University of Applied Sciences
Faculty of Technology, Lappeenranta
Degree Programme in Mechanical Engineering and Production Technology

Thesis 2017
Instructor: Lecturer Jouni Könönen, Saimaa University of Applied Sciences

The purpose of the study was to develop a 4-axis numerically controlled hot-wire foam cutter suitable for flying aircraft modelling purposes. Numerically controlled foam cutter had to be able to cut intricate shapes out of foam material such as extruded polystyrene by using 3D virtual model.

The study reviewed existing foam cutters built by modellers all around the world and set priorities for a new system, such as usage of common 3D format for input and common programming language for software. Unconventional mechanical design for the foam cutter was realized with Lego Mindstorms Ev3 parts. C++ language was used to program the Ev3 brick. A new algorithm was developed for approximation of triangulated models in STL format with ruled surfaces that are possible to cut with a straight hot wire.

The built foam cutter and its software were tested for production of fuselage of an airplane. The machine did not reach sufficient accuracy of cut, but showed a wide range of capabilities for cutting of sophisticated shapes. Questions covered in the study may be helpful for those who wants to built a numerically controlled machine, in particular, the foam cutter.

Keywords: hot-wire foam cutting, laminated object manufacturing, Lego Mindstorms Ev3, numerical control, ruled surfaces

Table of contents

Appendices

Terminology

3D – Three Dimensional

ABS – Acrylonitrile Butadiene Styrene

CAD – Computer Aided Design

CAM – Computer Aided Manufacturing

CPU – Central Processing Unit

DC – Direct Current

DIY – Do It Yourself

EPS – Expanded Polystyrene

EPP – Expanded Polypropylene

FPU – Floating Point Unit

GPU – Graphical Processing Unit

LOM – Layered Object Manufacturing

NC – Numerical Control

RAM – Random Access Memory

RC – Radio Control

RP – Rapid Prototyping

STL – STereoLithography (file format)

XPS – Extruded Polystyrene Foam (Styrofoam)

1 Introduction

1.1 Motivation behind the Study

Flying aircraft modelling as a hobby has undergone significant development in the 21st century. This occurred because of miniaturization of power and radio electronics, increase in power density for both electric motors and batteries as well as availability of strong and lightweight and easy to process foamed polymer materials: EPP, EPS, XPS. Many companies produce RC model bodies from these materials with intricate shape either for advanced aerodynamics or resemblance to famous airplanes (see Yak-130 model launched by FMS company in 2017 (1)).

Figure 1 Yak-130 by FMS company (1)

Manufacturers use foam moulding process. This process is well suited for mass production, since one high quality matrix can be used to produce many identical parts, like fuselage or wings (in case of RC models). Hobbyists that prefer to make their models at home with hand tools are usually limited to much simpler shapes that are obtained via bending plain Styrofoam sheets around multiple cross-sectional elements (see partly assembled Concorde model by Alexander Degtyarev (2))

Figure 2 Concorde model by Alexander Degtyarev (2)

Other modellers adopted layered object manufacturing (LOM) technique used for rapid manufacturing. They build their models from thick (approximately 30-100 mm) layers of Styrofoam that were cut with hot wire and glued together to form the intended object. Hot wire (heated with electric current) must follow precisely contours of cross sections which is achieved with templates glued on both sides of each layer (see FliteTest guide on how to build a model of Viggen plane (3))

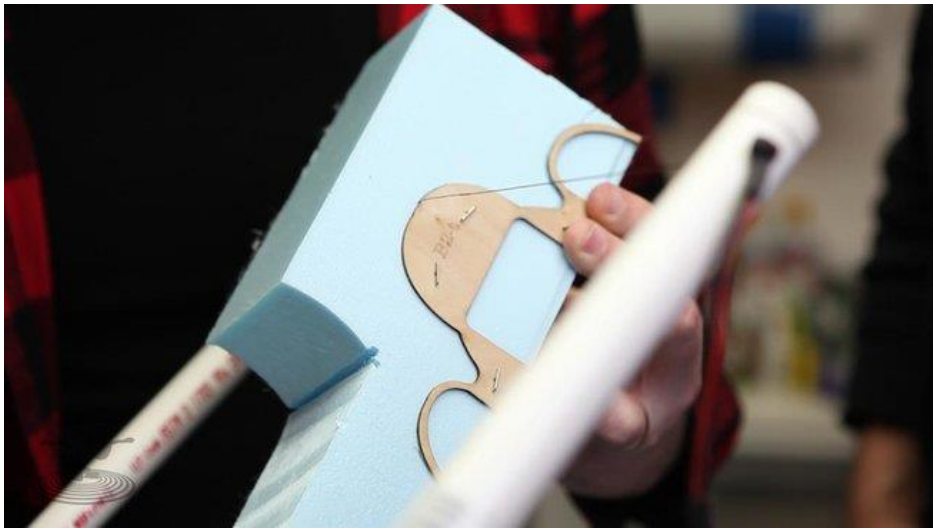


Figure 3 FliteTest. Hot-wire cutting (3)

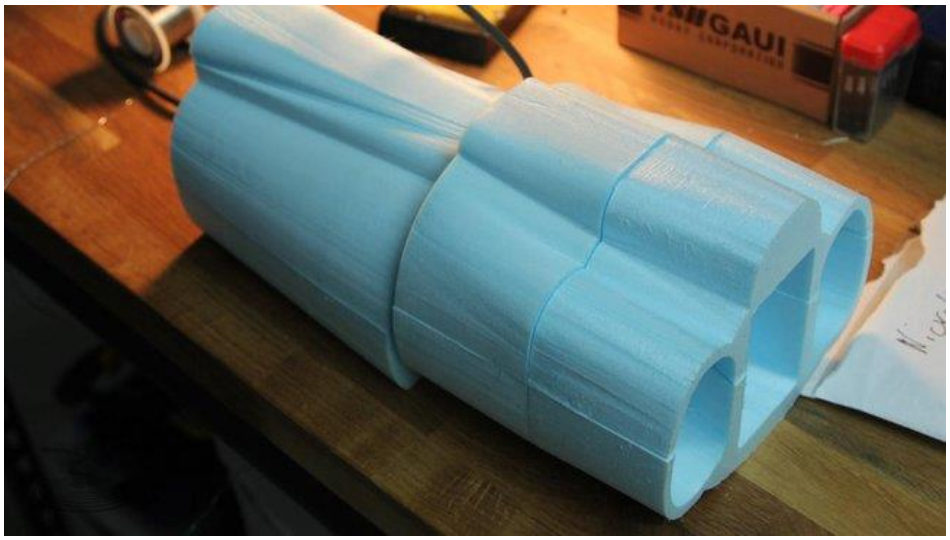


Figure 4 FliteTest. Hot-wire cut layers (3)

Again, it requires much skill and labour in order to achieve the desired result. The current research is an attempt to build a system that can serve similar needs (but not limited to) of flying aircraft modelers.

1.2 Numerical Control in Modelling

Procedure described earlier for manual building of a Styrofoam body suffers from several drawbacks:

- Precise templates from a hard material (like laser cut plywood) are needed, they should be firmly attached to flat surface of the material
- In aircraft design the weight is crucial, but making cross sections hollow is quite challenging
- Manual propulsion of hot wire is not smooth, synchronicity of following both contours may be far from ideal

There were several attempts to bring modern technology into this field. A member of RCGroups forum shared his experience in building a numerically controlled hot wire cutter that uses devWing Foam software suited for cutting long one-piece wings (4).

Figure 5 RCGroups. Hot-wire cutter (4)

Hans Seybold from Germany has made a NC hot wire cutter with 4 degrees of freedom. He used a unique mechanical layout where a bow with hot wire is positioned in space with 4 filaments of variable length controlled by 4 stepping motors. He also developed a program to drive the bow along a desired route which uses plain text description of all coordinates of the shape. All information is available for free on the internet at (5).

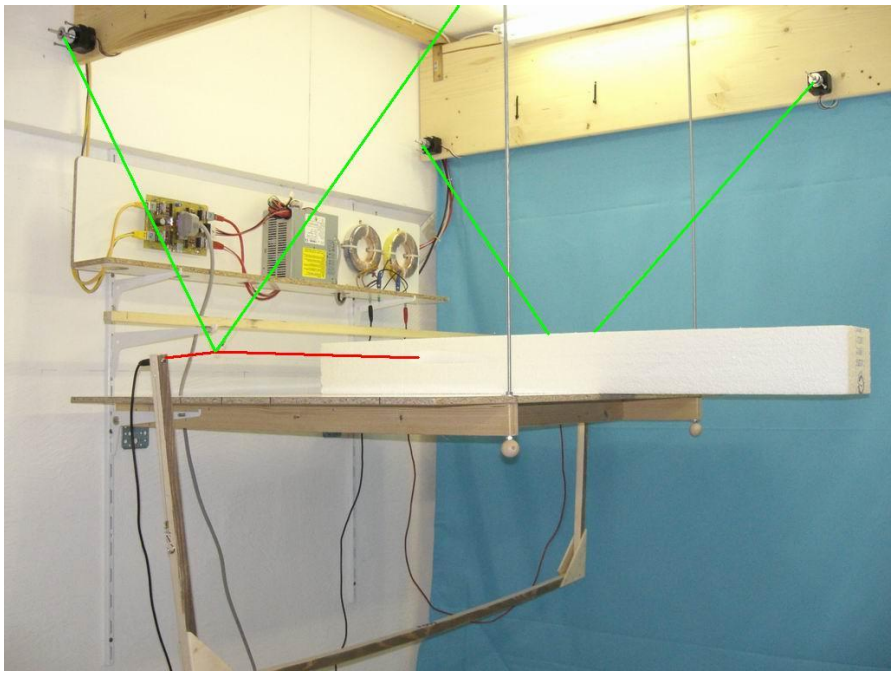


Figure 6 Hot-wire cutter by Hans Seybold (5)

Probably the most advanced technology is used in AeroTetris company located in Russia. They use a 6 axis NC machine which allows to achieve accuracy of 0,09-0,5 mm with wire inclination up to 165 degrees. During cutting process speed and temperature are varied. The company sells sets of Styrofoam parts for building large models (up to 4 metres wingspan) with careful exterior and interior design (suited for installation of impellers). See example below (6).

Figure 7 Starfighter from Aerotetris (6)

Figure 8 Parts for Starfighter from Aerotetris (6)

All opensource solutions require large working space and manipulate with cross sections (not 3D models).

1.3 Priorities for a New System

Several goals were set for a newly designed system to bring CAM technology to the hot wire foam cutting:

- Try to use capabilities of CAD design software for 3D modelling
- Avoid new formats, prefer existing ones
- Use commonly used programming language
- Use commonly accessible parts

The last point suggested LEGO Mindstorms Ev3 as a universal platform for creation of both the hardware (mechanical) and software part of the new system, since it provides structural elements (LEGO Technic), actuators (servomotors) and processing unit with open software (Ev3 Brick). All parts from LEGO can be reversibly assembled and disassembled many times which makes them particularly suitable for developing a new design. Once created in LEGO, the design can be rebuilt using standard parts for mechatronics – stepping motors, Arduino board etc.

2 Structural Design

2.1 Working Principle of Hot-Wire Cutting

There are two types of hot-wire foam cutters (7). The first group uses stiff preformed wire used to cut prespecified profiles. Currently there is a research about bending such a wire or strip during the cutting process in order to produce convex or concave shapes out of one solid foam block (8).

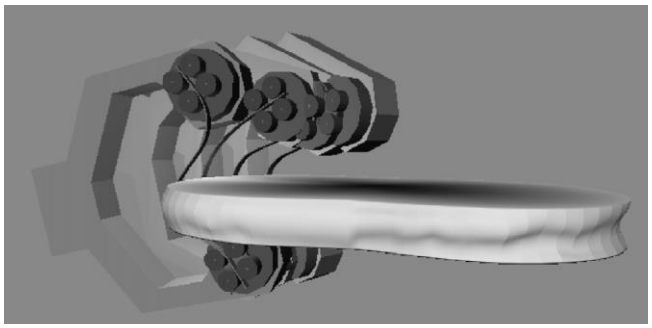


Figure 9 Cutting head with a hot strip (8)

The other type (which is considered in this work) uses thin taut wire that remains straight during cutting. The wire is heated with current, therefore, the wire should have high internal resistance as electric conductor and good corrosion resistance at elevated temperature. A typical material for the wire is stainless steel or ni-chrome. Hot-wire cutters are offered both as hand tools (usually there is a bow that stretch the wre) or as numerically controlled machines that are able to pro-duce complicated 3D shapes. Unlike CNC milling, the hot wire cutting has ad-vantage in speed and effectiveness of material removal. The finite diameter of a hot wire (starting from 0,2 mm) produces a gap during the cutting known as kerf. Kerf width is a function of feed rate, temperature, wire diameter and material properties such as heat conductivity, heat capacity, melting point (9). The kerf width can be reduced by using smaller wires, however the kerf is often much wider than the wire itself due to the thermo-mechanical nature of hot wire cutting. Cutting occurs below the melting point of the material (material just softens be-cause of its thermoplastic nature) and requires application of force in the direction of cutting to intensify the heat input and to go through viscous softened material. Any change in feed rate affects the kerf width and the cut surface. When a hot wire is held stationary for a second it will produce a round hole in the material which may be one order of magnitude bigger in diameter than the wire itself (10).

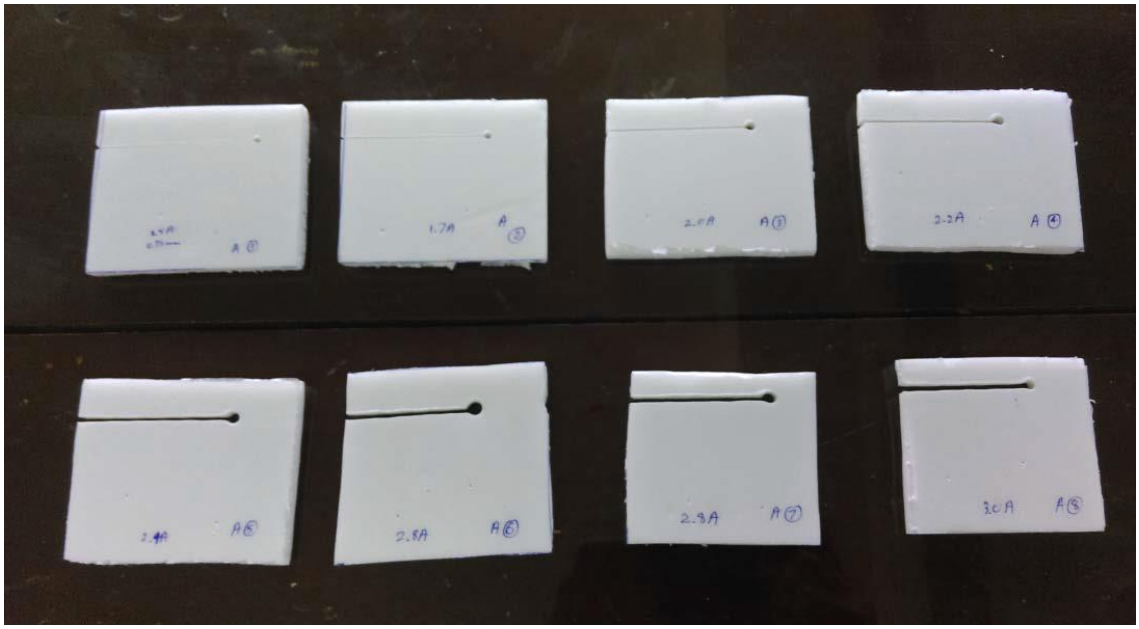


Figure 10 Test cuts with different temperature and speed (10)

A professional CNC hot-wire cutter has 4 axis freedom which is enough to position straight wire in relative to the piece of material. Rigid frame serves as a base for 2-axis positioning devices consisting of steeping motors and orthogonally organized lead screws and guide rails on the opposite sides of the machine. Cutting force acting perpendicular to the wire may bend it easily, so, to prevent this, the wire is under high tension provided by springs that are capable to adopt to wire thermal extension and varying distance between wire attachment points when the wire goes diagonally from one side to another.

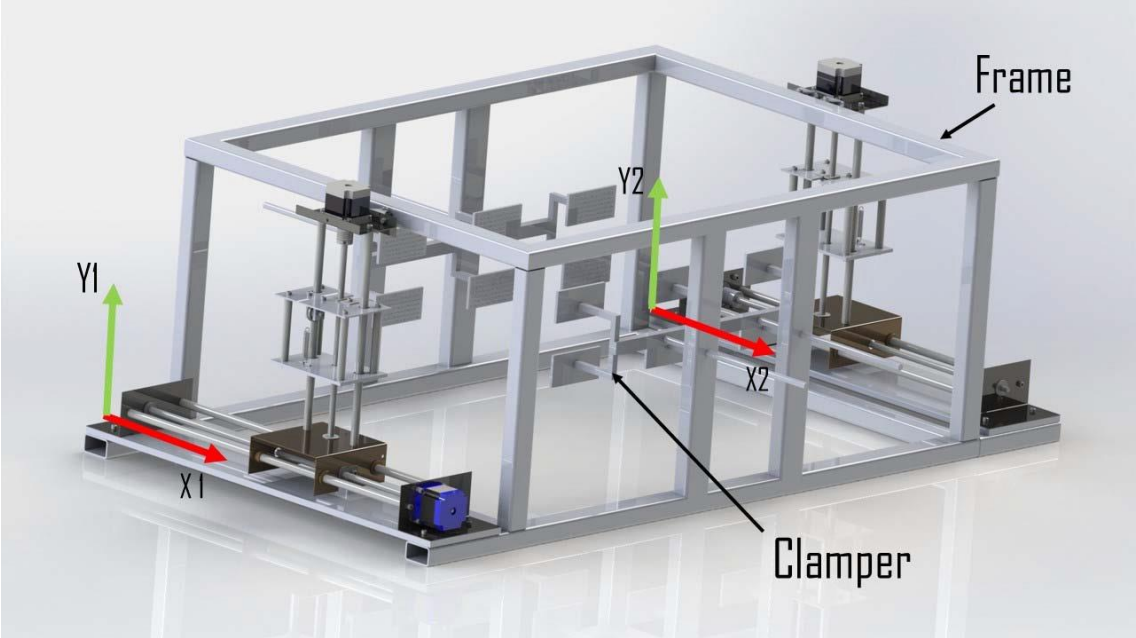


Figure 11 Professional 4 axis foam cutter (10)

Advanced software in addition to the following of the specified shape may provide necessary offsetting for kerf width compensation as well as wire temperature control. As all NC machines, NC hot wire cutters use G-code file that specifies cutting parameters and lists all coordinates to be gone through. Each line contains, respectively, four coordinates (*x* and *y* for both sides). The geometry of the machine must be made known to the software in advance.

2.2 Examples of Numerically Controlled Machines in LEGO

There is evidence of many experiments with NC implemented with LEGO Mindstorms platform, either NXT or Ev3. The NXT computer brick has 3 ports for motors and 4 for sensors, which correspondingly limits the amount of degrees of

freedom. One example is a 3-axis milling machine by Arthur Sacek which can mill 3D shapes from light material called floral foam using virtual model from the Autodesk Softimage software (11).

