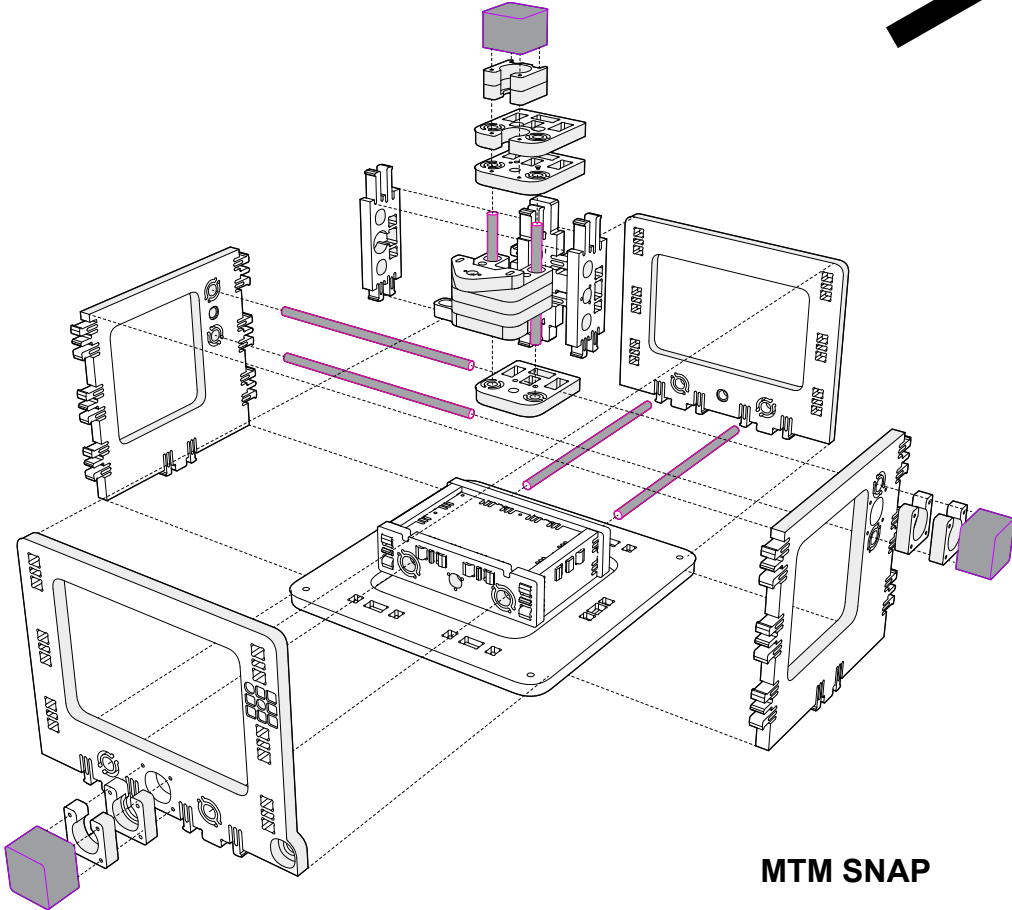


How to Make Almost Anything *Machine!*



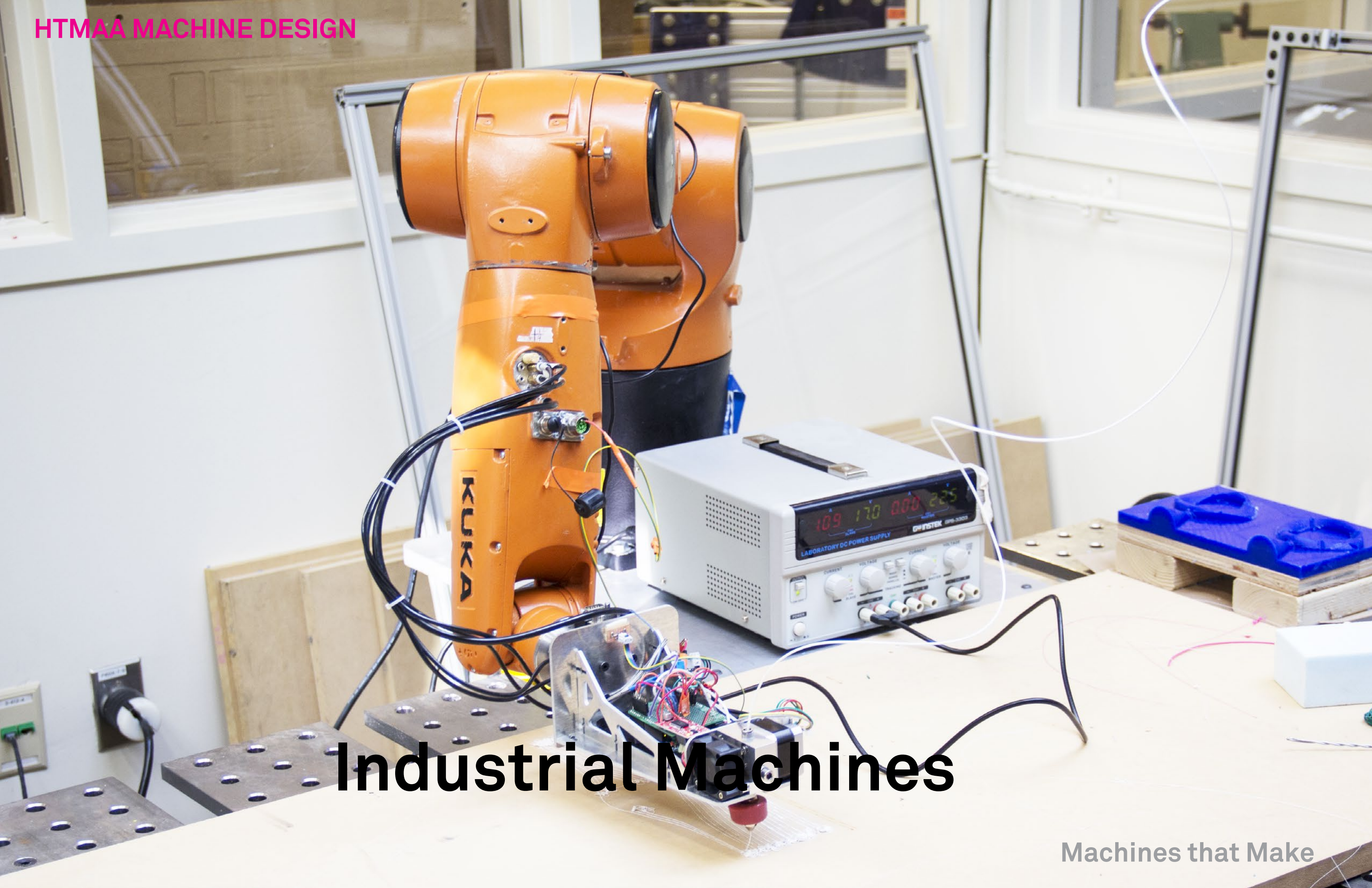


Industrial Machines



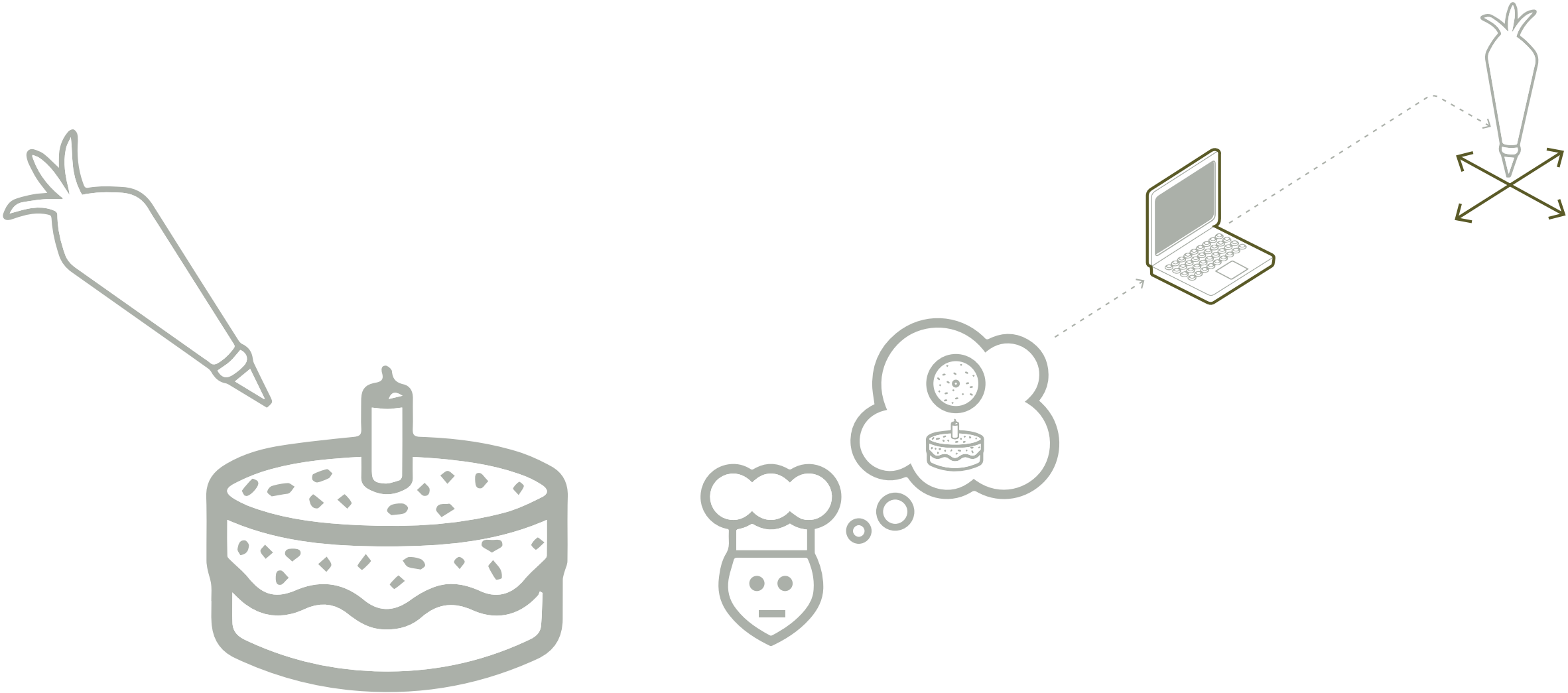
Industrial Machines

Industrial Machines





Consumer machines



Personal Fabrication

Machines that Make

The Machine that make project at the [MIT Center for Bits and Atoms](#) seeks to develop low-cost machines that can be made using CNC equipment, like available in [fab labs](#).

[m]MTM: modular machines that make



More modular machines out of prototyping materials for prototyping

Reconfigurable Stages



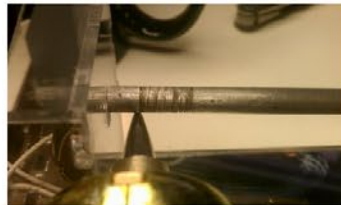
Reconfigurable one-axis stages for multi-purpose motion.

foldafab



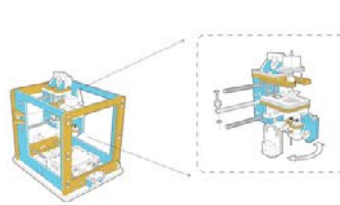
A deployable medium-format CNC router.