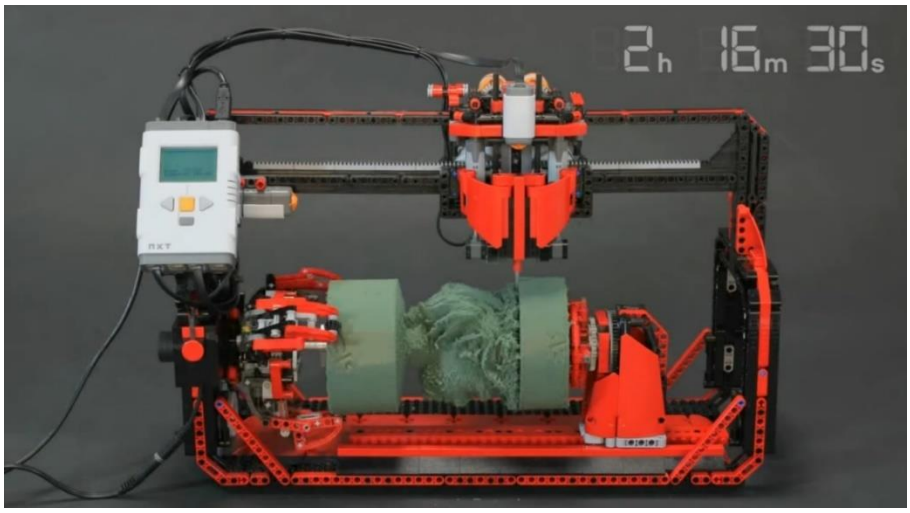


Figure 12 LEGO milling machine (11)

LEGO is held in esteem also in industrial companies like ABB Robotics. Their employees recreated with LEGO parts a programmable 6-axis robotic manipulator designed in ABB (12). They faced problem with ensuring structural stiffnes of the whole robot, but the end result mimics all capabilities of the real prototype. The model uses two Ev3 bricks, each of them has 4 ports for motors and 4 for sensors.

Figure 13 ABB robot from LEGO (12)

Finally, there were attempts to build a hot-wire foam cutter out of LEGO parts. Existing designs have only 2 axis freedom, like the one shown at (13).

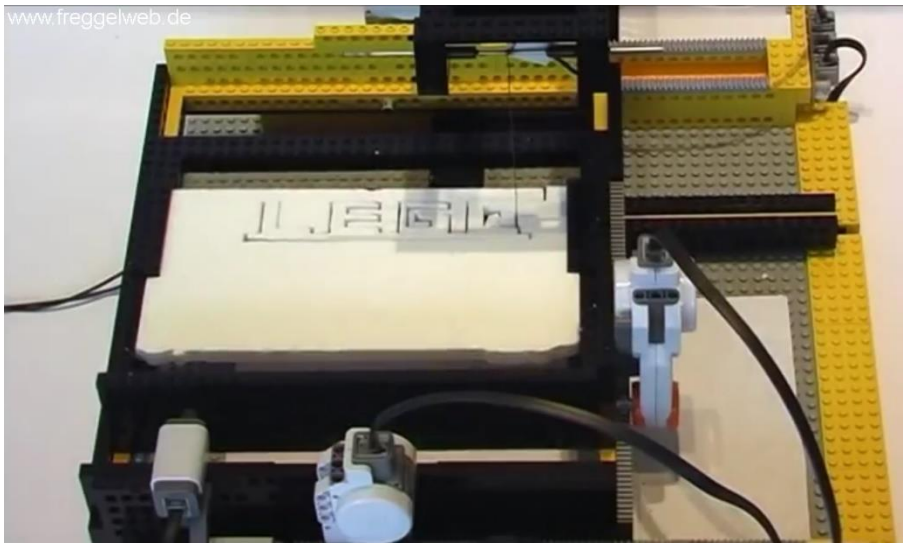


Figure 14 LEGO foam cutter (13)

2.3 The First Prototype

Limitations of LEGO Technics parts that strongly affect design are the following:

- Material – ABS plastic has approximately 10 times lower strength(ultimate tensile strength 40MPa) and 100 times lower stiffness (modulus of elasticity 1,4-3,1 GPa) than steel or aluminium (14)
- Softening point at 105°C, LEGO parts should be isolated from the hot wire
- Longest solid structural element is only 120 mm long
- LEGO obviously does not have nichrome wire and power supply for that

While cutting forces are quite low, the wire must be strongly tensioned. Moreover, unlike mill, laser or water jet, the wire must be held from both sides of the material. Closed structure of the machine limits dimensions of piece of material that can be processed. Building 500-600 mm large closed frame for handling standard foam sheets (that come in this size) seemed impractical, so preference was given to the open structure (C-frame) that would be able to reach 200-250 mm from the boundary of a sheet and make usable all of its area when approaching it from one side and then from the opposite. Another idea for extending the available working space while keeping the machine compact was to let the machine move

(ride on a table) while keeping material stationary using a suitable clamp (depending on the dimensions and the shape of a piece). With this in mind, the first prototype was built.

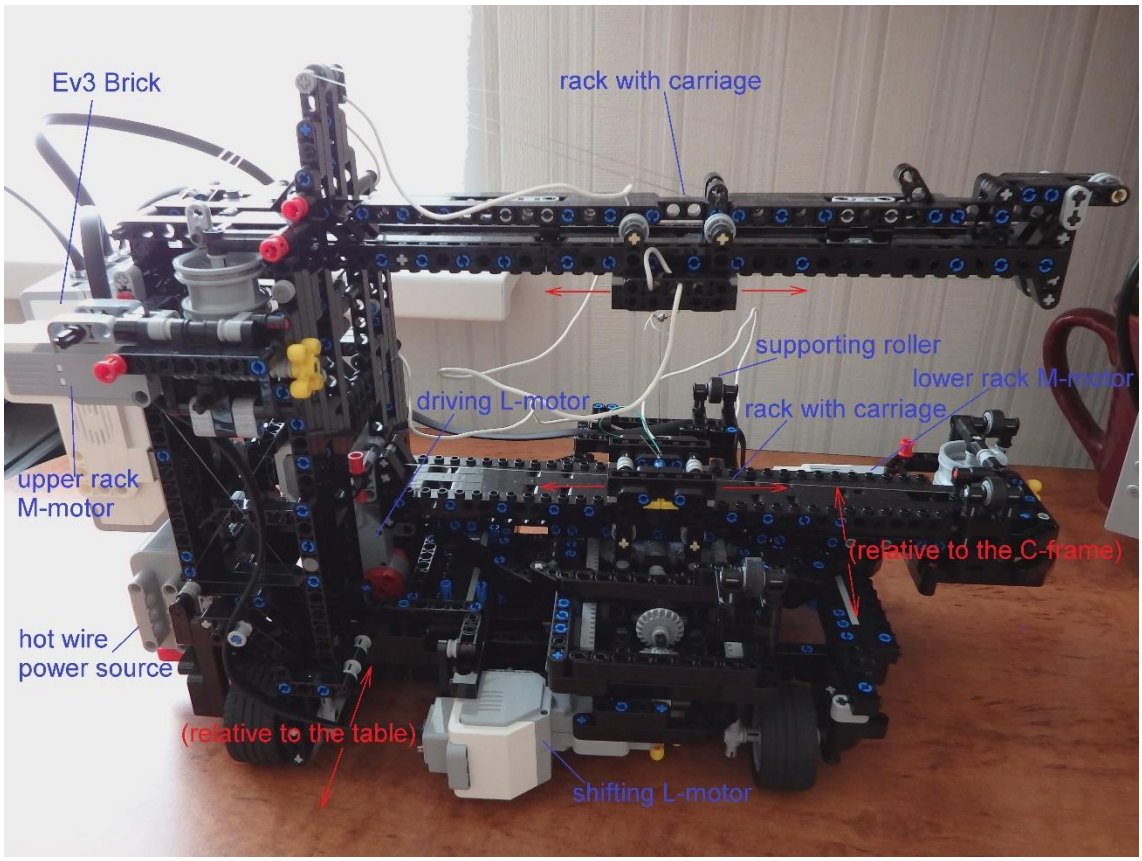


Figure 15 The first model of the 4-axis foam cutter

The C-frame was supported by five wheels and could ride on a table under the control of L-servomotor (large) from LEGO Mindstorms Ev3, allowing to cut arbitrarily long pieces. The upper arm of the C-frame was formed by a 25 cm long rack with a carriage sliding along it and holding the hot wire. This carriage was driven by the M-servomotor (medium). The position of the carriage on the rack was defined by a taut nylon cord (0,4 mm fishing rod) that was winded on a double coil. When the coil rotated, the cord was unwinded on the one side and winded on the other, as consequence, the carriage moved.

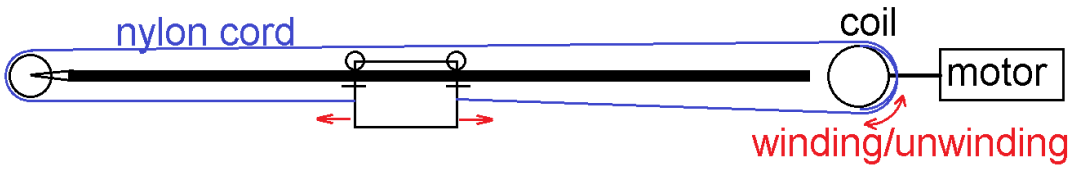


Figure 16 Carriage mechanism

A similar rack was arranged on the lower arm of the C-frame, but this could be shifted with the help of rack and pinion drive across the basement. The carriage was driven by another M-motor, while the rack – by the L-motor (large).

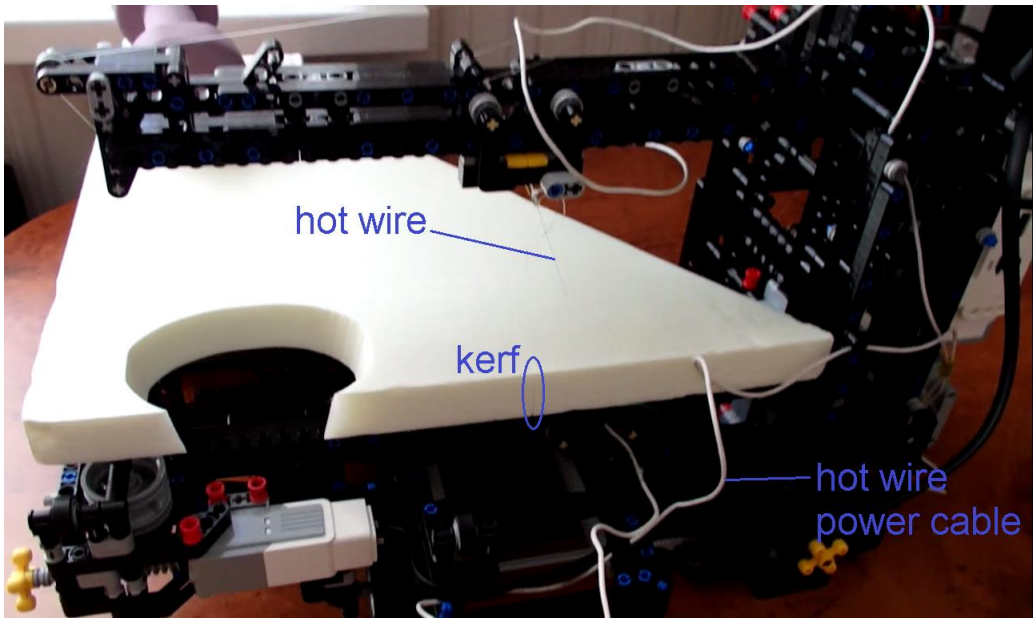


Figure 17 The first model is cutting XPS

To increase the torque and adapt the speed of motors, worm gearboxes were chosen. Worm gearing has an advantage over spur gears that it transmits torque from the input to the output and not backwards. It means that the mechanism equipped with worm gear will not move under load even without any power supply, although it has lower efficiency. Putting a worm gear as a last stage in each drive set helped to solve a very common problem of precise CNC machines – a backlash.

Backlash, or play is a clearance or lost motion in a mechanism caused by gaps between the parts (15). Two main strategies are used to overcome that:

- manufacture precise parts

- pre-load parts to eliminate backlash

The first option is not possible with LEGO, since all moving parts use loose fitting to lighten the motion. The second method is often realized in mechanisms by splitting gear into two gears – one half is firmly attached to the shaft and the other half of the gear is allowed to turn on the shaft, but pre-loaded in rotation by coil springs. The springs work in the direction to eliminate any backlash. In LEGO this method was realized by using a pair of worm gears in each gear set. With correct positioning worm gears were pressed against the housing by the teeth of the spur gear.

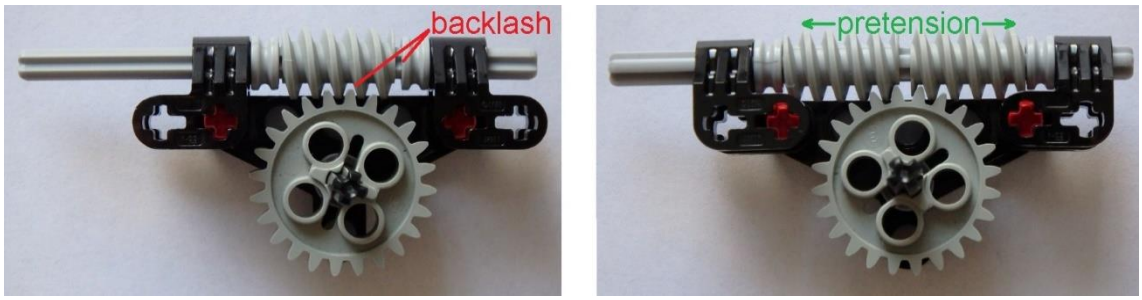


Figure 18 Backlash elimination in LEGO worm gearbox

A unique wire tightening mechanism was implemented. The lack of available space made it impossible to use springs, so gravity was used to keep the wire under tension. The Ev3 Brick with battery weighs very close to 300 grams and in combination with a system of pulleys it maintained constant tension in the wire regardless its position or slope.

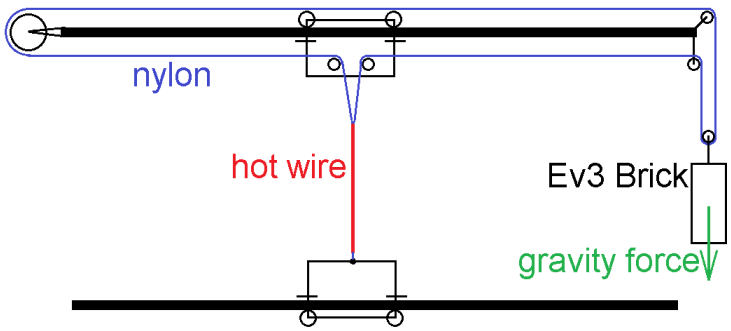


Figure 19 Tensioning mechanism

A very primitive program was written to test the machine in action. The machine was programmed to cut a truncated cone out of 20 mm thick XPS (extruded pol-

ystyrene). The 0,16 mm wire was heated with 6 AA batteries (current 0,7 A, temperature 140-150°C). While the machine proved viability of the foam cutter with open structure, it had significant drawbacks that led to the following conclusions:

- tension of the wire should be increased
- wire tension creates significant deflection (skew) in the C-frame. The deflection is not constant, it varies (up to 1 cm) depending on the position and the slope of the wire
- inclined wire caused high load on motors that were moving the carriages
- the sheet of material must be held on a constant height more tightly
- nylon cordage was too much complicated – once the hot wire was overheated and nylon melted, it took more than one hour to make a new wiring inside the C-frame

These conclusions have led to the creation of a new version of the foam cutter.

2.4 Better Model

Deflections in the structure is a known problem of CNC machines. They may cause notable deviations from the programmed path, so the software may include specific error compensation module (16, p. 157). Such a module requires a detailed physical model of the machine which is excessively complicated for a DIY(do-it-yourself) project.

Instead, by changing the mechanical layout of the machine, deformations were made practically constant and not affecting neither wire position nor servomotors. An 18 cm long wire was tightened in a bow with parallelogram structure. The longer wire made it possible to cut shapes like tapered airplane wings. Joints that held the bow allowed to incline the wire up to 40 degrees from vertical. The tension of the wire was not affecting servo motors anymore, they had to overcome only friction and unbalanced weight of the bow, which was partly compensated by a counterweight at the bottom.

The concept of the riding machine was turned upside down, making sheet of material to slide on the table under driven wheels held in place at adjustable height depending on material thickness. Silicone paper between the surface of a table

and foam sheet reduced friction. Rack-and-pinion drive was used for tilting the bow. Backlash was eliminated by using two shifted racks in each gear set.

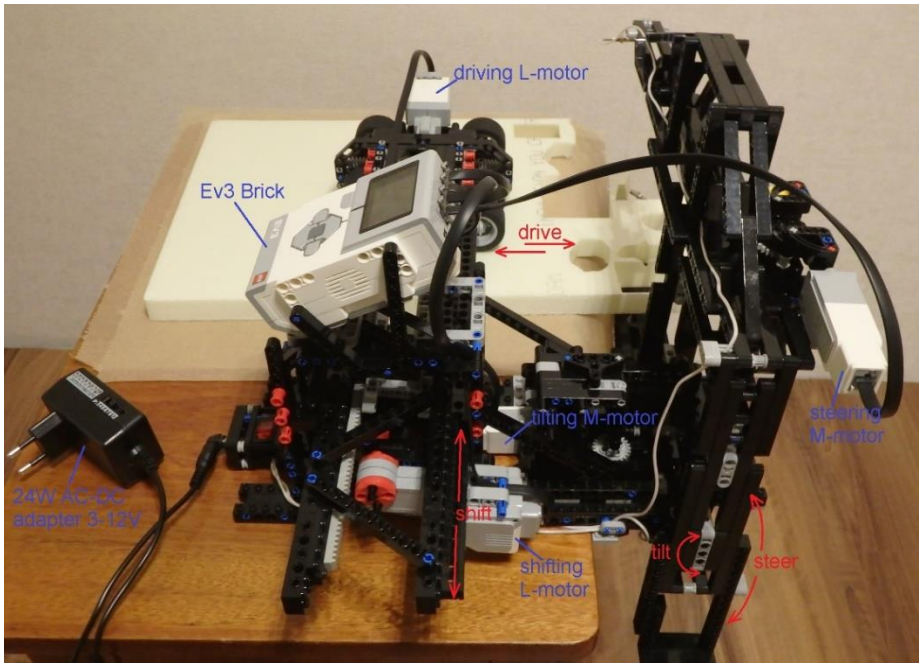


Figure 20 The second model of the 4-axis foam cutter

The hot wire was powered by a universal charger – 24 W power adapter that provided a range of voltages: 3; 4,5; 5; 6; 7,5; 9; 12V. Finer tuning of working temperature was possible via appropriate placement of crocodile clips on the wire (different length under the same voltage means different current and temperature, as consequence).

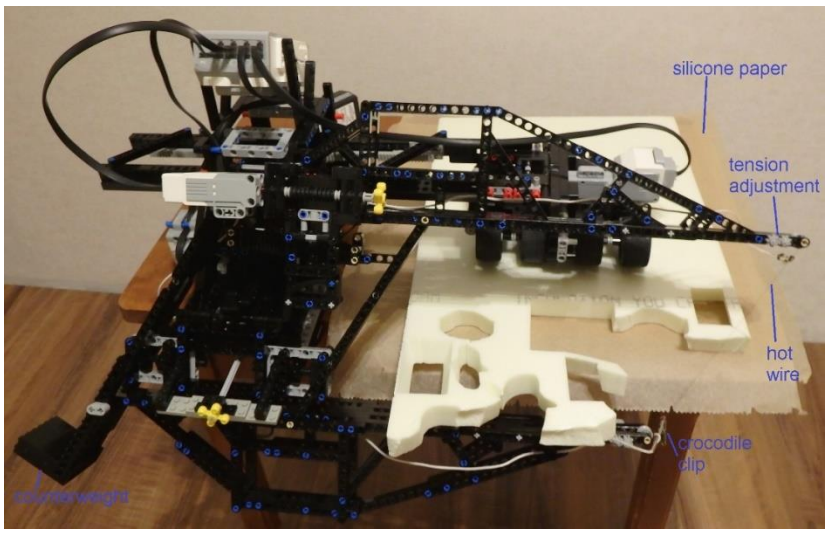


Figure 21 Inclination of the hot wire

The new machine has lost compactness of the previous model and also had issues with stiffness of the bow and arm that feeds the material, but appeared a lot more convenient in use.

3 Algorithms for Ruled Surfaces

3.1 Ruled Surfaces

The cutting tool of a foam cutter remains always straight, therefore the surface produced in a single pass belongs to the so called ruled surfaces. According to the definition (17), a surface is ruled if through every point of this surface there is a straight line that lies on it. In addition to trivial examples like plane, cylinder or cone (as well as their variations), ruled surfaces may be doubly curved like hyperbolic paraboloid with equation $z = xy$.

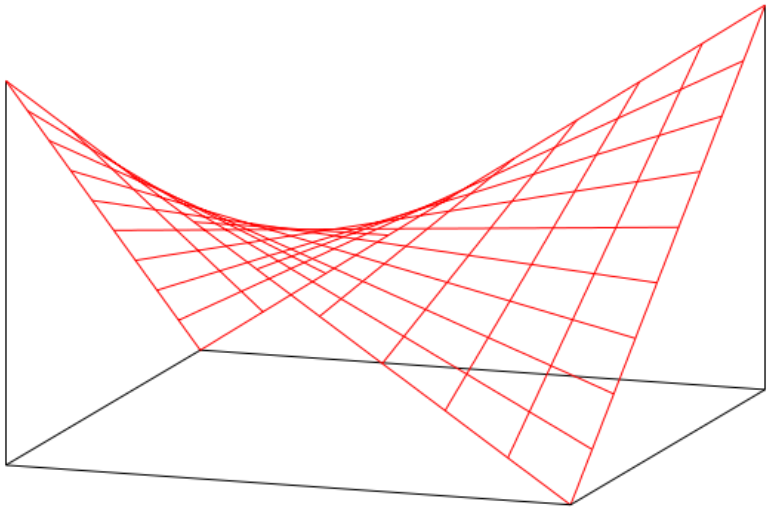


Figure 22 Hyperbolic paraboloid (17)

Ruled surfaces are often described as a set of points swept by a moving straight line, which, in turn, is defined by two points moving along two spatial parametric curves called directrices. In case of foam cutter, these curves lie on either sides of the plate material to be cut.

Ruled surfaces play an important role in modern manufacturing because they can be produced by a single pass of a straight tool, thus allowing to produce curved surfaces effectively. Approximation with ruled surfaces is used to create tool

paths in manufacturing of impellers (18). In this method a 5-axis CNC milling machine may use a cylindrical or conical mill to produce a complicated 3D shape in a fewer passes instead of a conventional layer-by-layer material removal strategy. On the other hand, hot-wire foam cutting is used in for manufacturing of large-size moulds for concrete panels forming facades of modern buildings (19).

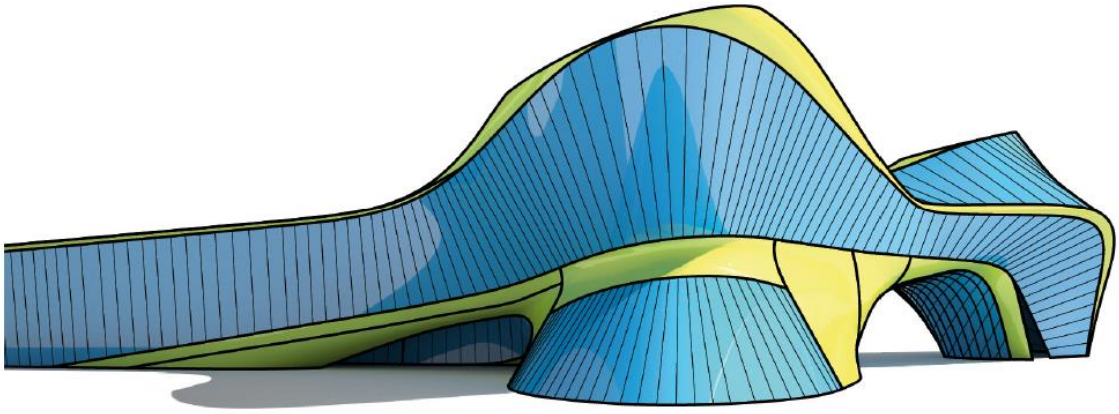


Figure 23 Facade rationalized with ruled surfaces (19)

The main difficulty in generation of a tool path for ruled surface is the choice of the right parametrization for both directrices. In this context, parametrization means definition which points of both directories are reached by the ruling simultaneously. Different parametrizations lead to different surfaces between the directrices. In case of circular directrices, possible surfaces range from cylinder or cone to paraboloid. This makes parametrization a non-trivial problem and requires a special algorithm to solve it.

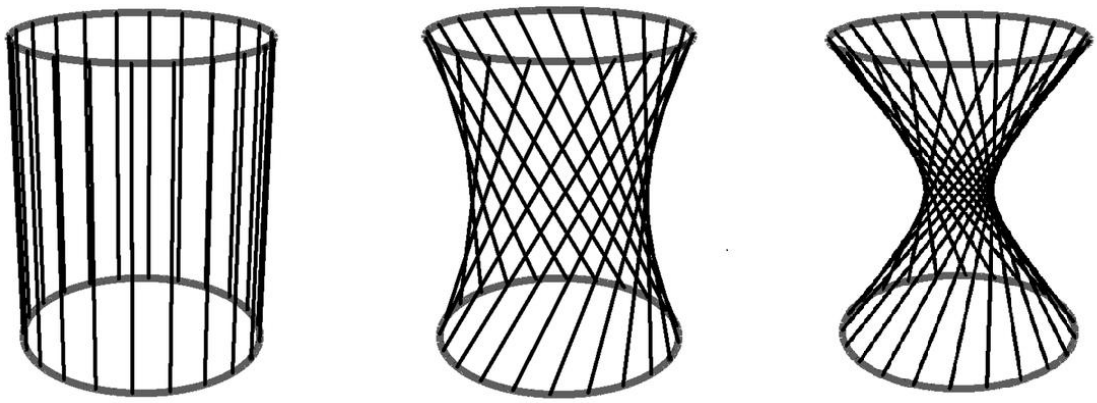


Figure 24 Twisted cylinder becomes a paraboloid (20)

3.2 Algorithms for 2D Contours

Simpler algorithms for parametrization of directrices neglect any information about 3D shape of the CAD model and use contours obtained when slicing model in layers for rapid prototyping (RP) from polystyrene foam. In RP, cutting sloped layers instead of plain 2D layers reduces surface error of produced object with the same layer thickness (21).

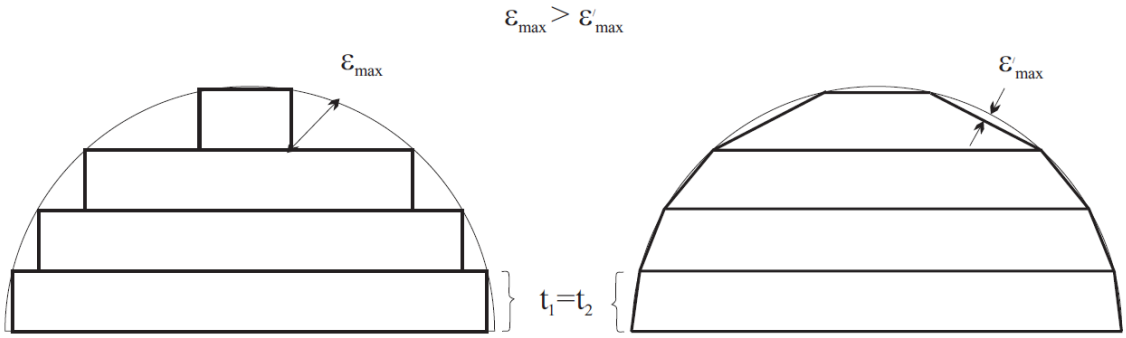


Figure 25 Surface error: 2D layers versus sloped layers (21)

Algorithms belonging to this type may use different objectives to optimize the parametrization. Examples for such objectives include (22):

- Minimal surface area
- Minimal twist
- Maximal convexity
- Minimal bending energy
- Minimal mean curvature variation
- Combination of several objectives

See Figure 26 for illustration.

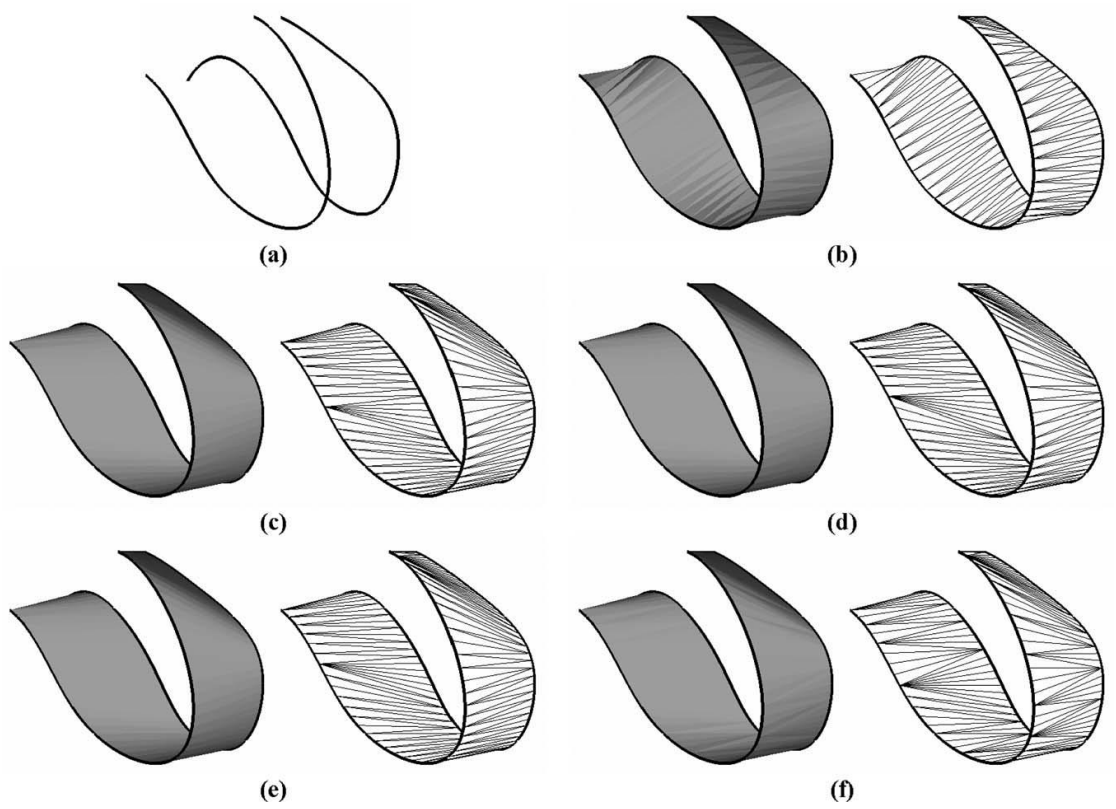


Figure 26 Surface triangulation for different objectives: (a) the directrices, (b) min. area (c) min. twist (d) max. convexity, (e) min. bending, (f) min. mean curvature variation (22)

RP machines often use .STL format files that store triangulated representation of the solid CAD model. Individual triangles (called facets) that make up the surface of the model are represented by *x, y, z* coordinates and facet normal vectors. Slicing triangulated model with planes produces polygonal contours that are the input directrices of parametrization algorithm. The algorithm spans then given contours with triangles. Every triangle has one vertex coincident with a vertex of one contour and opposing side coincident with an edge of another contour. When the first ruling line is defined by the ends of the contours, the successive triangle is obtained by making a step forward along either the first or the second contour. The sequence of such steps on both contours up to the last ruling defines a triangulated ruled surface between two contours. In order to assess quality of such a surface, a cost regarding optimization criteria is assigned to each triangle (like area, for example). As a result, the quality of the obtained surface corresponds to the sum of costs of individual triangles forming the surface.

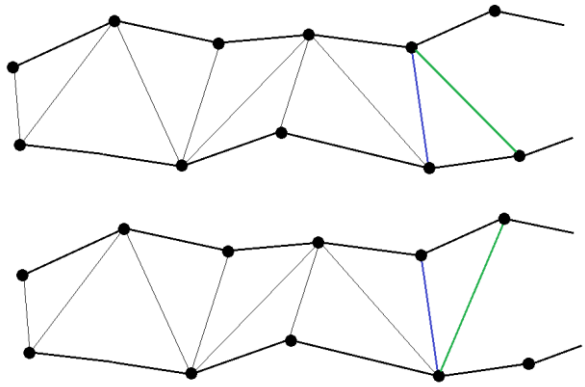


Figure 27 Possible advancement of ruling: on the lower or on the upper contour

The amount of possible triangle sequences grows exponentially with the number of contours edges, proportional to $\frac{(m+n)!}{m!n!}$, where m and n are the numbers of contour edges (23). However, the problem of finding an optimal surface may be reformulated as finding the shortest path in a graph, where graph vertices (or nodes) correspond to ruling lines between the contours and edges (or arcs) between the vertices correspond to possible triangles formed by two successive rulings. Each ruling then has a corresponding node and forming a surface from the first ruling to the last one means going in allowed directions (along the directed arcs) from node to node in the graph.

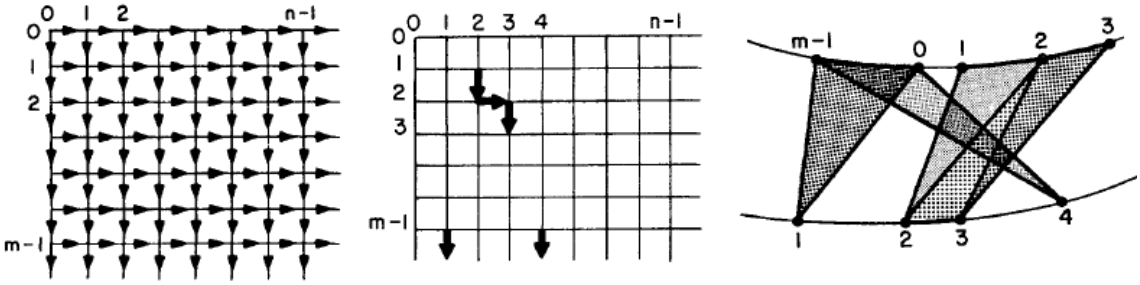


Figure 28 Graph formulation of optimization problem: allowed connections between nodes (left), correspondence between arcs and triangular patches(middle and right) (23)

The structure of the graph allows to use dynamic programming to find the best path between the starting node and the ending node. The method of dynamic programming breaks down a complex problem into a collection of simpler subproblems, solving each of those subproblems just once and storing their solutions (24). In the ruled surface fitting problem the general problem is to find the lowest cost to reach the last (target) node starting from the first one (the source). The

subproblem is, respectively, the lowest cost to reach an arbitrary node starting from the first one.

The algorithm runs as follows:

1. Weights (or costs) of all arcs are computed.
2. For every node a current estimate of cost is stored. At the beginning all the estimates are initialized with infinity, except the source for which it equals 0.
3. Nodes from left to right and from top to bottom try to improve estimates of the related nodes, presuming direction of the arcs (rightward and downward). From the current node the next node may be reached at cost of connecting arc, therefore the new estimate is the current estimate summed with the cost of arc. The new estimate is applied if it is better than the existing one.
4. At the end, the estimate of the target node corresponds to the best possible quality of the fitted surface.

Since the algorithm must return the whole sequence of rulings that lead to the target in an optimal way, for each node the best predecessor is stored, so that it is possible to reconstruct the optimum path and, returning to the terminology of directrix parametrization problem, the correspondence between the contours.

3.3 Algorithms for 3D Shapes

Although algorithms that work with contours are sufficient for RP or LOM purposes, they do not use capabilities of cutting ruled surfaces. Figure 29 illustrates how much can be achieved when appropriate direction of the cutting tool is chosen. In aircraft modelling the bodywork must withstand significant loads while keeping weight as low as possible. This is achievable by keeping integrity of bodywork with low amount of parts and adhesive bonds between them where mechanical stresses tend to concentrate because of different material properties. Therefore, a more advanced algorithm for the hot-wire cutter was required.

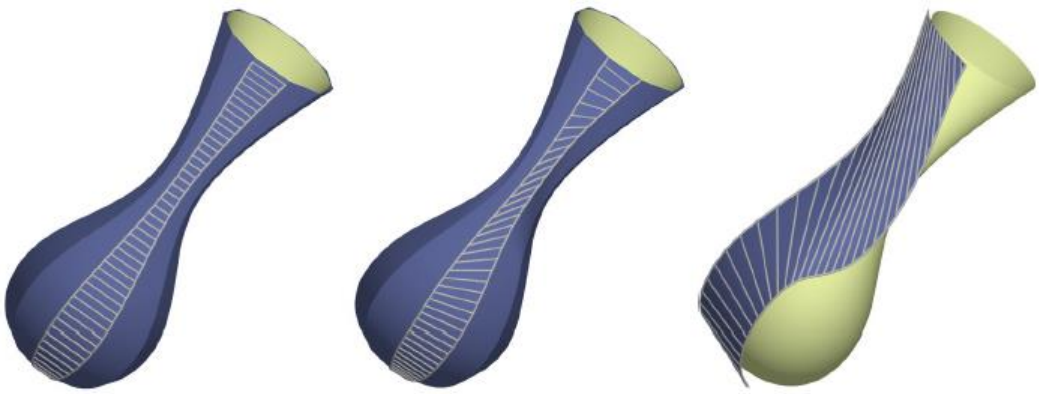


Figure 29 How tool inclination improves surface quality (19)

General algorithms for fitting ruled surfaces posess similar structure as the previously described algorithm for contour matching. Even though they use parametric representation of smooth freeform surfaces instead of triangulated ones as input, they also require some sort of discretization along contours defined by splines (Bézier type or others). Such discretization serves as a base for applying dynamic programming for finding best rulings connecting two contours. The aspect that changes is the objective for optimization. Now for each patch between two successive rulings the maximum distance between the patch and given smooth surface is computed or estimated (25). Another type of discretization is to find possible (discrete) directions of rulings that pass through numerous sample points and coincide well with the surface (26). Again, the optimal set is found by dynamic programming.

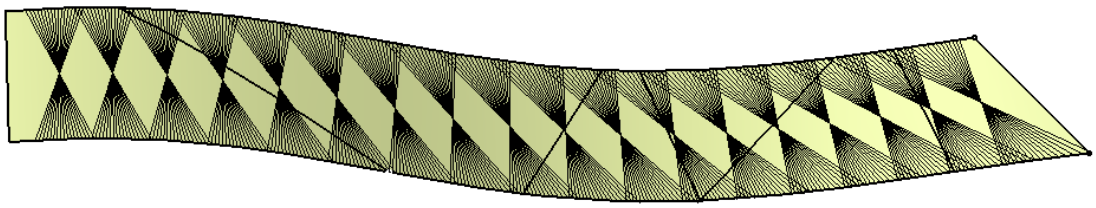


Figure 30 Example of candidate rulings (26)

The aforementioned algorithms have several drawbacks:

- They are computationally expensive (researchers even adapt them for running on GPU to increase the speed)

- They are difficult in realization for a DIY project (both parsing of CAD files with parametric surfaces and computations themselves are complicated)
- They do not address the problem of triangular patches

Triangular patches cause problems in hot-wire cutting because the cutting tool must remain stationary at the apex of triangle which creates a surface defect because material melts too far away from the wire. In the work (27) special trajectory correction is proposed which avoids stalling of the cutting tool, but it works only for convex shapes.