DIY EDM



An entry level (under \$500) EDM machine for making carbide/HSS tooling and/or lead screws

5 Axis Timing Belt MTM



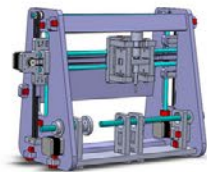
Low cost 5 axis machining.

POP Fab



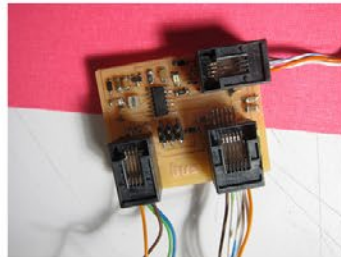
A suitcase milling machine, 3d printer, and vinyl cutter.

Multi-processes lathe



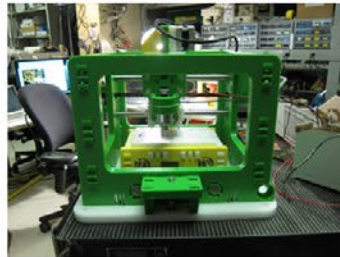
The additive lathe is a 3D printer that prints on rotation objects.

Virtual Machine Network



Modular control for the MTM project.

Timing Belt MTM

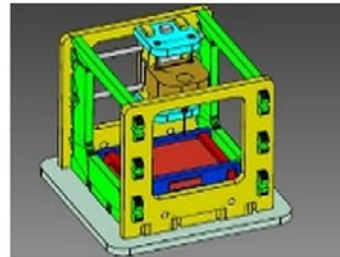


A design without lead screws, reducing cost.