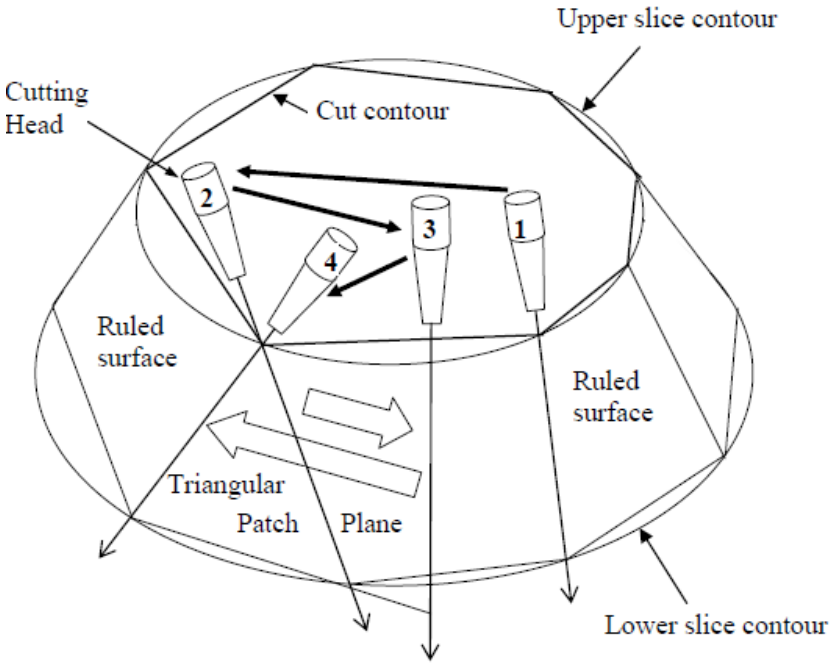


Figure 31 Tool trajectory correction to cut triangular patch (27)

3.4 Developed Algorithm

The algorithm that was developed specifically for the foam cutter uses triangulated surface representation for input and has a similar structure of those described in the previous sections, but introduces some new features.

Firstly, it searches edge to edge correspondence between contours instead of ruling lines connecting vertex to vertex. In other words, it searches for an optimal set of bilinear patches instead of triangular ones. Bilinear patches have an advantage that the cutting tool has non-zero speed at all points, but they overlap when it is needed to go for the next edge on one contour while staying at the

same edge on the other. The algorithm overcomes this issue by proportional subdivision of edges that belong to multiple patches. The edge is subdivided in the same proportion as the opposite edges of the patches.

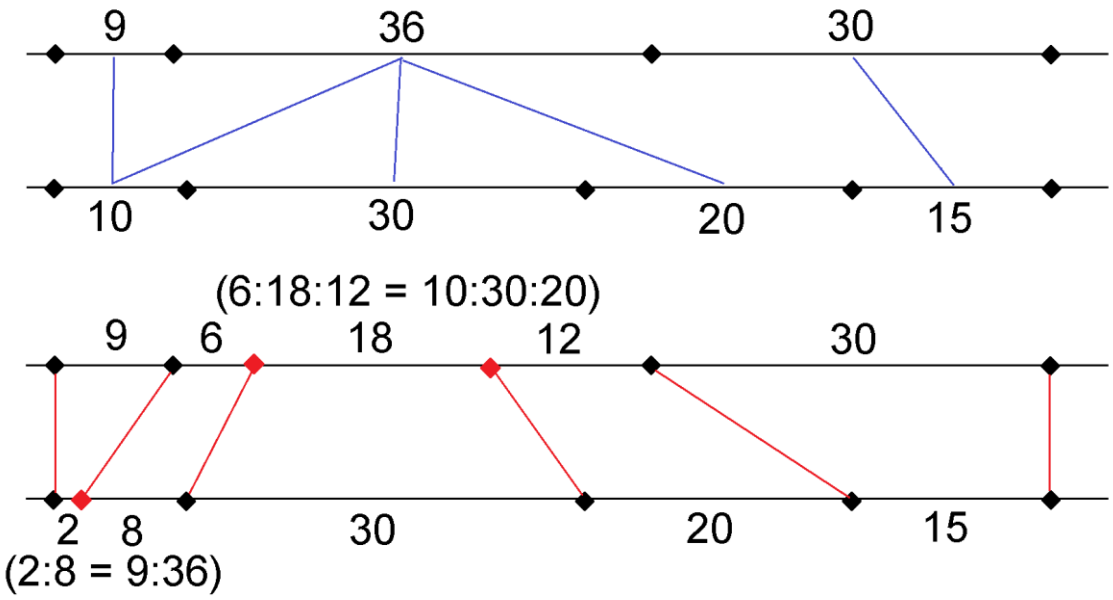


Figure 32 Patch interpolation rule: optimum overlapping patches (blue lines above), non-overlapping interpolated patches (red)

Secondly, the cost of possible patches is equal to the normal distance between the closest facet of surface and the centre point of the patch. The cost estimation algorithm takes four vertices of the patch ABCD and a candidate facet $\Delta V_1V_2V_3$ for distance computation. For the first candidate a facet is chosen that shares one common edge with the considered patch. For the patch the centre point H is determined (as average of A, B, C and D point coordinates) and unit normal vector **n**.

$$\boldsymbol{n} = \frac{\left(\overrightarrow{AB} + \overrightarrow{CD}\right)\times\left(\overrightarrow{AC} + \overrightarrow{BD}\right)}{\left|\left(\overrightarrow{AB} + \overrightarrow{CD}\right)\times\left(\overrightarrow{AC} + \overrightarrow{BD}\right)\right|} \quad (1)$$

Then, centre mass point C_M of the candidate facet is computed. In order to check whether the candidate facet is pierced (or is close to be pierced) by the normal vector **n**, the perpendicular distance d from the point C_M to the normal is computed using auxiliary vector **p**:

$$\boldsymbol{p} = \overrightarrow{HC_M} \quad (2)$$

$$d = |\boldsymbol{p} - (\boldsymbol{p} \cdot \boldsymbol{n}) \cdot \boldsymbol{n}| \qquad (3)$$

If the perpendicular distance d of the current candidate is lower than that of its neighbour facets (local minimum is achieved), then the search is stopped and the cost c is computed:

$$c = \boldsymbol{p} \cdot \boldsymbol{n} \qquad (4)$$

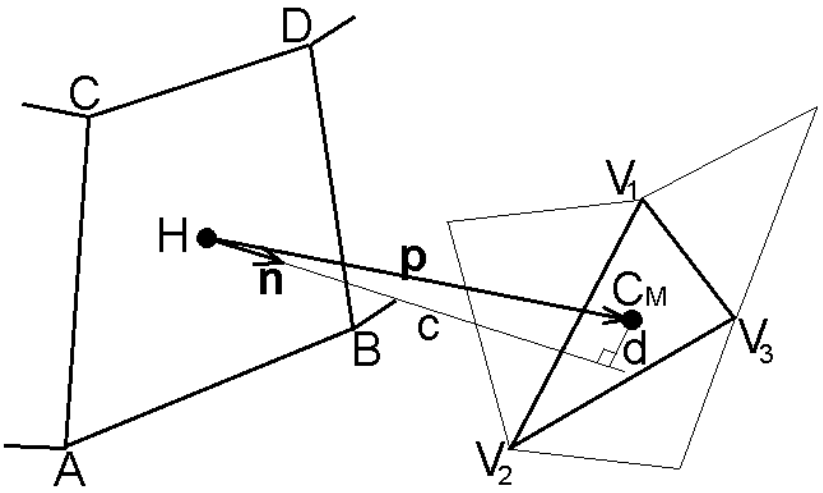


Figure 33 Patch cost estimation

The squared value of this cost (to get rid of sign and to make difference between optimal and bad patches) is stored for each viable patch for later dynamic programming. Checking patches for viability reduces the amount of possible patches by omitting those that are sloped too much and cannot be cut due to limited inclination of the hot wire.

Lastly, the algorithm works with closed contours because they are easy to extract from an .STL model of the surface without any reference where the surface should start and end. Therefore, there are multiple starting patches that should be treated as a source node in dynamic programming. Similarly, the target node must correspond to the same patch as the source.

C++ program was written that implemented this algorithm and visualized the results in .STL format. Since .STL format does not support other facets than triangular ones, each bilinear patch was represented with three triangles.

Different shapes were created in Solidworks 2016 software (student version) to test the algorithm. Solidworks provides all necessary tools for creation of input files: solid modelling, transforming solid bodies into surfaces by deleting unnecessary faces and converting surfaces into .STL format with adjustable accuracy. Microsoft 3D Builder was used for fast review of .STL files, both input and output.

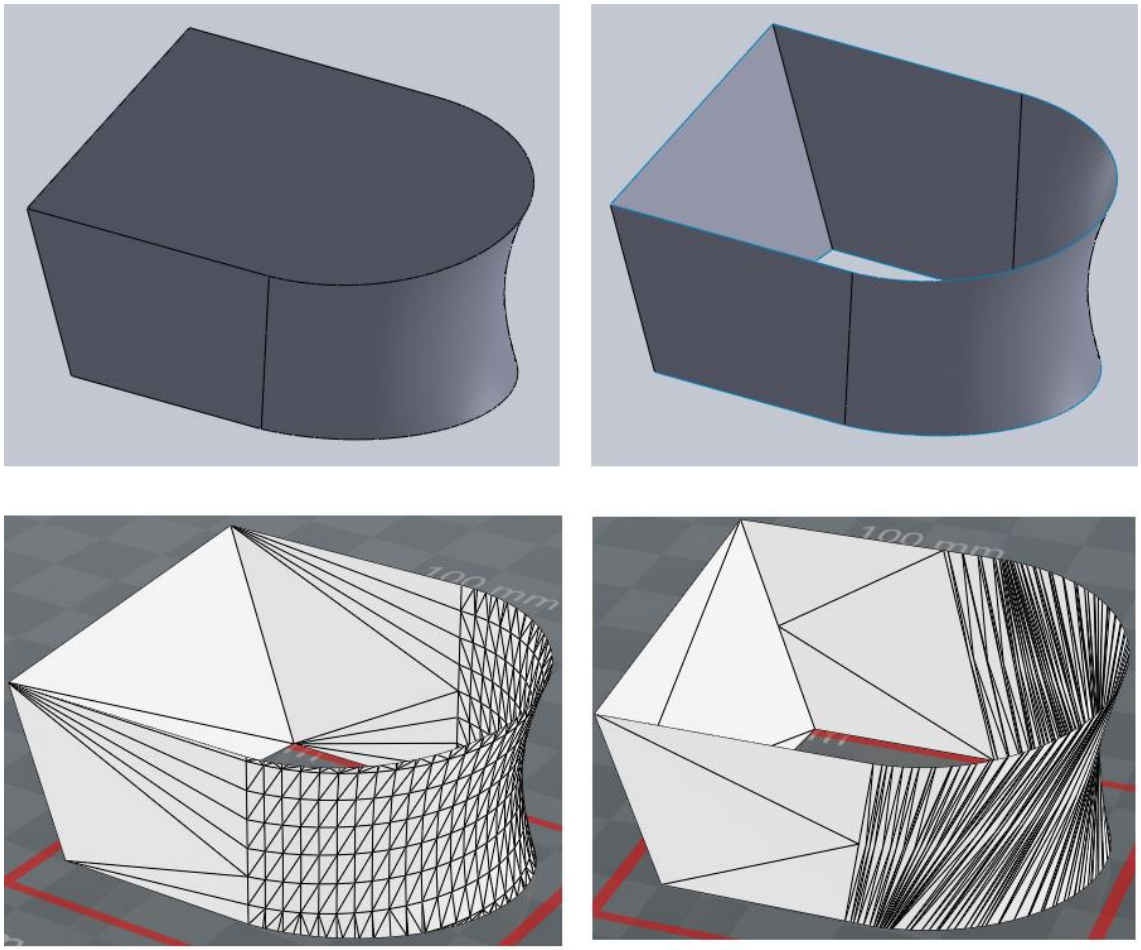


Figure 34 Test 3D shape: solid model, surface model, STL model, ruled surface

As can be seen, the algorithm was able to extract upper and lower contours from the shape and replace facets with bilinear patches that matched well both flat and concave regions of the test surface. More examples of free form surfaces are given in Section 5.

4 Programming LEGO Mindstorms Cutter

4.1 Tools for Programming Mindstorms Ev3

LEGO Mindstorms Ev3 programmable brick has a CPU with the working frequency of 300 MHz that runs Linux as an operating system. LEGO company provides a graphical environment to program robots, but it has a limited set of instructions that may be given to a robot. Several other operating systems were created, so that programmers could use Java language (in LejOS system) or Python, C and C++ (in Ev3dev) (28). This approach requires installation of a respective operating system on a MicroSD memory card for which there is a slot in Ev3 Brick. However, Linux operating system works in such a way that the software may access external devices connected to CPU (like motors and sensors) by simply writing to a system file (to send a command) or reading from (to get a feedback). Lauro Ojeda has created an extensive tutorial at (29) on how to use this interface on the Ev3 brick while programming on C++ language. C++ is a general-purpose programming language which has many tools to control memory, create data structures and implement algorithms. C++ is a compiled language, meaning that a compiler program for a target platform is needed which will translate human-written instructions into commands understood by CPU. Codelite company offers different C++ cross-compilers, including that which compiles C++ programs on a computer running Windows for CPU with ARM architecture (like in Ev3 brick) running Linux. Plain text editor, in this case it was Geany 1.24 (personal preference of the author) ends the list of tools for programming Ev3 in C++.

A minor problem caused by Ev3 native software (firmware) is that it prevents launching any applications other than that of .rbf format produced with LEGO programming software. This issue has been solved by Ojeda who has made an .rbf application that is able to launch any prespecified application. The author has used (30) firmware documentation from LEGO and created an .rbf application called Runstl which provides interface for user to select an .STL input file and launch foam cutter application written in C++. The code is provided in Appendix 1.

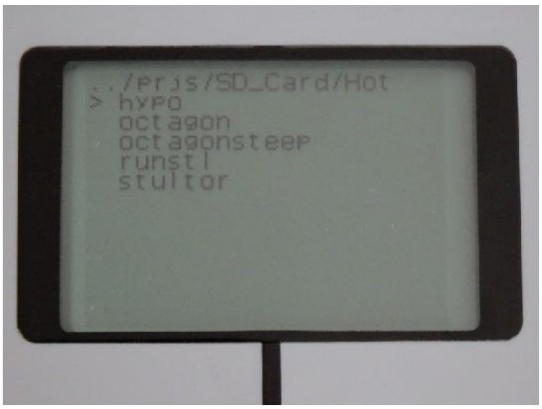


Figure 35 Runstl program screenshots

4.2 Cutter Software

The C++ code of the cutter software is shown in Appendix 2. The following sections outline how the software works.

4.2.1 Input and Preprocessing

The program receives a path to the input .STL file as an argument and reads it facet by facet. STL format has a disadvantage that it provides no information about the topology of the represented object, or in other words, which facets have common edges and form one surface. Therefore, this is what the software does as it reads the description of facet vertices: first, it finds matching vertices, second, it finds pairs of adjacent facets.

All facets have now information about their neighbours (adjacent facets). The program uses this information to classify into internal and boundary facets. Internal facets have all three neighbours, but facets lying on a boundary of surface lack one or more neighbours. By analyzing the relative positioning of boundary edge and the facet itself, boundary facets are classified into lower and upper ones. Here it is assumed that the model is oriented in x,y,z coordinate system so that z-axis corresponds to its vertical, but xy-plane (z=0) matches the horizontal surface of the table on which the foam material will slide. Z-axis positive direction points upward. A modern CAD software like Solidworks allows the user to create arbitrary coordinate systems and use them as a reference to export models into .STL format.

4.2.2 Algorithm to Match Contours

The program should now process all pairs of upper and lower contours. While there are unprocessed upper facets, one of them (the choice criteria is given below) will be taken and the corresponding closed upper contour will be assembled from the information about connections between facets.

Next, the program walks from facet to facet (they are connected) until it finds a lower facet. Similarly, the lower contour is assembled. Both contours are oriented in counter-clockwise direction for clarity. For each pair of upper and lower contours the dynamic programming algorithm (from section 3.4) is run. The algorithm outputs a sequence of double coordinates (upper x,y,z and lower x,y,z) that define points (they are called checkpoints inside the program) on the upper and lower contours to be swept by the hot wire. The sequence of checkpoints from the origin along all the contours and back again to the origin defines the whole tool path. At last, each checkpoint becomes a time value when it should be passed (time values are assigned, starting from 0, in order to main predefined average cutting speed for both upper and lower ends of the hot wire).

Every next closed loop that cuts one object out of sheet of material is connected to the already existing path where the distance between them is minimal. This is used as criteria for choosing a new unprocessed upper facet – that one which the closest to already defined checkpoints.

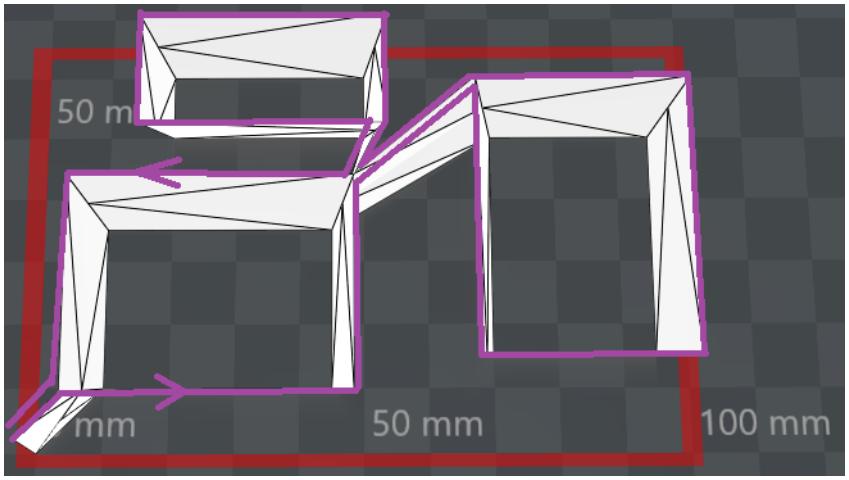


Figure 36 Example of tool path

4.2.3 Motion Control

In small NC applications stepper motors are typically used for their simplicity of position control. Stepper motor is controlled by pulses of current, each pulse forces the rotor to move to the next discrete position (typically several tens per revolution) which eliminates necessity in position sensor (called shaft encoder) (31). Mindstorms Ev3 set uses DC servomotors, with incremental encoders that have 1 degree accuracy for the output shaft position. Incremental encoder does not give an absolute position of the shaft in every moment, but sends a signal when the position is changed in forward or backward direction and requires a counter to transform these signals into position value withrespect to the starting position. Servomotors may have higher speeds and torques compared to stepper motors (16, pp. 12-13).

Control of servomotors can be done in open loop or closed loop. The open-loop control does not require any feedback from a servomotor. It uses well-described physical model of a system and controls the voltage or current fed to a servomotor depending on commanded position. Such controller is called a feedforward controller. Because of presence of unpredictable factors in virtually all real processes, the open-loop control cannot insure precise following of commanded position. Therefore, in CNC applications closed-loop, or feedback controllers are used together with feedforward controllers to accommodate changing circumstances of operation (16, pp. 157-185). Closed-loop controller makes decision about suitable voltage or current level based on the error between commanded position

The most widely used feedback controller is PID controller. The output of PID controller is a weighted sum of proportional, integral and derivative components of position error. Proportional component is directly proportional to the actual value of error with some coefficient K_p, called proportional gain. Proportional term serves for direct minimization of error. The integral term equals the integral (or sum) of error value over time. The respective coefficient K_i is called integral gain. The integral term is responsible for reduction of steady-state error, when small error does not produce enough power supply to overcome friction in a system when based on proportional term only. Big proportional and integral gains result

in fast response of a system, but may cause instability when servomotor overshoots the commanded position, tries to return back, overshoots again and continue to oscillate around the target position. In such systems a derivative term is added which is proportional to the rate of change of the error and starts to brake servomotor before it reaches the commanded position. The derivative gain is denoted as K_d.

In the foam cutter program, checkpoints coordinates are recalculated into the required encoder counts for each servomotor. The resolution of the motors were the following:

- Drive motion: 63,7 counts per mm
- Shift motion: 75 counts per mm
- Tilting motions (in both directions): 9000 counts per sine of inclination angle, or roughly 160 counts per one degree

Test runs with the first prototype of the foam cutter have showed that the simplest proportional controller causes a big lag in hot wire's position. Therefore, a feedforward control was added for the second model. Because of linear interpolation of position between successive checkpoints, the speed of each motion was easy to calculate. This value was used for feedforward controller that suggested the needed level of power (between 0 and 100) from the maximum and the required speed. The LEGO guide for Ev3 set states that L-motor runs at 160-170 rpm, while M-motor have 240-250 rpm speed. Additionally, for the closed-loop control, proportional gain for all motors was set. For the L-motors the choice of proportional gain was easy and straight forward: the gain was raised until the motors began to oscillate (this happened at $K_p = 5$) and then reduced to roughly half of that value ($K_p = 3$). The drive remained stable under all circumstances with the maximum error of 2-3 encoder counts (this information together with the power level was displayed on the screen of the Ev3 brick during the cutting process).

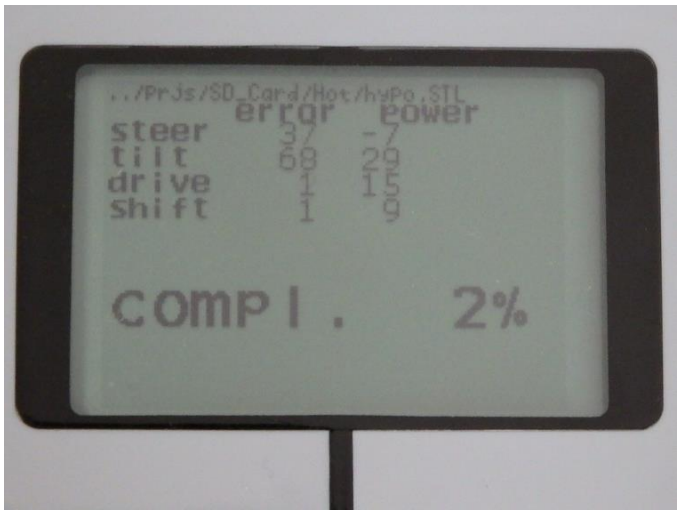


Figure 37 Cutting progress on the screen

The M-motors appeared to be much more difficult for control. Even with K_p = 0,1 they tended to oscillate. Because of significant friction in the worm gearboxes compared to their torque (8 Ncm versus 20 Ncm by L-motors) at the start and stop moments there were notable jerks (even with integral terms) that worsened the quality of the cut surface. Very high contoller loop execution frequency (more than 10000 cycles per second, with Linux timer accuracy of 1 microsecond) did not prevent from overshoots. To increase the stability, the derivative term was added. Its contribution suffered from the noise - at maximum speed, the M-motor must have 1500 encoder counts per second, which makes estimation of the derivative term from too frequently sampled data quite unprecise. This is a known problem in servomotor control and is resolved with decrease in sampling frequency and application of low-pass filter that removes high-frequency noise (32). In the cutter software the sampling frequency was reduced to about 1000 cycles per second and exponential smoothing was applied to the derivative term according to the following formula (33):

$$s_t = \alpha x_t + (1 - \alpha) s_{t-1} \quad (5)$$

Here s_t denotes smoothened value at time t, x_t is the actual value at time t, while the smoothing factor α was chosen to be equal 0,1 after several trials and errors. These measures reduced oscillations in operation of M-motors, but rapid changes in torque induced undesired oscillations of the bow because of its moment of inertia and limited stiffness of the drive. Finally, a solution that helped to

improve the situation was limiting changes of the power level to just one unit per cycle (which still allowed power to raise from 0 to 100 in 0,1 seconds).

5 Foam Cutting

5.1 Test Runs

Test runs of the second foam cutter model allowed to tune servomotors control and find the most suitable cutting speed and voltage to be applied to 0,25 mm thick spring steel wire (resistance at room temperature 0,6 Ω). Material for cutting was 20 mm thick XPS with density of 30 kg/m^3. While speeds up to 4-5 mm/s were possible, the cutting forces were distorting frame of the machine and required fast accelerations from the tilting servomotors. Therefore, the speed was set between 1,5 and 2,0 mm/s, usually 1,7 mm/s. The resulting kerf was 1-2 mm, depending on the actual speed at different points.

Simple geometrical shape was cut in order to test accuracy. The shapes were regular octogonal frustums with the upper width of 6 cm and lower 7 and 8 cm respectively. While produced shapes were not geometrically precise, some dimensions could deviate for more than 5 mm, the hot-wire after completing a closed contour always returned to the starting point with deviation less than 1 mm, which implies that the repeatability of the results is much better and improvement in accuracy can be achieved with a better mechanical model inside the cutter's software. Appendix 3 includes weblinks to videos that have captured the foam cutter in operation.

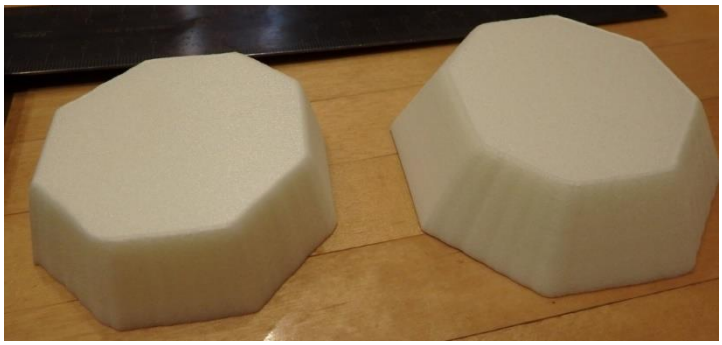


Figure 38 Octagonal frustums

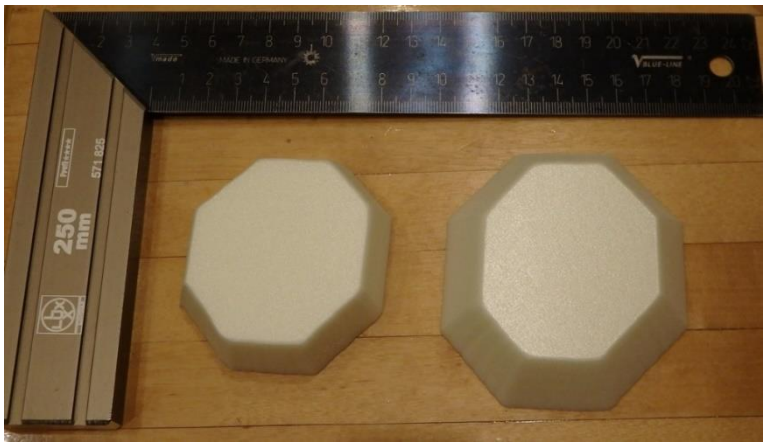


Figure 39 Octagonal frustums with ruler

The cutter's capability to cut curvelinear surfaces was tested together with correctness of generating tool path for multiple contours. Specifically, the test involved cutting one contour inside of another, so that the software had to choose place where to cut through the outer contour to reach the inner one and return back. While there was again a mismatch in the resulting dimensions, all supposed features were realized.

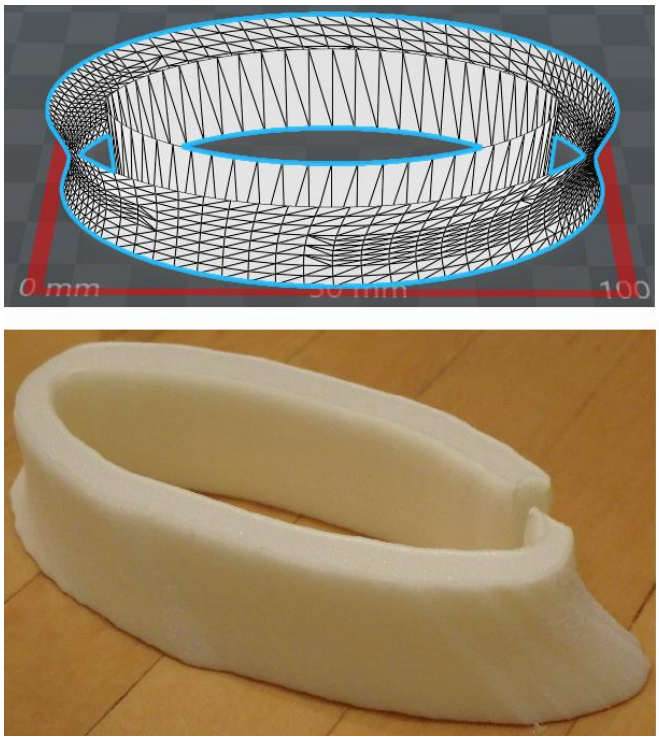


Figure 40 Two contours one inside another

5.2 Case Study: Airplane Fuselage

Despite the imperfections present in simple shapes, the machine was tested for primarily intended use – production of airplane foam body. For reproduction in XPS a model of Yak-130 jet was chosen. Precise Solidworks CAD model of its complicated exterior was available in the GrabCAD library at (34). The model was downscaled in ratio of 1:20. The fuselage was sliced into 28 layers of 20 mm thickness. Additionally a large segment of its vertical stabilizer (fin) was prepared for a single cut (dimensions of the cutter did not allow to cut all the height of it but only 10 cm out of 14 cm). 28 pieces for the fuselage were cut in sets of 7 pieces. STL representation of these sets contained up to 4000 facets and it took Ev3 brick between 12 to 15 seconds to process the models and generate tool paths. One set required 15-20 minutes to cut.

When all 28 layers were hot-glued together, mismatches between adjacent layers were easy to notice, especially on various protruding thin details of the exterior. However, the foam model showed resemblance to the prototype, including the 10 cm tall fin cut in one piece. Appendix 3 refers to media files capturing stages of work.

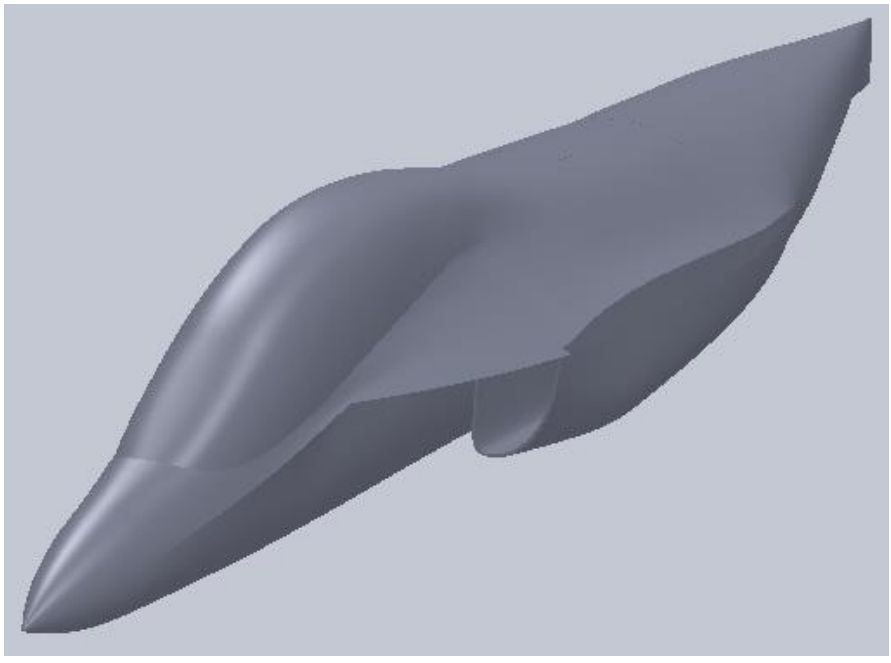


Figure 41 Yak-130 fuselage Solidworks model

Figure 42 Yak-130 XPS model: top view (above), bottom view (below)

The surface structure shows unnecessary change in direction of the hot wire which indicates flaws of ruled surface approximation algorithm. It can be eliminated by introducing a secondary optimization criterion into dynamic programming that will preserve the directionof the tool unless a change is really needed.

During operation one feature of the construction was revealed that needs special attention. When some pieces in a set are fully cut they may fall down before the motion is ended. If the fall was not complete, it may create an obstacle for the drive motion of the machine because of interference with the table. Absence of any offsetting mechanism in trajectory planning clearly decreases the accuracy of cut. Partly it maybe compensated by using a surface offsetting tool in CAD software, but for better results this should consider a true physical model of the cutting process.

6 Summary and Discussion

The thesis dealt with designing a numerically controlled hot-wire foam cutter dedicated for do-it-yourself hobby applications, like modelling of flying aircraft. The challenge involved multiple disciplines, starting with mechanical engineering, continueing with mechatronics and control engineering, ending with programming and information technology.

The mechanical design showed limitations of conventional LEGO parts, but allowed to create a new kinematical layout of a 4-axis hot-wire foam cutter that allows a wide range of the wire's positions while reducing the load on servodrives. LEGO showed its full educational power in engineering and technology, because it poses engineering challenges that are common in the real industry (like backlash or excessive flexibility).

In the programming part of the design it was showed that a quite small program with a little more than 1000 lines of code when run on a 300 MHz processing unit may successfully use results of CAD software, which requires sometimes 10 gigabytes of disk space on a personal computer (like Solidworks does). While there are CAM programs that have hundreds of settings, a self-written program gives an extensive knowledge of how the machine works. In the author's opinion, this makes a process of interaction between a user and a program more clear and unambiguous. Programming of the developed foam cutter required to create a new algorithm for processing of 3D shapes that would match the set objectives (use information about the shape from 3D modelling software in a common 3D format). While the algorithm could be made more stable regarding avoidance of

unnecessary changes in the position of the cutting tool, it proved its applicability to the stated problem.

Both the hardware and software part of this project can be improved. LEGO can be substituted with other components that perform better and do not wear out as fast as plastic pieces. The tool path planning software may become more customizable (concerning cutting parameters or input formats) and give the user more feedback about the progress of cutting.

The author hopes that this work may serve as a guide for motivated hobbyists to build their own NC systems. It will inform them about difficulties and methods to overcome them.

Figures

References

1. Yak-130 model by FMS company. http://www.horizonhobby.com/product/airplanes/airplanes-14501--1/plug-n-play/yak-130-jet-pnp--70mm:-red-p-fmm088pred . Accessed on 7 May 2017.
2. Degtyarev, A. Concorde from Styrofoam. http://rc-aviation.ru/polucopy/1938-concorde . Accessed on 7 May 2017.
3. FliteTest JA-37 Viggen. http://www.flitetest.com/articles/ja-37-viggen-scratch-build . Accessed on 7 May 2017.
4. Simple but Mighty DIY 4 Axis CNC Foam Cutter. http://www.rcgroups.com/forums/showthread.php?2563600-Simple-but-mighty-DIY-4-axis-CNC-Foam-Cutter-Theremino-Controller-(Not-Arduino)! . Accessed on 7 May 2017.
5. Seybold, H. CNC Hot-Wire Cutter. http://www.cnc-hotwire.de . Accessed on 7 May 2017.
6. Aerotetris company. Official website. http://bukvorez.com . Accessed on 7 May 2017.
7. Wikipedia. Hot-Wire Foam Cutter. http://en.wikipedia.org/wiki/Hot-wire_foam_cutter . Accessed on 7 May 2017.
8. Broek, J.J., Horvath, I., de Smit, B., Lennings, A.F., Rusak, Z. Vergeest, J.S.M. 2002. Free-Form Thick Layer Object Manufacturing Technology for Large-Sized Physical Models. Elsevier, Automation in Construction 11, 335-347.
9. Aitchison, D., Brooks, H., Bain., J., Pons, D. 2009. An Investigation into the Prediction of Optimal Machining Conditions for Polystyrene Foam Cut with a Taut Hot-Wire. The Annals of “Dunarea de Jos” University of Galati, Fascicle V, Technologies in Machine Building, ISSN 1221- 4566.
10. Abeysinghe, A., Abeysiriwardena, S., Nanayakkarawasam, R., Wimalsiri, W., Lalitharatne, T.D., Tennakoon, S. 2016. Development of a Numerically Controlled Hot Wire Foam Cutting Machine for Wing Mould Construction.Moratuwa Engineering Research Conference (MERCon). DOI: 10.1109/MERCon.2016.7480116. Accessed on 25 January 2017.

11. Arthur Sacek LEGO 360° Milling Machine. 2012. http://www.youtube.com/watch?annotation_id=annotation_686662&feature=iv&src_vid=pX1cO2XhMrg&v=oF0pMILT7_Y . Accessed on 7 May 2017.
12. ABB Robotics. Engineering with Lego can transform the future. 2014. http://www.youtube.com/watch?v=-tsPuHaHFDw&t=7s . Accessed on 7 May 2017.
13. Lego® Mindstorms NXT Styrofoam Cutter. 2009. http://www.youtube.com/watch?v=tzNww9_m5Uk . Accessed on 7 May 2017.
14. The Engineering Toolbox. Young's Modulus – Tensile Modulus or Modulus of Elasticity – for steel, glass, wood and other common materials. http://www.engineeringtoolbox.com/young-modulus-d_417.html . Accessed on 7 May 2017.
15. Wikipedia. Backlash (engineering). http://en.wikipedia.org/wiki/Backlash_(engineering) . Accessed on 7 May 2017.
16. Suh, S.-H., Kang, S.-K., Chung, D.-H., Stroud, I., 2008, Theory and Design of CNC Systems. London: Springer Series in Advanced Manufacturing ISSN 1860-5168
17. Wikipedia. Ruled surface. http://en.wikipedia.org/wiki/Ruled_surface . Accessed on 7 May 2017.
18. Quan, L., Yongzhang, W., Hongya, F., Zhenyu, H., 2008. Cutting Path Planning for Ruled Surface Impellers. Chinese Journal of Aeronautics 21(2008) 462-471.
19. Flöry, S., Pottmann, H., 2010. Ruled Surfaces for Rationalization and Design in Architecture. Association for Computer Aided Design in Architecture (ACADIA). Life Information. On Responsive Information and Variations in Architecture: pp. 103-109.
20. Yao, Z., de la Cruz, M.O., 2014. Ruled Surface Underlying KcsA Potassium Channels. Soft Matter, Issue 4.
21. Koc, B., Lee, Y.-S., 2002. Adaptive ruled layers approximation of STL models and multiaxis machining applications for rapid prototyping. Journal of Manufacturing Systems. DOI: 10.1016/S0278-6125(02)80159-5. Accessed on 7 May 2017.

22. Wang, C.C.L., Tang, K., 2005. Optimal Boundary Triangulations of an Interpolating Ruled Surface. Journal of Computing and Information Science in Engineering. DOI: 10.1115/1.2052850. Accessed on 7 May 2017.
23. Fuchs, H., Kedem, Z.M., Uselton, S.P., 1977. Optimal Surface Reconstrcution from Planar Contours. Association for Computing Machinery, Inc.
24. Wikipedia. Dynamic Programming. http://en.wikipedia.org/wiki/Dynamic_programming . Accessed on 7 May 2017.
25. Wang., C.C.L., Elber, G., 2013. Multi-Dimensional Dynamic Programming in Ruled Surface Fitting. Preprint submitted to Elsevier.
26. Bi, Q., Lu, Y., Li, Z., Zhu, L., Ding, H., Liu, G., 2013. Constrained Ruled Surface Reconstruction for 5-axis NC Machining of Aero Structure. DOI: 10.4028/www.scientific.net/AMM.365-366.938. Accessed on 7 May 2017.
27. Gupta, T., Chandila, P.K., Tripathi, V., Choudry, A.R., 2004. Direct Slicing with Optimum Number of Contour Points. International Journal of CAD/CAM, Vol 4, No 1.
28. Nine alternative programming languages for LEGO MINDSTORMS. http://www.legoengineering.com/alternative-programming-languages/ . Accessed on 7 May 2017.
29. Ojeda, L. Robot Navigation. www.robotnav.com . Accessed on 7 May 2017.
30. LEGO Mindstorms Ev3 Firmware Documentation. http://ev3.fantastic.computer/doxygen/index.html . Accessed on 7 May 2017.
31. Lifewire. Stepper Motors vs. Servo Motors - Selecting a Motor. http://www.lifewire.com/stepper-motor-vs-servo-motors-selecting-a-motor-818841 . Accessed on 7 May 2017.
32. Wikipedia. PID controller. http://en.wikipedia.org/wiki/PID_controller . Accessed on 7 May 2017.
33. Wikipedia. Exponential Smoothing. http://en.wikipedia.org/wiki/Exponential_smoothing . Accessed on 7 May 2017.
34. GrabCAD. Yak-130 Solidworks model. http://grabcad.com/library/yak-130-1 . Accessed on 7 May 2017.