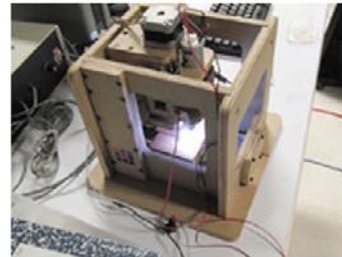
Fab-In-A-Box



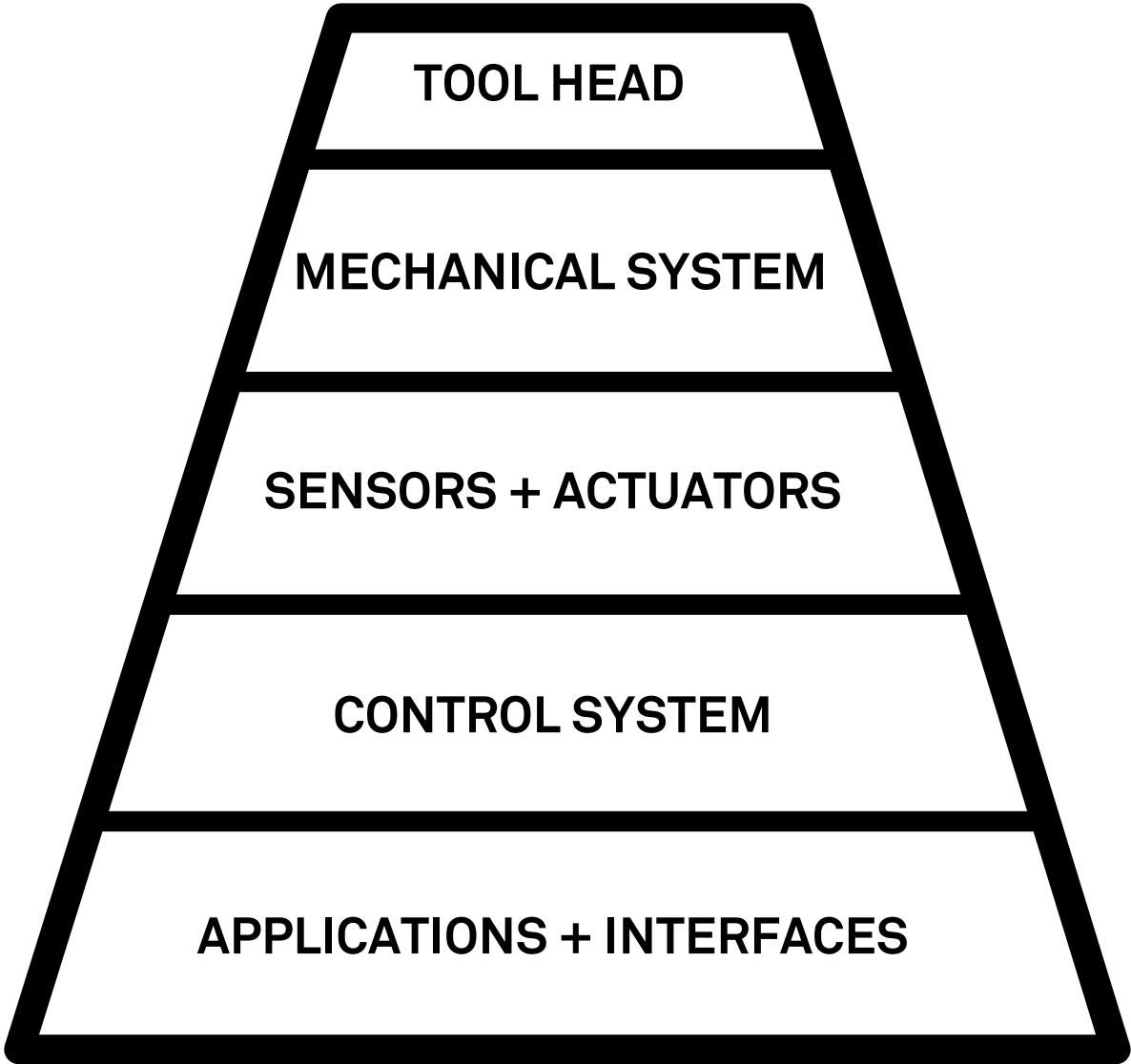
MtM Snap-Lock

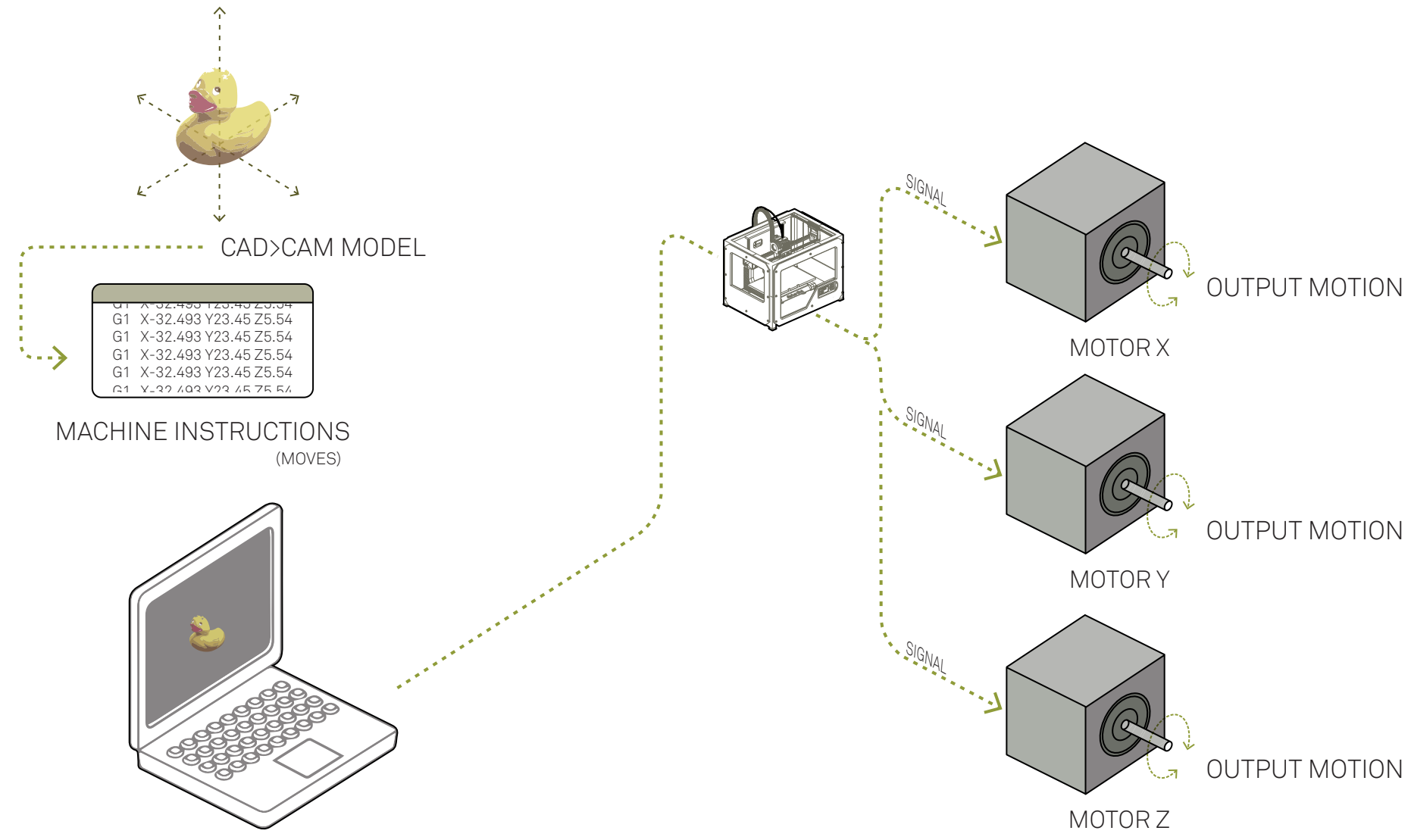


MtM A-Z



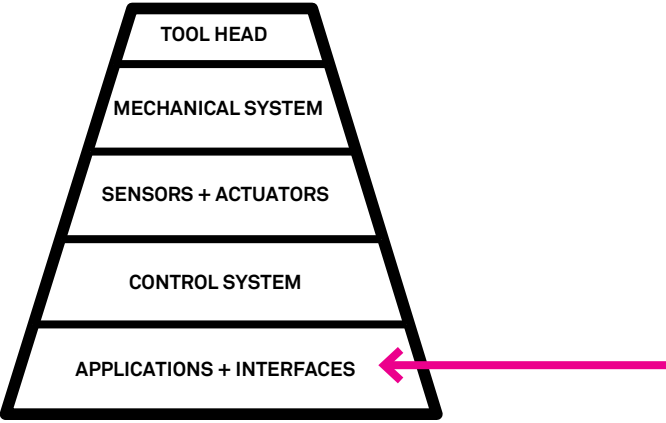
**MAKING
MACHINES THAT MAKE**





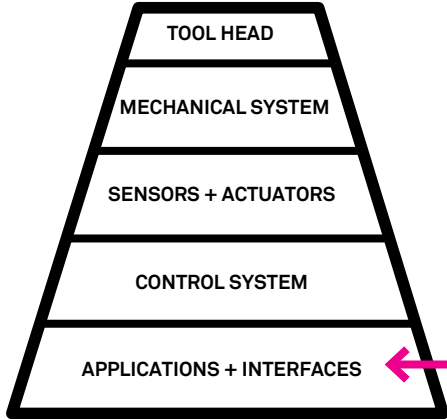
Machine programming and control

HTMAA MACHINE DESIGN



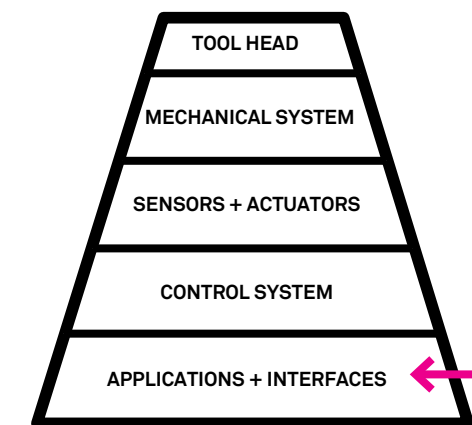
Machines that Make

HTMAA MACHINE DESIGN



Machines that Make

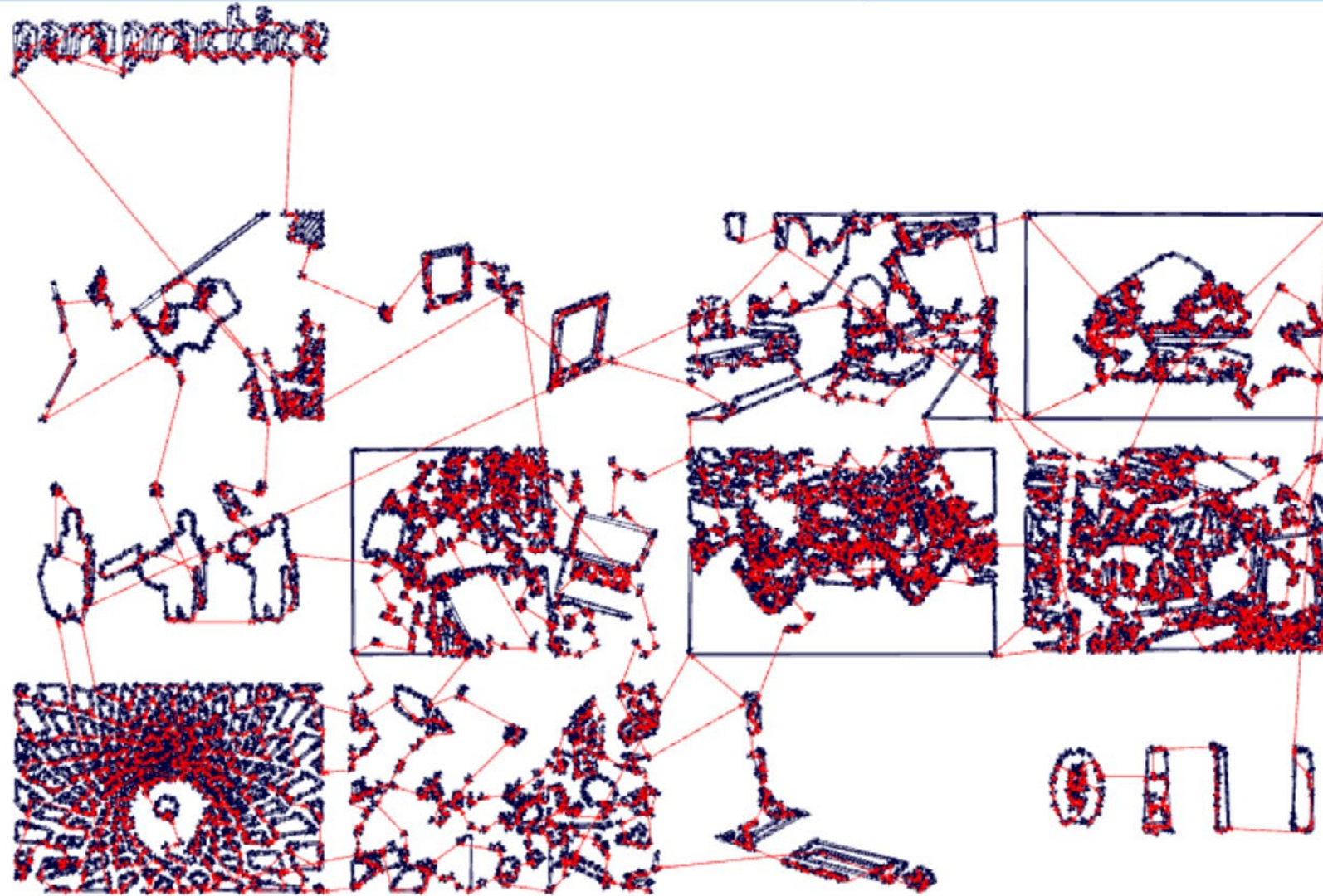