Appendix 1. Runstl Program

```
vmthread MAIN
{
	UI_DRAW(SELECT_FONT,SMALL_FONT)

	DATA32 timer
	DATA32 status

	DATAS rootdir 100
	DATAS curdir 100
	DATAS subdir 20
	DATAS exec 150
	DATA8 nfolders
	DATA8 pressed
	DATA8 line
	DATA16 lnheight
	MOVE16_16(10,lnheight)
	DATA16 indent
	MOVE16_16(15,indent)
	DATA16 height
	DATA16 len

	FILENAME(GET_FOLDERNAME,100,rootdir)
	FILENAME(GET_FOLDERNAME,100,curdir)

UPD_DIRS:
	UI_DRAW(FILLWINDOW,BG_COLOR,0,0)
	UI_DRAW(TEXT,FG_COLOR,0,0,curdir)

	FILE(GET_FOLDERS,curdir,nfolders)
	MOVE8_8(0,line)
	MOVE16_16(0,height)
	JR_FALSE(nfolders, CONTROLS)

PRINT_DIRS:
	ADD16(height,lnheight,height)
	ADD8(line,1,line)
	FILE(GET_SUBFOLDER_NAME,curdir,line,20,subdir)
	UI_DRAW(TEXT,FG_COLOR,indent,height,subdir)
	JR_LT8(line,nfolders,PRINT_DIRS)

	MOVE8_8(1,line)
	MOVE16_16(lnheight, height)
	UI_DRAW(TEXT,FG_COLOR,0,height,'>')

CONTROLS:
	UI_DRAW(UPDATE)
	UI_BUTTON(WAIT_FOR_PRESS)
	TIMER_WAIT(100,timer)
	TIMER_READY(timer)

	UI_BUTTON(SHORTPRESS,UP_BUTTON,pressed)
```

```
        JR_TRUE(pressed,PREV_DIR)

        UI_BUTTON(SHORTPRESS,DOWN_BUTTON,pressed)
        JR_TRUE(pressed,NEXT_DIR)

        UI_BUTTON(SHORTPRESS,LEFT_BUTTON,pressed)
        JR_TRUE(pressed,PARENT_DIR)

        UI_BUTTON(SHORTPRESS,RIGHT_BUTTON,pressed)
        JR_TRUE(pressed,ENTER_DIR)

        UI_BUTTON(SHORTPRESS,ENTER_BUTTON,pressed)
        JR_TRUE(pressed,EXEC_DIR)

        UI_BUTTON(SHORTPRESS,BACK_BUTTON,pressed)
        JR_TRUE(pressed,EXIT)

PREV_DIR:
        JR_EQ8(line,0,CONTROLS)
        JR_EQ8(line,1,CONTROLS)
        UI_DRAW(TEXT,FG_COLOR,0,height,' ')
        SUB8(line,1,line)
        SUB16(height,lnheight,height)
        UI_DRAW(TEXT,FG_COLOR,0,height,'>')
        JR(CONTROLS)

NEXT_DIR:
        JR_EQ8(line,nfolders,CONTROLS)
        UI_DRAW(TEXT,FG_COLOR,0,height,' ')
        ADD8(line,1,line)
        ADD16(height,lnheight,height)
        UI_DRAW(TEXT,FG_COLOR,0,height,'>')
        JR(CONTROLS)

PARENT_DIR:
        FILENAME(SPLIT,curdir,100,curdir,subdir,subdir)
        STRINGS(GET_SIZE,curdir,len)
        SUB16(len,1,len)
        WRITE8(0,len,curdir)
        JR(UPD_DIRS)

ENTER_DIR:
        JR_FALSE(line,CONTROLS)
        FILE(GET_SUBFOLDER_NAME,curdir,line,20,subdir)
        STRINGS(ADD,curdir,'/',curdir)
        STRINGS(ADD,curdir,subdir,curdir)
        JR(UPD_DIRS)

EXEC_DIR:
        FILE(GET_SUBFOLDER_NAME,curdir,line,20,subdir)
        STRINGS(ADD,rootdir,'/stultor ',exec)
```

```
        STRINGS(ADD,exec,curdir,exec)
        STRINGS(ADD,exec,'/',exec)
        STRINGS(ADD,exec,subdir,exec)
        STRINGS(ADD,exec,'.STL',exec)
        SYSTEM(exec,status)
        UI_DRAW(TEXT,FG_COLOR,0,0,'@')
        UI_DRAW(UPDATE)
        UI_DRAW(TEXT,FG_COLOR,0,0,curdir)
        JR(CONTROLS)

EXIT:
}
```

Appendix 2. C++ Foam Cutter Program Stultor

```
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <fcntl.h>
#include <set>
#include <string.h>
#include <sys/mman.h>
#include <sys/time.h>
#include <unistd.h>
#include "lms2012.h"
#include "d_lcd.h"

const float LARGE = 1e9f;
const float ZERO = 1e-9f;
const float MAX_INCLINATION = 40.0f; //degrees
const float PRECISION = 1e-4f;  //mm precision to distinguish vertices

//constants for EV3 hardware
const char MOTOR_PORT_A = 0;
const char MOTOR_PORT_B = 1;
const char MOTOR_PORT_C = 2;
const char MOTOR_PORT_D = 3;

const float PI = 3.1415926f;
/*machine geometry & material properties*/
const float steer_steps_per_sin = (40.0f*360.0f) / (0.5f*3.2f);  //resolution of the steering M-motor
const float tilt_steps_per_sin = (40.0f*360.0f) / (0.5f*3.2f); //resolution of the tilting M-motor
const float shift_steps_per_mm = (360.0f) / (1.5f*3.2f);      //resolution of the shifter L-motor
const float drive_steps_per_mm = (24.0f*360.0f) / (PI*43.2f);  //resolution of the drive L-motor
const float joint_height_mm = 6.0f;
const float M_rev_per_sec = 170.0f/60.0f; //nominal speed of the M-motor
const float L_rev_per_sec = 110.0f/60.0f; //nominal speed of the L-motor
const float CUTTING_SPEED = 1.7f; // mm/s
const int CYCLE_SLEEP = 500; //microseconds
const int DISPLAY_SLEEP = 500000; //microseconds

using namespace std;

struct vec3_t {
    float x, y, z;

    vec3_t()
    {
    }

    vec3_t(float* point):
        x(point[0]), y(point[1]), z(point[2])
    {
    }

    vec3_t(float x0, float y0, float z0):
        x(x0), y(y0), z(z0)
    {
    }

    friend bool operator<(const vec3_t& lhs, const vec3_t& rhs) //"comparator"
    {
        return ((lhs.x+PRECISION < rhs.x) ||
                (!(lhs.x-PRECISION > rhs.x) && (lhs.y+PRECISION < rhs.y)) ||
                (!(lhs.x-PRECISION > rhs.x) && !(lhs.y-PRECISION > rhs.y) && (lhs.z+PRECISION < rhs.z)));
    }

    friend float operator*(const vec3_t& u, const vec3_t& v) //dot product
    {
        return (u.x*v.x + u.y*v.y + u.z*v.z);
    }
```

```
67.
68.     friend vec3_t operator%(const vec3_t& u, const vec3_t& v) //cross product
69.     {
70.         return vec3_t(u.y*v.z - u.z*v.y, u.z*v.x-u.x*v.z, u.x*v.y - u.y*v.x);
71.     }
72.
73.     friend vec3_t operator*(const float& n, const vec3_t& v)  //multiplucation with scalar
74.     {
75.         return vec3_t(n*v.x, n*v.y, n*v.z);
76.     }
77.
78.     friend vec3_t operator+(const vec3_t& u, const vec3_t& v)
79.     {
80.         return vec3_t(u.x+v.x, u.y+v.y, u.z+v.z);
81.     }
82.
83.     friend vec3_t operator-(const vec3_t& u, const vec3_t& v)
84.     {
85.         return vec3_t(u.x-v.x, u.y-v.y, u.z-v.z);
86.     }
87.
88.     friend float operator^(const vec3_t& u, const vec3_t& v)
89.     {
90.         return (u.x*v.y - u.y*v.x);
91.     }
92.
93.     friend float magnitude(const vec3_t& v)
94.     {
95.         return sqrtf(v*v);
96.     }
97.
98.     friend vec3_t interpolate(const vec3_t& A, const vec3_t& B, float q)
99.     {
100.             return A+q*(B-A);
101.         }
102.
103.         friend float square(const vec3_t& v)
104.         {
105.             return (v*v);
106.         }
107.     };
108.     typedef vec3_t * pvertex_t;
109.     bool pvertex_comp(pvertex_t l, pvertex_t r) {return *l<*r;}
110.     const vec3_t ZAXIS = vec3_t(0.0f, 0.0f, 1.0f);
111.     const vec3_t ORIGIN = vec3_t(0.0f, 0.0f, 0.0f);
112.
113.
114.     enum facet_type { upper_facet, internal_facet, lower_facet, used_facet };
115.     typedef struct facet_t facet_t;
116.     typedef facet_t * pfacet_t;
117.     struct facet_t
118.     {
119.         vec3_t n;
120.         pvertex_t v[3];
121.         pfacet_t a[3]; //adjacent facets
122.         float dist;
123.         pfacet_t bktrace;
124.         int id;
125.         facet_type type;
126.     };
127.
128.
129.     struct edge_t
130.     {
131.         pair<pvertex_t, pvertex_t> v;
132.         pfacet_t f;
133.         pfacet_t * edgeid;
134.
135.         edge_t(pvertex_t v1, pvertex_t v2, pfacet_t f0, pfacet_t * edgeid0):
136.             f(f0), edgeid(edgeid0)
```

```
137.            {
138.                if(*v1<*v2) v = make_pair(v1, v2);
139.                else v = make_pair(v2, v1); //edges are represented in oriented manner
140.            }
141.
142.            friend bool operator<(const edge_t& lhs, const edge_t& rhs) //"comparator"
143.            {
144.                return (lhs.v.first < rhs.v.first) || (!(rhs.v.first < lhs.v.first) && (lhs.v.secon
    d < rhs.v.second));
145.            }
146.        };
147.        typedef set<edge_t> edges_t;
148.
149.
150.        struct boundarel_t
151.        {
152.            pvertex_t v;
153.            pfacet_t f;
154.        };
155.        typedef pair<boundarel_t, boundarel_t> bounode_t;  //"boundary node"
156.        typedef bounode_t * pbounode_t;
157.
158.        struct patch_t
159.        {
160.            pfacet_t upp, low;
161.        };
162.        typedef patch_t * ppatch_t;
163.
164.        struct steep_enough_t
165.        {
166.            float limcos;
167.            vec3_t v;
168.
169.            steep_enough_t(float inclination_limit):
170.                limcos(cosf(3.1415926f/180.0f*inclination_limit))
171.            {
172.            }
173.
174.            bool operator()(vec3_t& A, vec3_t& B, vec3_t& C, vec3_t& D) //AB on the one boundary, C
    D on the other one
175.            {
176.                v = A+B-C-D;
177.                return (v*ZAXIS > magnitude(v)*limcos) || (v*ZAXIS < -magnitude(v)*limcos);
178.            }
179.        };
180.
181.        enum direction_t {west, north, norwest, indef, start};
182.        typedef struct routel_t routel_t;
183.        typedef routel_t * proutel_t;
184.        struct routel_t
185.        {
186.            float selfcost, routecost;
187.            direction_t from;
188.
189.            void improve(proutel_t r, direction_t dir)
190.            {
191.                if(r->routecost < routecost)
192.                {
193.                    routecost = r->routecost;
194.                    from = dir;
195.                }
196.            }
197.        };
198.
199.        struct get_cost_t
200.        {
201.            pfacet_t f, next;
202.            vec3_t H, n;
203.            vec3_t CM, nextCM;
204.            vec3_t p, nextp;
```

Appendix 2. C++ Foam Cutter Program Stultor

```
            float dev, nextdev;

            get_cost_t()
            {
                f = NULL;
            }

            void setseed(pfacet_t start_facet)
            {
                f = start_facet;
                CM = (1.0f/3.0f) * (*(f->v[0]) + *(f->v[1]) + *(f->v[2]));
            }

            float operator()(vec3_t& A, vec3_t& B, vec3_t& C, vec3_t& D) //AB on the one boundary, CD on the other one
            {
                if(f==NULL)
                {
                    printf("uninitialized facet\n");
                    return -1.0f;
                }

                H = 0.25*(A+B+C+D);      //centre of the patch
                n = (B-A+D-C)%(C-A+D-B); //averaged normal to the patch
                if(magnitude(n) < ZERO) return LARGE; //something is wrong, not a good patch anyway

                n = (1.0f/magnitude(n)) * n; //unit normal
                p = CM-H;
                if(magnitude(p)<ZERO) return ZERO;
                dev = magnitude(p - (p*n)*n);  //deviation

                for(;;)
                {
                    next = NULL;
                    for(int i=0; i<3; i++)
                    {
                        if(f->a[i] != NULL)
                        {
                            nextCM = (1.0f/3.0f) * (*(f->a[i]->v[0]) + *(f->a[i]->v[1]) + *(f->a[i]->v[2]));
                            nextp = nextCM-H;
                            if(magnitude(nextp)<ZERO) return ZERO;
                            nextdev = magnitude(nextp - (nextp*n)*n);

                            if(nextdev < dev)
                            {
                                next = f->a[i];
                                CM = nextCM;
                                p = nextp;
                                dev = nextdev;
                            }
                        }
                    }

                    if(next!=NULL)
                        f = next;
                    else
                        return (p*n)*(p*n);
                }
                return LARGE;
            }
        };


        enum mech_dof {steer, tilt, drive, shift, MAX_MECH_DOF}; //degrees of freedom of the machine - actuators
        typedef struct checkpoint_t checkpoint_t;
        typedef checkpoint_t * pcheckpoint_t;
        struct checkpoint_t
        {
```

Appendix 2. C++ Foam Cutter Program Stultor

```
            vec3_t upp, low;
            float axis[MAX_MECH_DOF];
            int ct; //check time
            pcheckpoint_t next;

            checkpoint_t()
            {
            }

            checkpoint_t(const vec3_t& upper, const vec3_t& lower, pcheckpoint_t nextcp):
                upp(upper), low(lower), next(nextcp)
            {
                vec3_t d = upper - lower;
                float len = magnitude(d);
                axis[steer] = -steer_steps_per_sin * (d.y/len);
                axis[tilt]  = -tilt_steps_per_sin * (d.x/sqrtf(d.x*d.x+d.z*d.z));
                d = lower + ((joint_height_mm - lower.z)/d.z) * d;
                axis[drive] = drive_steps_per_mm * d.x;
                axis[shift] = shift_steps_per_mm * d.y;
            }

            float& operator[](int id) { return axis[id]; }

            const float& operator[](int id) const { return axis[id]; }

            friend checkpoint_t operator-(const checkpoint_t& u, const checkpoint_t& v)
            {
                checkpoint_t res;
                for(int i=0; i<MAX_MECH_DOF; i++)
                    res[i] = u[i] - v[i];
                return res;
            }
        };

        struct state_t
        {
            float x;
            float xdot;
        };

        struct coordinates_t
        {
            state_t state[MAX_MECH_DOF];
            pcheckpoint_t p;
            int pn;

            struct timeval tv_begin;
            struct timeval tv_cur;
            int t, tlast;
            pcheckpoint_t cur, old;

            coordinates_t(pcheckpoint_t p0, int pn0, float cut_speed): //cut speed in mm/s
                p(p0), pn(pn0)
            {
                char s[100];

                float dist2time = 1e6f*0.5f/cut_speed;
                p[0].ct = 0;
                old = p;
                cur = old->next;
                for(int i=1; i<pn; i++)
                {
                    cur->ct = old->ct + lrintf(dist2time * (magnitude(cur->upp - old-
    >upp) + magnitude(cur->low - old->low)));
                    old = cur;
                    cur = cur->next;
                }
                tlast = old->ct;
            }

```

```
340.            void runchrono()
341.            {
342.                gettimeofday(&tv_begin, NULL);
343.                cur = p;
344.            }
345.
346.            bool finished()
347.            {
348.                gettimeofday(&tv_cur, NULL);
349.                t = (tv_cur.tv_sec-tv_begin.tv_sec)*1000000 + (tv_cur.tv_usec-tv_begin.tv_usec);
350.                return (t > tlast);
351.            }
352.
353.            void update(int i)
354.            {
355.                if(finished()) return;
356.
357.                if(cur->ct < t)
358.                {
359.                    old = cur;
360.                    while(cur->ct < t) cur = cur->next;
361.                }
362.
363.                float dx = cur->axis[i] - old->axis[i];
364.                float dt = cur->ct - old->ct;
365.                state[i].xdot = dx/dt;  // in steps/microsec
366.                state[i].x = old->axis[i] + state[i].xdot * (t - old->ct);
367.            }
368.
369.            ~coordinates_t()
370.            {
371.                free(p);
372.            }
373.        };
374.
375.        struct motor_t
376.        {
377.            static MOTORDATA *pMotorData;
378.            static char motor_command[3];
379.            static int motor_file;
380.            static int encoder_file;
381.
382.            char port;
383.            float pos_coef, der_coef, speed_coef;
384.            int beginpos;
385.            float pos, err, prev, buf, predict;
386.            state_t *target;
387.            int axis;
388.            int power;
389.            char command[3];
390.
391.            motor_t(char port0, float pos_coef0, float der_coef0,
392.                    float speed_coef0, state_t *target0):
393.                port(port0), pos_coef(pos_coef0), der_coef(der_coef0),
394.                speed_coef(speed_coef0), target(target0), err(0.0f), prev(0.0f), buf(0.0f), power(0
    )
395.            {
396.                command[0] = opOUTPUT_POWER;
397.                command[1] = 1 << port;
398.                command[2] = 0;
399.                write(motor_file, command, 3);
400.
401.                command[0] = opOUTPUT_START;
402.                write(motor_file, command, 2);
403.
404.                command[0] = opOUTPUT_RESET;
405.                write(motor_file, command, 2);
406.
407.                command[0] = opOUTPUT_POWER;
408.
```

```
409.                beginpos = pMotorData[port].TachoSensor;
410.            }
411.
412.            void act()
413.            {
414.                pos = pMotorData[port].TachoSensor - beginpos;
415.                pos = target->x-pos;
416.                buf = 0.1*(pos-prev)+0.9f*buf;
417.                predict = target->xdot*speed_coef + pos*pos_coef + buf*der_coef;
418.                err += pos;
419.                if((predict > power) && (power < 100)) power++;
420.                else if((predict < power) && (power > -100)) power--;
421.
422.                command[1] = 1 << port;
423.                command[2] = power;
424.                write(motor_file, command, 3);
425.                prev = pos;
426.            }
427.
428.            static bool init()
429.            {
430.                //Open the device file asscoiated to the motor controlers
431.                if((motor_t::motor_file = open(PWM_DEVICE_NAME, O_WRONLY)) == -1) return false;
432.                //Open the device file asscoiated to the motor encoders
433.                if((motor_t::encoder_file = open(MOTOR_DEVICE_NAME, O_RDWR | O_SYNC)) == -
    1) return false;
434.                motor_t::pMotorData = (MOTORDATA*)mmap(0, sizeof(MOTORDATA)*vmOUTPUTS, PROT_READ |
    PROT_WRITE, MAP_FILE | MAP_SHARED, motor_t::encoder_file, 0);
435.                if (motor_t::pMotorData == MAP_FAILED) return false;
436.                return true;
437.            }
438.
439.            static void deinit()
440.            {
441.                close(motor_t::motor_file);
442.                close(motor_t::encoder_file);
443.            }
444.
445.            ~motor_t()
446.            {
447.                command[1] = 1 << port;
448.                command[2] = 0;
449.                while(pMotorData[port].Speed)
450.                {
451.                    write(motor_file, command, 3);
452.                }
453.
454.                command[0] = opOUTPUT_STOP;
455.                write(motor_file, command, 2);
456.            }
457.        };
458.
459.        MOTORDATA* motor_t::pMotorData;
460.        char motor_t::motor_command[3];
461.        int motor_t::motor_file;
462.        int motor_t::encoder_file;
463.
464.        struct buttons_t
465.        {
466.            int button_file;
467.            UI *pbuttons;
468.            enum {UP, CENTER, DOWN, RIGHT, LEFT, BACK};
469.
470.            buttons_t()
471.            {
472.                if((button_file = open(UI_DEVICE_NAME, O_RDWR | O_SYNC)) == -1) return;
473.                pbuttons  =  (UI*)mmap(0, sizeof(UI), PROT_READ | PROT_WRITE, MAP_FILE | MAP_SHARED
    , button_file, 0);
474.                if (pbuttons == MAP_FAILED) return;
475.            }
```

Appendix 2. C++ Foam Cutter Program Stultor

```
476.
477.            bool keypressed()
478.            {
479.                for(int key = 0; key < BUTTONS; key++)
480.                {
481.                    if(pbuttons->Pressed[key]) return true;
482.                }
483.
484.                return false;
485.            }
486.
487.            ~buttons_t()
488.            {
489.                close(button_file);
490.            }
491.        };
492.
493.        pfacet_t find_other_side(pfacet_t facets, pfacet_t seed, int maxn, facet_type target, steep
    _enough_t *steep)
494.        {
495.            pfacet_t q[maxn];
496.            q[0] = seed;
497.            pvertex_t A, B, C, D;
498.            for(int i=0; i<3; i++)
499.            {
500.                if(seed->a[i] == NULL)
501.                {
502.                    A = seed->v[(i+1)%3];
503.                    B = seed->v[(i+2)%3];
504.                    break;
505.                }
506.            }
507.
508.            int left=0, right=1;
509.            bool used[maxn];
510.            for(int i=0; i<maxn; i++) used[i] = false;
511.
512.            while(left<right)
513.            {
514.                if(q[left]->type == target)
515.                {
516.                    for(int i=0; i<3; i++)
517.                    {
518.                        if(q[left]->a[i] == NULL)
519.                        {
520.                            C = q[left]->v[(i+1)%3];
521.                            D = q[left]->v[(i+2)%3];
522.                            break;
523.                        }
524.                    }
525.                    if((*steep)(*A, *B, *C, *D))
526.                        return q[left];
527.                }
528.
529.                for(int i=0; i<3; i++)
530.                {
531.                    if(q[left]->a[i] == NULL) continue;
532.                    if(!used[q[left]->a[i] - facets])
533.                    {
534.                        q[right++] = q[left]->a[i];
535.                        used[q[left]->a[i]-facets] = true;
536.                    }
537.                }
538.                left++;
539.            }
540.
541.            return NULL; //normally should never happen, just in case if...
542.        }
543.
```

Appendix 2. C++ Foam Cutter Program Stultor

```
544.        bool input(const char *fname, pfacet_t& solid, int& nfacet, pvertex_t& vertices, int& nvert
    ex)
545.        {
546.            FILE *partfile;
547.            partfile = fopen(fname, "rb");
548.            if(!partfile)
549.            {
550.                printf("unable to open %s\n", fname);
551.                return false;
552.            }
553.
554.            char buf[80];
555.            fread(buf, 80, 1, partfile); //80 chars for the name of a solid in .STL
556.
557.            fread(&nfacet, sizeof(nfacet), 1, partfile); //4-byte integer for number of facets
558.
559.            solid = (pfacet_t)malloc(sizeof(facet_t)*nfacet);
560.            vertices = (pvertex_t)malloc(sizeof(vec3_t)*(nfacet+1));
561.            nvertex = 0;
562.            set<pvertex_t, bool(*)(pvertex_t, pvertex_t)> vdb(&pvertex_comp); //vertices data base
    
563.            edges_t edges;
564.            float point[3];
565.
566.            for(int i=0; i<nfacet; i++)
567.            {
568.                fread(point, sizeof(point), 1, partfile); //read facet normal
569.                solid[i].n = vec3_t(point);
570.
571.                for(int j=0; j<3; j++)
572.                {
573.                    fread(point, sizeof(point), 1, partfile); //read facet vertices
574.
575.                    vertices[nvertex] = vec3_t(point);
576.                    solid[i].v[j] = (pvertex_t) *(vdb.insert(vertices + nvertex).first);
577.                    if(solid[i].v[j] == vertices+nvertex) nvertex++;
578.                    if(nvertex>nfacet)
579.                    {
580.                        printf("weak surface %d %d\n", nvertex, nfacet);
581.                        return false;
582.                    }
583.                }
584.
585.                fread(buf, 2, 1, partfile); //read two useless colour bytes
586.
587.                //let's find facet connectivity (neighbourhood)
588.                //we put all the edges of the current facet in a set
589.                //if such edge is already there, we make corresponding facets to "know" about each
    other
590.                for(int j=0; j<3; j++)
591.                {
592.                    solid[i].a[j] = NULL;
593.                    pair<edges_t::iterator, bool> test;
594.                    //triangle ABC has edges BC, CA, AB
595.                    test = edges.insert( edge_t(solid[i].v[(j+1)%3], solid[i].v[(j+2)%3], solid+i,
    solid[i].a+j) );
596.
597.                    if(!test.second) //such edge has already been found
598.                    {
599.                        solid[i].a[j] = test.first-
    >f; //the corresponding facet and the current facet are neighbours
600.                        *(test.first->edgeid) = solid+i;
601.                    }
602.                }
603.            }
604.            vdb.clear();
605.            edges.clear();
606.
607.            return true;
608.        }
```

```
609.
610.        bool lower_boundary(vec3_t& B1, vec3_t& B2, vec3_t& I)
611.        {
612.            if((B1.z < I.z) && (B2.z < I.z))  //obviously it is lower boundary
613.                return true;
614.
615.            vec3_t b = B2-B1;
616.            vec3_t v = I-B1;
617.            v = v - ((v*b)/square(b)) * b;  //component of vector B1I perpendicular to B1B2
618.
619.            return (v.z > 0.0f);
620.        }
621.
622.        bool classify(const pfacet_t solid, const int nfacet, const pvertex_t vertices, const int n
    vertex, pbounode_t& b, int& uppn, int& lown)
623.        {
624.            //classification of facets into internal, upper boundary and lower boundary
625.            b = (bounode_t*)malloc(sizeof(bounode_t)*nvertex);
626.            for(int i=0; i<nvertex; i++) b[i].first.v = NULL;
627.
628.            for(int i=0; i<nfacet; i++)
629.            {
630.                int cnt = 0;
631.                for(int j=0; j<3; j++)
632.                {
633.                    if(solid[i].a[j] == NULL)
634.                        cnt++;
635.                }
636.
637.                if(cnt == 3)  //stand-alone facet
638.                {
639.                    return false;
640.                }
641.                if(cnt == 2)   //we'll make two facets, each having just one edge at the boundary
642.                {
643.                    pfacet_t f1 = solid+i, f2 = f1->a[0];
644.                    if(f2 == NULL) f2 = f1->a[1];
645.                    if(f2 == NULL) f2 = f1->a[2];
646.                    if((f2->a[0] == NULL) || (f2->a[1] == NULL) || (f2->a[2] == NULL))
647.                    {
648.                        printf("defect in STL: incurable\n");
649.                        return false;
650.                    }
651.                    //f1 - current "bad" facet, f2 - its only neighbour
652.
653.                    pvertex_t A, B, C, D;  //vertices of the quadrilateral formed by two old facets
    ,
654.                    //for which we'll swap the diagonal
655.                    if((f1->v[0] == f2->v[0]) || (f1->v[0] == f2->v[1]) || (f1->v[0] == f2-
    >v[2]))
656.                    {
657.                        A = f1->v[0];
658.                        if((f1->v[1] == f2->v[0]) || (f1->v[1] == f2->v[1]) || (f1->v[1] == f2-
    >v[2]))
659.                        {
660.                            C = f1->v[1];
661.                            B = f1->v[2];
662.                        }
663.                        else
664.                        {
665.                            C = f1->v[2];
666.                            B = f1->v[1];
667.                        }
668.                    }
669.                    else
670.                    {
671.                        A = f1->v[1];
672.                        C = f1->v[2];
673.                        B = f1->v[0];
674.                    }
```

```
675.                    D = f2->v[0];
676.                    if((D==A) || (D==B) || (D==C)) D = f2->v[1];
677.                    if((D==A) || (D==B) || (D==C)) D = f2->v[2];
678.
679.                    pfacet_t fA, fC; //related facets, we'll edit their connections to the old face
    ts
680.                    pfacet_t *pA, *pC;
681.                    for(int j=0; j<3; j++)
682.                    {
683.                        if(f2->a[j] == f1) continue;
684.                        if((f2->a[j]->v[0] == A) || (f2->a[j]->v[1] == A) || (f2->a[j]-
    >v[2] == A))
685.                        {
686.                            fA = f2->a[j];
687.                            for(int k=0; k<3; k++)
688.                            {
689.                                if(fA->a[k] == f2) pA = fA->a+k;
690.                            }
691.                        }
692.                        if((f2->a[j]->v[0] == C) || (f2->a[j]->v[1] == C) || (f2->a[j]-
    >v[2] == C))
693.                        {
694.                            fC = f2->a[j];
695.                            for(int k=0; k<3; k++)
696.                            {
697.                                if(fC->a[k] == f2) pC = fC->a+k;
698.                            }
699.                        }
700.                    }
701.
702.                    //replace facet 1 with a new one
703.                    f1->v[0] = A;
704.                    f1->v[1] = B;
705.                    f1->v[2] = D;
706.                    f1->a[0] = f2;
707.                    f1->a[1] = fA;
708.                    *pA = f1;
709.                    f1->a[2] = NULL;
710.
711.                    //replace facet 2 with a new one
712.                    f2->v[0] = C;
713.                    f2->v[1] = B;
714.                    f2->v[2] = D;
715.                    f2->a[0] = f1;
716.                    f2->a[1] = fC;
717.                    *pC = f2;
718.                    f2->a[2] = NULL;
719.                }
720.            }
721.
722.            //now all facets have either 0 or 1 edge on the boundary
723.            uppn=0, lown=0;
724.            for(int i=0; i<nfacet; i++)
725.            {
726.                pvertex_t bv1, bv2, iv=NULL; //boundary facet has two boundary vertices and one int
    ernal
727.
728.                for(int j=0; j<3; j++)
729.                {
730.                    if(solid[i].a[j] == NULL) //missing adjacent facet -
    > current facet lies on boundary
731.                    {
732.                        iv  = solid[i].v[j];
733.                        bv1 = solid[i].v[(j+1)%3];
734.                        bv2 = solid[i].v[(j+2)%3];
735.                    }
736.                }
737.
738.                if(iv==NULL) //internal facet
739.                {
```

```
740.                     solid[i].type = internal_facet;
741.                 }
742.                 else
743.                 {
744.                     if(lower_boundary(*bv1, *bv2, *iv)) //lower boundary
745.                     {
746.                         solid[i].type = lower_facet;
747.                         lown++;
748.                     }
749.                     else  //upper boundary
750.                     {
751.                         solid[i].type = upper_facet;
752.                         uppn++;
753.                     }
754.
755.                     if(b[bv1-vertices].first.v==NULL)
756.                     {
757.                         b[bv1-vertices].first.v = bv2;
758.                         b[bv1-vertices].first.f = solid+i;
759.                     }
760.                     else
761.                     {
762.                         b[bv1-vertices].second.v = bv2;
763.                         b[bv1-vertices].second.f = solid+i;
764.                     }
765.
766.                     if(b[bv2-vertices].first.v==NULL)
767.                     {
768.                         b[bv2-vertices].first.v = bv1;
769.                         b[bv2-vertices].first.f = solid+i;
770.                     }
771.                     else
772.                     {
773.                         b[bv2-vertices].second.v = bv1;
774.                         b[bv2-vertices].second.f = solid+i;
775.                     }
776.                 }
777.             }
778.
779.             return true;
780.         }
781.
782.         void make_contour(bounode_t *b, pvertex_t v0, pfacet_t start, boundarel_t *c, int& n)
783.         {
784.             c[0].v = (start->a[0] == NULL) ? (start->v[1]) : (start->v[0]);
785.             c[0].f = b[c[0].v-v0].first.f;
786.             int i;
787.             for(i=0; (i==0) || (c[i].v != c[0].v); i++)
788.             {
789.                 c[i+1].v = (b[c[i].v-v0].first.f == c[i].f) ? b[c[i].v-v0].first.v : b[c[i].v-
    v0].second.v;
790.                 c[i+1].f = (b[c[i+1].v-v0].first.f != c[i].f) ? b[c[i+1].v-
    v0].first.f : b[c[i+1].v-v0].second.f;
791.             }
792.             n = i;
793.         }
794.
795.         void make_ccw(boundarel_t *c, int n)
796.         {
797.             float dir = *(c[n-1].v) ^ *(c[0].v);
798.             for(int i=0; i<n-1; i++) dir+= *(c[i].v) ^ *(c[i+1].v);
799.
800.             if(dir<0.0f)
801.             {
802.                 for(int i=0, j=n-1; i<j; i++, j--) //reverse contour
803.                 {
804.                     boundarel_t swp = c[i];  c[i] = c[j];  c[j] = swp;
805.                 }
806.
```

```
807.                pfacet_t tmp = c[0].f; //cyclic shift of facets to restore their correspondence to
    vertices
808.                for(int i=0; i<n-1; i++) c[i].f = c[i+1].f;
809.                c[n-1].f = tmp;
810.            }
811.        }
812.
813.        bool best_route(const boundarel_t *uppc, const int uppn, const boundarel_t *lowc, const int
     lown, pfacet_t lowstart,
814.                        steep_enough_t& steep_enough, ppatch_t& p, int& pn)
815.        {
816.            //if found pair of lower and upper contour elements is not steep enough, let's find som
    ething better
817.            for(int l = lowstart->id, r = lowstart->id; r-l < lown/2; l--, r++)
818.            {
819.                if(steep_enough(*(uppc[0].v), *(uppc[1].v), *(lowc[(l+lown)%lown].v), *(lowc[(l+1+l
    own)%lown].v)))
820.                {
821.                    lowstart = lowc[(l+lown)%lown].f;
822.                    break;
823.                }
824.                if(steep_enough(*(uppc[0].v), *(uppc[1].v), *(lowc[r%lown].v), *(lowc[(r+1)%lown].v
    )))
825.                {
826.                    lowstart = lowc[r%lown].f;
827.                    break;
828.                }
829.            }
830.            if(!steep_enough(*(uppc[0].v), *(uppc[1].v), *(lowc[lowstart->id].v), *(lowc[lowstart-
    >id+1].v)))
831.            {
832.                printf("surface sloped too much\n");
833.                printf("shit\n");
834.                return false;
835.            }
836.
837.            //now for each upper contour element we can determine valid segment in the lower contou
    r
838.            //from lim[i].first to lim[i].second
839.            pair<int, int> lim[uppn+1];
840.            lim[0].first = lowstart-
    >id;  //for the 0th upper boundarel we start from the found starting lower boundarel
841.            lim[0].second = lowstart->id;
842.            bool cancel = false;
843.            while((lim[0].second - lim[0].first < lown) && (!cancel)) //let's see how far we can go
     from the starting boundarel in the lower contour
844.            {
845.                cancel = true;
846.                //try to extend allowable interval backwards
847.                if(steep_enough(*(uppc[0].v), *(uppc[1].v),
848.                                *(lowc[(lim[0].first-
    1+lown)%lown].v), *(lowc[(lim[0].first+lown)%lown].v)))
849.                {
850.                    lim[0].first--;
851.                    cancel = false;
852.                }
853.                //try to extend allowable interval backwards
854.                if(steep_enough(*(uppc[0].v), *(uppc[1].v),
855.                                *(lowc[(lim[0].second)%lown].v), *(lowc[(lim[0].second+1)%lown].v))
    )
856.                {
857.                    lim[0].second++;
858.                    cancel = false;
859.                }
860.            }
861.            if(lim[0].first < 0)
862.            {
863.                lim[0].first += lown;
864.                lim[0].second+= lown;
865.            }
```

```
866.
867.            //now we find such interval for each upper boundarel
868.            for(int i=1; i<uppn; i++)
869.            {
870.                lim[i].first = lim[i-1].first;
871.                while((lim[i].first<=lim[i-
    1].second) && !steep_enough(*(uppc[i].v), *(uppc[i+1].v),
872.                                        *(lowc[(lim[i].first)%lown].v), *(lowc[(lim[i].first+1)%lown].v
    )))
873.                {
874.                    lim[i].first++;
875.                }
876.                if(!steep_enough(*(uppc[i].v), *(uppc[i+1].v),
877.                                        *(lowc[(lim[i].first)%lown].v), *(lowc[(lim[i].first+1)%lown].v
    )))
878.                {
879.                    printf("surface sloped too much\n");
880.                    return false;
881.                }
882.
883.                lim[i].second = lim[i-1].second;
884.                while((lim[i].second <= lim[0].second+lown) && steep_enough(*(uppc[i].v), *(uppc[i+
    1].v),
885.                                        *(lowc[(lim[i].second)%lown].v), *(lowc[(lim[i].second+1)%lown]
    .v)))
886.                {
887.                    lim[i].second++;
888.                }
889.            }
890.
891.            lim[uppn].first = lim[0].first + lown; //repeat first item, but now everything is shift
    ed one loop forward
892.            lim[uppn].second= lim[0].second+ lown;
893.
894.            //initialize table for dynamic programming
895.            routel_t * t[uppn+1];
896.            for(int i=0; i<uppn+1; i++)
897.            {
898.                t[i] = (routel_t*) malloc(sizeof(routel_t) * (lim[i].second-lim[i].first));
899.                t[i] -= lim[i].first;
900.            }
901.
902.            get_cost_t get_cost;
903.            for(int i=0; i<uppn; i++)
904.            {
905.                for(int j = lim[i].first; j < lim[i].second; j++)
906.                {
907.                    get_cost.setseed(uppc[i].f);
908.                    t[i][j].selfcost = get_cost(*(uppc[i].v), *(uppc[i+1].v), *(lowc[j%lown].v), *(
    lowc[(j+1)%lown].v));
909.                }
910.            }
911.            for(int j = lim[uppn].first; j<lim[uppn].second; j++)
912.                t[uppn][j] = t[0][j-lown];
913.
914.            //let's run dynamic programming!
915.            p = (patch_t*)malloc(sizeof(patch_t)*(uppn+lown+1));
916.            float min = LARGE;
917.            //we try each possible patch between 0th upper boundarel and valid lower boundarel as s
    tarting patch
918.            //and find the cost of a route until the same patch (cycle)
919.            for(int k = lim[0].first; k < lim[0].second; k++)
920.            {
921.                for(int i=0; i<uppn+1; i++)
922.                {
923.                    for(int j = lim[i].first; j < lim[i].second; j++)
924.                    {
925.                        t[i][j].from = indef;
926.                        t[i][j].routecost = LARGE*LARGE;
927.                    }
```

```
928.                }
929.                t[0][k].from = start; //current starting patch for which we'll find optimal cycle
930.
931.                bool promising = true;
932.                for(int i=0; (i<uppn+1) && promising; i++)
933.                {
934.                    promising = false;
935.                    for(int j = lim[i].first; j < lim[i].second; j++)
936.                    {
937.                        proutel_t r = &(t[i][j]);
938.                        r->routecost = r->selfcost;
939.                        switch(r->from)
940.                        {
941.                            case west:   r->routecost += t[i-1][j].routecost; break;
942.                            case north:  r->routecost += t[i][j-1].routecost; break;
943.                            case norwest:r->routecost += t[i-1][j-1].routecost; break;
944.                            case indef:  r->routecost = LARGE*LARGE; continue;
945.                            case start:  r->routecost = 0.0f;
946.                        }
947.                        if(r->routecost > min) continue; //search of optimum is hopeless
948.                        else promising = true;  //we may still improve our result
949.
950.                        if(j+1 < lim[i].second)                                 //try to improve i,j
    -> i,j+1
951.                            t[i][j+1].improve(r, north);
952.                        if(i<uppn)
953.                        {
954.                            if((j >= lim[i+1].first) && (j < lim[i+1].second))     //try to improve
     i,j -> i+1,j
955.                                t[i+1][j].improve(r, west);
956.                            if((j+1 >= lim[i+1].first) && (j+1 < lim[i+1].second)) //try to improve
     i,j -> i+1,j+1
957.                                t[i+1][j+1].improve(r, norwest);
958.                        }
959.                    }
960.                }
961.
962.                proutel_t r = &(t[uppn][k+lown]);
963.                if(r->routecost < min)  //better cycle is found -> we should save the route
964.                {
965.                    min = r->routecost;
966.                    pn=uppn+lown;
967.                    for(int i=uppn, j=k+lown; t[i][j].from!=start; pn--)
968.                    {
969.                        p[pn].upp = uppc[i].f;
970.                        p[pn].low = lowc[j%lown].f;
971.
972.                        switch(t[i][j].from)
973.                        {
974.                            case west:   i--; break;
975.                            case north:  j--; break;
976.                            case norwest:i--; j--; break;
977.                        }
978.                    }
979.                }
980.            }
981.            if(min >= LARGE) return false;
982.
983.            //shift route to the beginning of the array
984.            for(int i=pn; i<uppn+lown; i++)
985.            {
986.                p[i-pn+1] = p[i+1];
987.            }
988.            pn = uppn+lown-pn;
989.            p[0] = p[pn];
990.
991.            for(int i=0; i++; i<uppn+1)
992.            {
993.                t[i] += lim[i].first;
994.                free(t[i]);
```

```
995.            }
996.
997.            return true;
998.        }
999.
1000.       void stitch_patches(boundarel_t *uppc, int uppn, boundarel_t *lowc, int lown, patch_t *p, i
    nt pn,
1001.                           pcheckpoint_t& cp, int& cpn, pcheckpoint_t branch)
1002.       {
1003.           //assembling patches in one route
1004.           float uppselflen[uppn], uppcouplen[uppn];
1005.           float lowselflen[lown], lowcouplen[lown];
1006.
1007.           uppc[uppn] = uppc[0];  lowc[lown] = lowc[0];
1008.           for(int i=0; i<uppn; i++)
1009.           {
1010.               uppcouplen[i] = 0.0f;
1011.               uppselflen[i] = magnitude(*(uppc[i+1].v) - *(uppc[i].v));
1012.           }
1013.           for(int i=0; i<lown; i++)
1014.           {
1015.               lowcouplen[i] = 0.0f;
1016.               lowselflen[i] = magnitude(*(lowc[i+1].v) - *(lowc[i].v));
1017.           }
1018.           for(int i=0; i<pn; i++)
1019.           {
1020.               uppcouplen[p[i].upp->id] += lowselflen[p[i].low->id];
1021.               lowcouplen[p[i].low->id] += uppselflen[p[i].upp->id];
1022.           }
1023.           p[pn] = p[0];
1024.
1025.
1026.           float upplen=0.0f, lowlen=0.0f;
1027.           for(int i=pn-1; p[i].upp == p[0].upp; i--)
1028.           {
1029.               lowlen+= lowselflen[p[i].low->id];
1030.           }
1031.           for(int i=pn-1; p[i].low == p[0].low; i--)
1032.           {
1033.               upplen+= uppselflen[p[i].upp->id];
1034.           }
1035.           vec3_t upper, lower;
1036.           cp[cpn++] = *branch;
1037.           cp[cpn-1].next = branch->next;
1038.           for(int i=0; i<pn; i++)
1039.           {
1040.               upper = interpolate(*(uppc[p[i].upp->id].v), *(uppc[p[i].upp-
    >id+1].v), lowlen/uppcouplen[p[i].upp->id]);
1041.               lower = interpolate(*(lowc[p[i].low->id].v), *(lowc[p[i].low-
    >id+1].v), upplen/lowcouplen[p[i].low->id]);
1042.               if(p[i].upp == p[i+1].upp) lowlen+= lowselflen[p[i].low->id];
1043.               else lowlen = 0.0f;
1044.               if(p[i].low == p[i+1].low) upplen+= uppselflen[p[i].upp->id];
1045.               else upplen = 0.0f;
1046.               cp[cpn++] = checkpoint_t(upper, lower, cp+cpn);
1047.           }
1048.
1049.           float dist, min = LARGE;
1050.           pcheckpoint_t closest, before_closest;
1051.           for(int i=cpn-pn; i<cpn; i++)
1052.           {
1053.               dist = square(branch->upp - cp[i].upp);
1054.               if(dist < min)
1055.               {
1056.                   min = dist;
1057.                   closest = cp+i;
1058.                   before_closest = (i>cpn-pn) ? (closest-1) : (closest+pn-1);
1059.               }
1060.           }
1061.           branch->next = closest;
```

```
1062.           cp[cpn-1].next = cp+cpn-pn;
1063.           cp[cpn++] = *closest;
1064.           cp[cpn-1].next = cp+cpn-pn-2;
1065.           before_closest->next = cp+cpn-1;
1066.       }
1067.
1068.       int main(int argc, char* argv[])
1069.       {
1070.           //Initialize and clear screen
1071.           LCD my_lcd;
1072.           dLcdInit(my_lcd.Lcd);
1073.           LCDClear(my_lcd.Lcd);
1074.
1075.           if(argc < 2)
1076.           {
1077.               dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 30, NORMAL_FONT, (signed char*)"no model is p
    rovided");
1078.               dLcdUpdate(&my_lcd);
1079.               usleep(DISPLAY_SLEEP);
1080.               return 1;
1081.           }
1082.
1083.           dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 1, TINY_FONT, (signed char*)argv[1]);
1084.           dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 10, NORMAL_FONT, (signed char*)"model is processe
    d...");
1085.           dLcdUpdate(&my_lcd);
1086.
1087.           pfacet_t solid=NULL;
1088.           int nfacet;
1089.           pvertex_t vertices=NULL;
1090.           int nvertex;
1091.
1092.           if(!input(argv[1], solid, nfacet, vertices, nvertex))
1093.           {
1094.               dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 30, NORMAL_FONT, (signed char*)"something is
    wrong with the model");
1095.               dLcdUpdate(&my_lcd);
1096.               usleep(DISPLAY_SLEEP);
1097.               return 1;
1098.           }
1099.
1100.           pbounode_t boundary;
1101.           int uppn, lown;
1102.           if(!classify(solid, nfacet, vertices, nvertex, boundary, uppn, lown))
1103.           {
1104.               dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 30, NORMAL_FONT, (signed char*)"defects in ST
    L");
1105.               dLcdUpdate(&my_lcd);
1106.               usleep(DISPLAY_SLEEP);
1107.               return 1;
1108.           }
1109.
1110.           boundarel_t *uppc, *lowc; //upper, lower contour
1111.           uppc = (boundarel_t*)malloc(sizeof(boundarel_t)*(uppn+1));
1112.           lowc = (boundarel_t*)malloc(sizeof(boundarel_t)*(lown+1));
1113.           checkpoint_t *cp;
1114.           cp = (checkpoint_t*)malloc(sizeof(checkpoint_t)*4*(uppn+lown+1));
1115.           int cpn = 0;
1116.           cp[cpn++] = checkpoint_t(ZAXIS, ORIGIN, NULL);
1117.           int uppleft = uppn;
1118.
1119.           while(uppleft>0)
1120.           {
1121.               float min = LARGE;
1122.               pfacet_t uppstart=NULL, lowstart=NULL;
1123.               pcheckpoint_t branch=NULL;
1124.               for(int i=0; i<nfacet; i++)
1125.               {
1126.                   if(solid[i].type == upper_facet)
1127.                   {
```

```
1128.                       vec3_t middle;
1129.                       for(int j=0; j<3; j++)
1130.                       {
1131.                           if(solid[i].a[j] == NULL)
1132.                           {
1133.                               middle = 0.5f * (*(solid[i].v[(j+1)%3]) + *(solid[i].v[(j+2)%3]));

1134.                           }
1135.                       }
1136.                       for(int j=0; j<cpn; j++)
1137.                       {
1138.                           float dist = square(middle - cp[j].upp);
1139.                           if(dist < min)
1140.                           {
1141.                               min = dist;
1142.                               uppstart = solid+i;
1143.                               branch = cp+j;
1144.                           }
1145.                       }
1146.                   }
1147.               }
1148.               if(uppstart == NULL)
1149.               {
1150.                   dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 30, NORMAL_FONT, (signed char*)"no upper
    contour");
1151.                   dLcdUpdate(&my_lcd);
1152.                   usleep(DISPLAY_SLEEP);
1153.                   return 1;
1154.               }
1155.
1156.               //make upper and lower contours in CCW direction (positive against Z-axis)
1157.               make_contour(boundary, vertices, uppstart, uppc, uppn);
1158.               for(int i=0; i<uppn; i++) uppc[i].f->type = used_facet;
1159.               make_ccw(uppc, uppn);
1160.               uppstart = uppc[0].f;
1161.               uppleft -= uppn;
1162.
1163.               steep_enough_t steep_enough(MAX_INCLINATION);   //steepness tester
1164.
1165.               lowstart = find_other_side(solid, uppstart, nfacet, lower_facet, &steep_enough);
1166.               if(lowstart == NULL)
1167.               {
1168.                   dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 30, NORMAL_FONT, (signed char*)"no lower
    contour");
1169.                   dLcdUpdate(&my_lcd);
1170.                   usleep(DISPLAY_SLEEP);
1171.                   return 1;
1172.               }
1173.               make_contour(boundary, vertices, lowstart, lowc, lown);
1174.               make_ccw(lowc, lown);
1175.               for(int i=0; i<uppn; i++) uppc[i].f->id = i;
1176.               for(int i=0; i<lown; i++) lowc[i].f->id = i;
1177.               uppc[uppn] = uppc[0];  lowc[lown] = lowc[0];
1178.
1179.               patch_t *p; //storage for optimal route found by DP (maximal possible length)
1180.               int pn;
1181.               if(!best_route(uppc, uppn, lowc, lown, lowstart, steep_enough, p, pn))
1182.               {
1183.                   dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 30, NORMAL_FONT, (signed char*)"impossibl
    e to cut");
1184.                   dLcdUpdate(&my_lcd);
1185.                   usleep(DISPLAY_SLEEP);
1186.                   return 1;
1187.               }
1188.
1189.               stitch_patches(uppc, uppn, lowc, lown, p, pn, cp, cpn, branch);
1190.               free(p);
1191.           }
1192.
1193.           free(solid);
```

```
1194.           free(vertices);
1195.           free(boundary);
1196.           free(uppc);
1197.           free(lowc);
1198.           //cp is freed later
1199.
1200.           //make position at the origin the same height as the connected one
1201.           cp[0].upp.z = cp[0].next->upp.z;
1202.           cp[0].low.z = cp[0].next->low.z;
1203.           cp[1].upp.z = cp[0].next->upp.z;
1204.           cp[1].low.z = cp[0].next->low.z;
1205.
1206.           if(!motor_t::init()) return 1;
1207.
1208.           buttons_t buttons;
1209.
1210.           // initializing machine
1211.           coordinates_t coordinates(cp, cpn, CUTTING_SPEED);
1212.
1213.           motor_t *motor[MAX_MECH_DOF];
1214.           motor[steer] = new motor_t(MOTOR_PORT_D, 0.1f, 1.0f, 0.6e5f, coordinates.state+steer);
1215.           motor[tilt]  = new motor_t(MOTOR_PORT_C, 0.1f, 1.0f, 0.6e5f, coordinates.state+tilt);
1216.           motor[drive] = new motor_t(MOTOR_PORT_A, 3.0f, 0.0f, 1.0e5f, coordinates.state+drive);
1217.           motor[shift] = new motor_t(MOTOR_PORT_B, 3.0f, 0.0f, 1.0e5f, coordinates.state+shift);
1218.
1219.           LCDClear(my_lcd.Lcd);
1220.           dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 1, TINY_FONT, (signed char*)argv[1]);
1221.           dLcdDrawText(my_lcd.Lcd, FG_COLOR, 50, 8, SMALL_FONT, (signed char*)"error  power");
1222.           dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 16, SMALL_FONT, (signed char*)"steer");
1223.           dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 26, SMALL_FONT, (signed char*)"tilt");
1224.           dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 36, SMALL_FONT, (signed char*)"drive");
1225.           dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 46, SMALL_FONT, (signed char*)"shift");
1226.           dLcdDrawText(my_lcd.Lcd, FG_COLOR, 1, 80, LARGE_FONT, (signed char*)"compl.");
1227.
1228.           char s[100];
1229.           int cnt = 0;
1230.           bool alright = true;
1231.           coordinates.runchrono();
1232.
1233.           while(!coordinates.finished() && alright) {
1234.               for(int i=0; i<MAX_MECH_DOF; i++)
1235.               {
1236.                   coordinates.update(i);
1237.                   motor[i]->act();
1238.               }
1239.               if(cnt%256 == 0)
1240.               {
1241.                   for(int i=0; i<MAX_MECH_DOF; i++)
1242.                   {
1243.                       sprintf(s, "%4.0f%4d", motor[i]->pos, motor[i]->power);
1244.                       dLcdDrawText(my_lcd.Lcd, FG_COLOR, 50, 16+10*i, NORMAL_FONT, (signed char*)
    s);
1245.                   }
1246.                   sprintf(s, "%3d%%", 100*(coordinates.t/1024)/(coordinates.tlast/1024));
1247.                   dLcdDrawText(my_lcd.Lcd, FG_COLOR, 100, 80, LARGE_FONT, (signed char*)s);
1248.                   dLcdUpdate(&my_lcd);
1249.               }
1250.
1251.               usleep(CYCLE_SLEEP);
1252.
1253.               if(buttons.keypressed()) alright = false;
1254.
1255.               cnt++;
1256.           }
1257.
```