HTMAA MACHINE DESIGN



Machines that Make

left: pan, scroll: zoom, right: rotate, c: connections

image (.png) Roland MDX-20 (.rml) PCB traces (1/64)



input

file: paraprac.png
dpi: 72.009
size:
1187 x 794 px
418.695 x 280.071 mm
16.484 x 11.026 in

output

speed (mm/s): 4
jog height (mm): 1
xmin (mm): 20
ymin (mm): 20

process

send command:

send server:

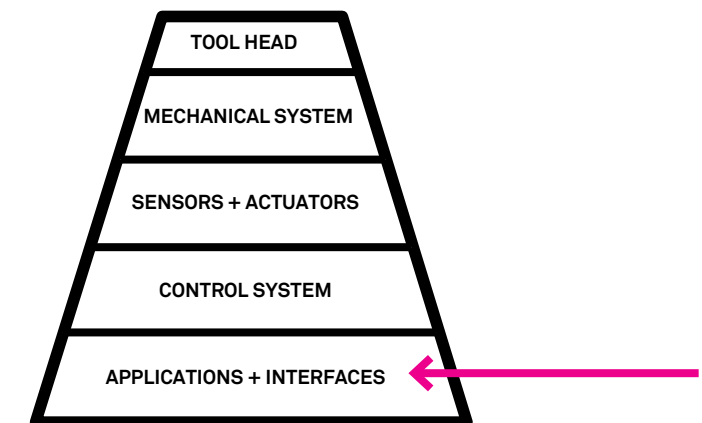
conventional climb
cut depth (mm):

tool diameter (mm):

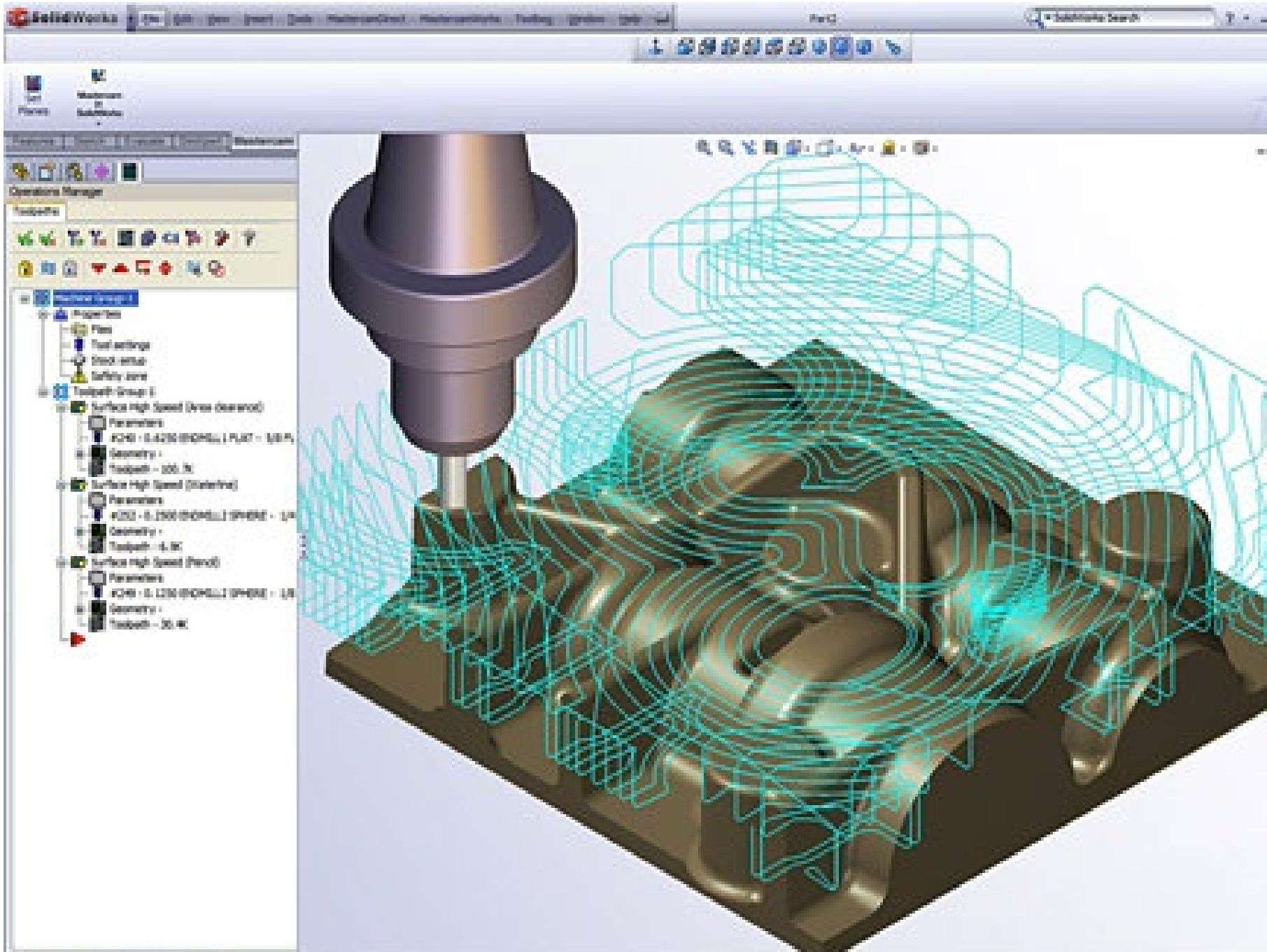
number of offsets (-1 to fill):

offset overlap (%):

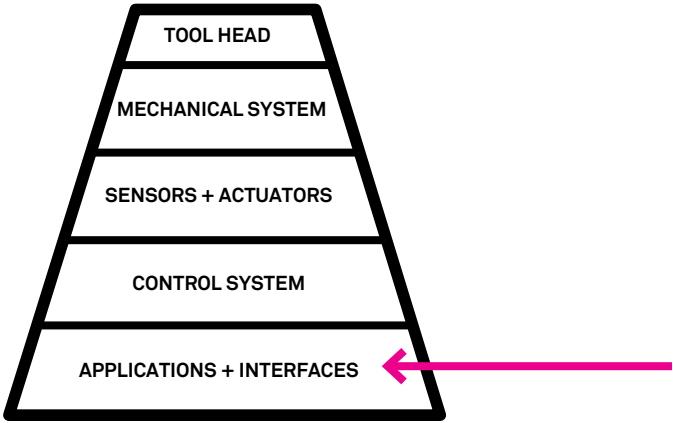
path error (pixels):

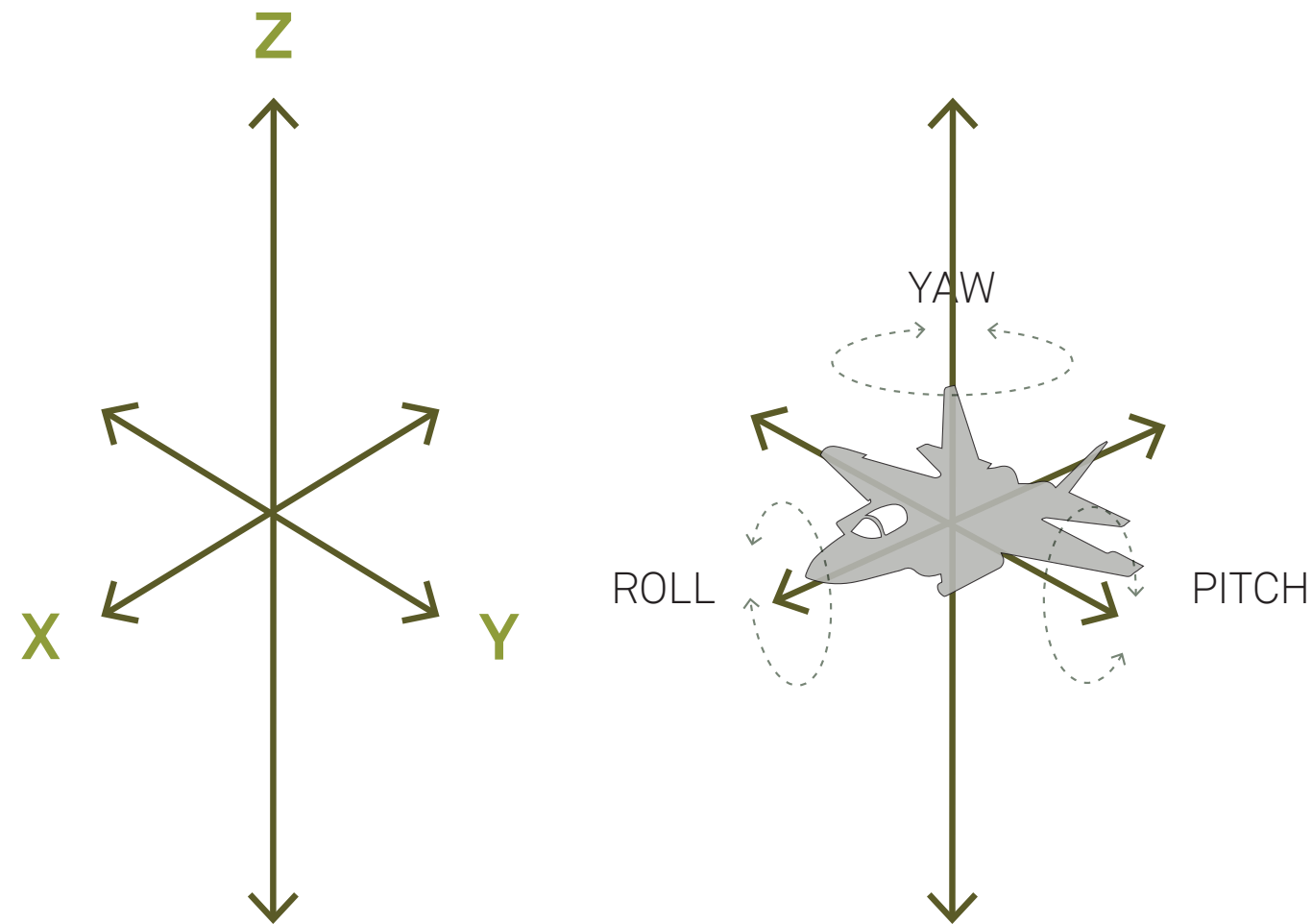


Toolpathing



Toolpathing



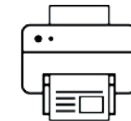


1.0 AXIS



(X)

2.0 AXIS



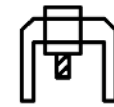
(X,Y)

2.5 AXIS



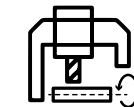
(Z)(X,Y)

3.0 AXIS



(X,Y,Z)

4.0 AXIS



(C)(X,Y,Z)

5.0 + AXIS



(X,Y,Z,A,B,C)+

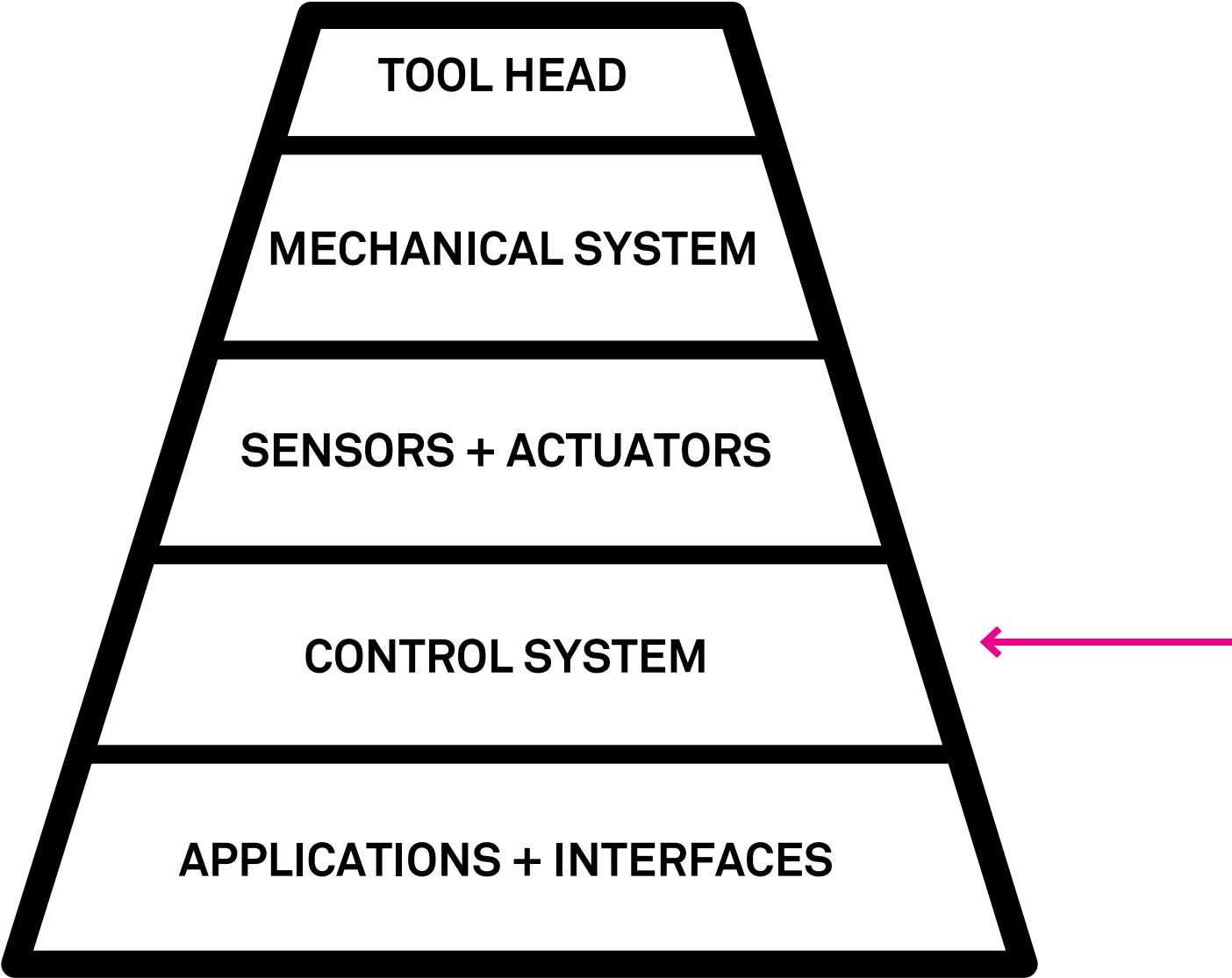
Machine instructions/ post processors

HTMAA MACHINE DESIGN

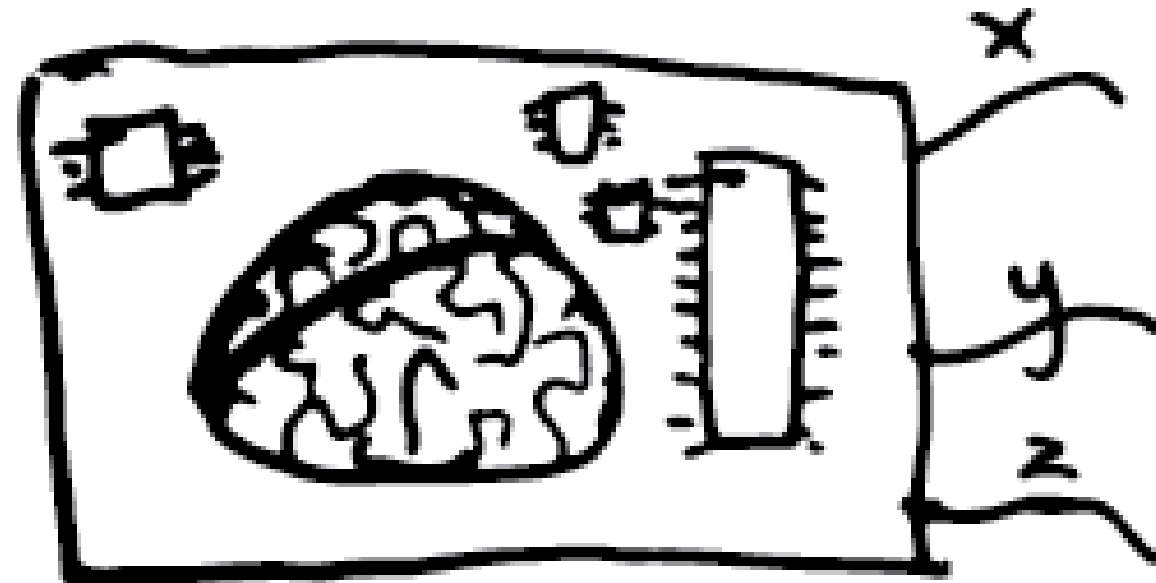
G-code	Functions	G-code	Functions
G0	Rapid positioning	G53	Move in absolute machine coordinate system
G1	Linear interpolation	G54 à G59	Use fixture offset 1 to 6, G59 to select a general fixture number
G2	Clockwise circular / helical interpolation	G61	Exact Stop mode
G3	Counterclockwise circular / helical interpolation	G64	Constant Velocity mode
G4	Dwell	G73	Canned cycle - drilling - fast pullback
G10	Coordinate system origin setting	G80	Cancel canned cycle mode
G12	Clockwise circular pocket	G81	Canned cycle - drilling
G13	Counterclockwise circular pocket	G82	Canned cycle - drilling with dwell
G15	Polar Coordinate moves in G0 and G1	G83	Canned cycle - peck drilling
G16	Cancel polar Coordinate moves in G0 and G1	G84	Canned cycle - right hand rigid taping (not yet implemented)
G17	XY plane select	G85	Canned cycle - boring, no dwell, feed out
G18	XZ plane select	G86	Canned cycle - boring, spindle stop, rapid out
G19	YZ plane select	G87	Canned cycle - back boring (not yet implemented)
G20	Inch unit	G88	Canned cycle - boring, spindle stop, manual out
G21	Millimeter unit	G89	Canned cycle - boring, dwell, feed out
G28	Return machine home (parameters 5161 to 5166)	G90	Absolute distance mode
G30	Return machine home (parameters 5181 to 5186)	G91	Incremental distance mode
G28.1	Reference axis	G92	Offset coordinates and set parameters
G31	Straight Probe	G92.1	Reset G92 offset and parameter
G40	Cancel cutter radius compensation	G92.2	Reset G92 offset but leave parameters untouched
G41	Start cutter radius compensation left	G92.3	Recall G92 from parameters
G42	Start cutter radius compensation right	G93	Inverse time feed mode
G43	Apply tool length offset (plus)	G94	Feed per minute mode
G49	Cancel tool length offset	G95	Feed per revolution mode
G50	Reset all scale factors to 1.0	G98	Initial level return after canned cycles
G51	Set axis data input scale factors	G99	R-point level return after canned cycles

G-code (and M-code)