Appendix 2. C++ Foam Cutter Program Stultor

```
            for(int i=0; i<MAX_MECH_DOF; i++) {
                delete motor[i];
            }
            motor_t::deinit();
            dLcdExit();

            return 0;
        }
```

Appendix 3. Media Files about Foam Cutting

Cutting octogonal frustum (sloped) https://youtu.be/3NRxQBad2iE

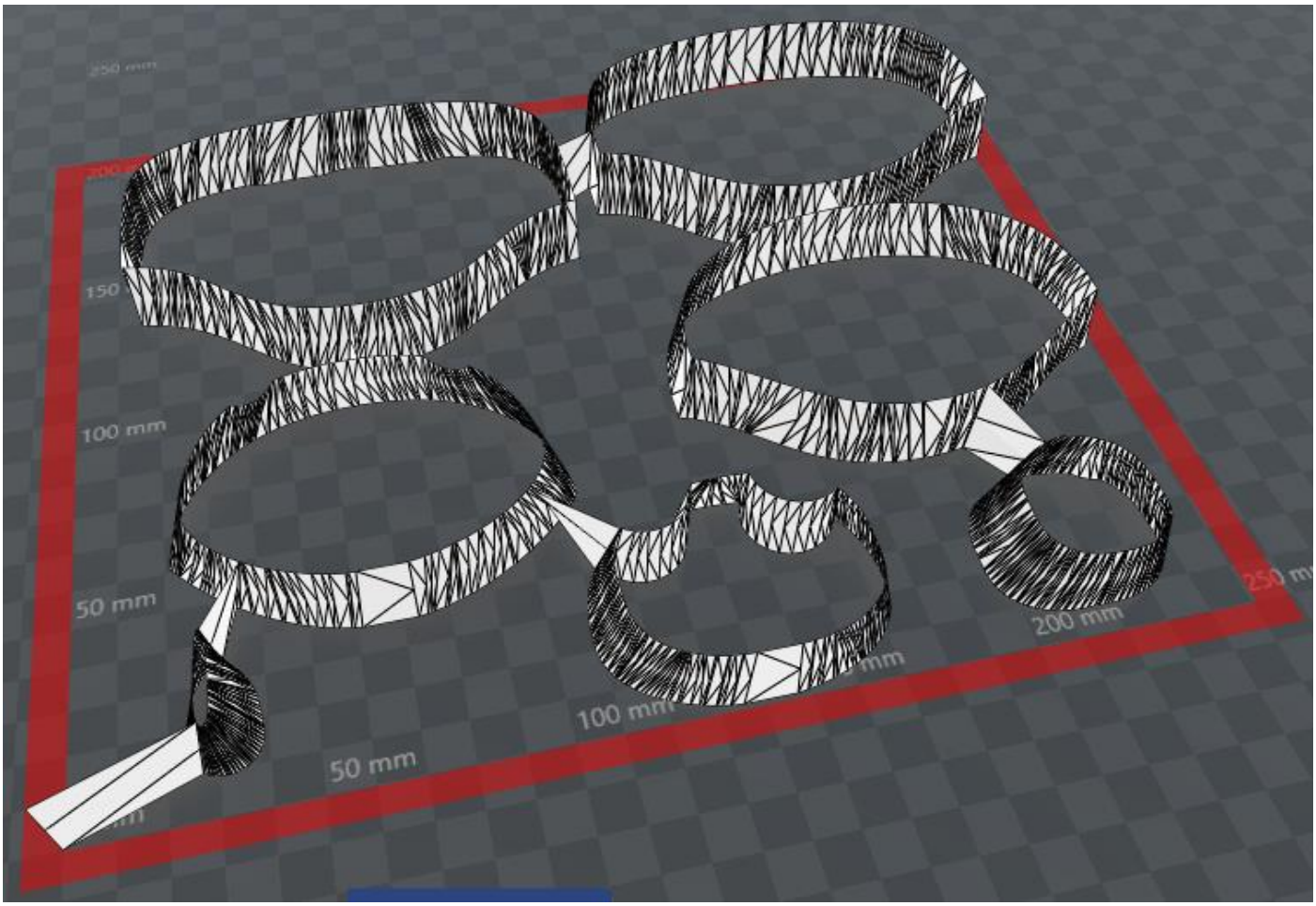
Cutting octogonal frustum (steeper) https://youtu.be/YDKHIMDHst0

Cutting nested contours https://youtu.be/62PK5w1T_YM

Cutting a set of 7 Yak-130 pieces https://youtu.be/UVwoC9tWKEY

Cutting vertical stabilizer https://youtu.be/QRS_CMgMO4Y

Yak-130 set of 7 parts with a tool path:



Appendix 3. Media Files about Foam Cutting

Yak-130 cut parts:

Yak-130 assembled: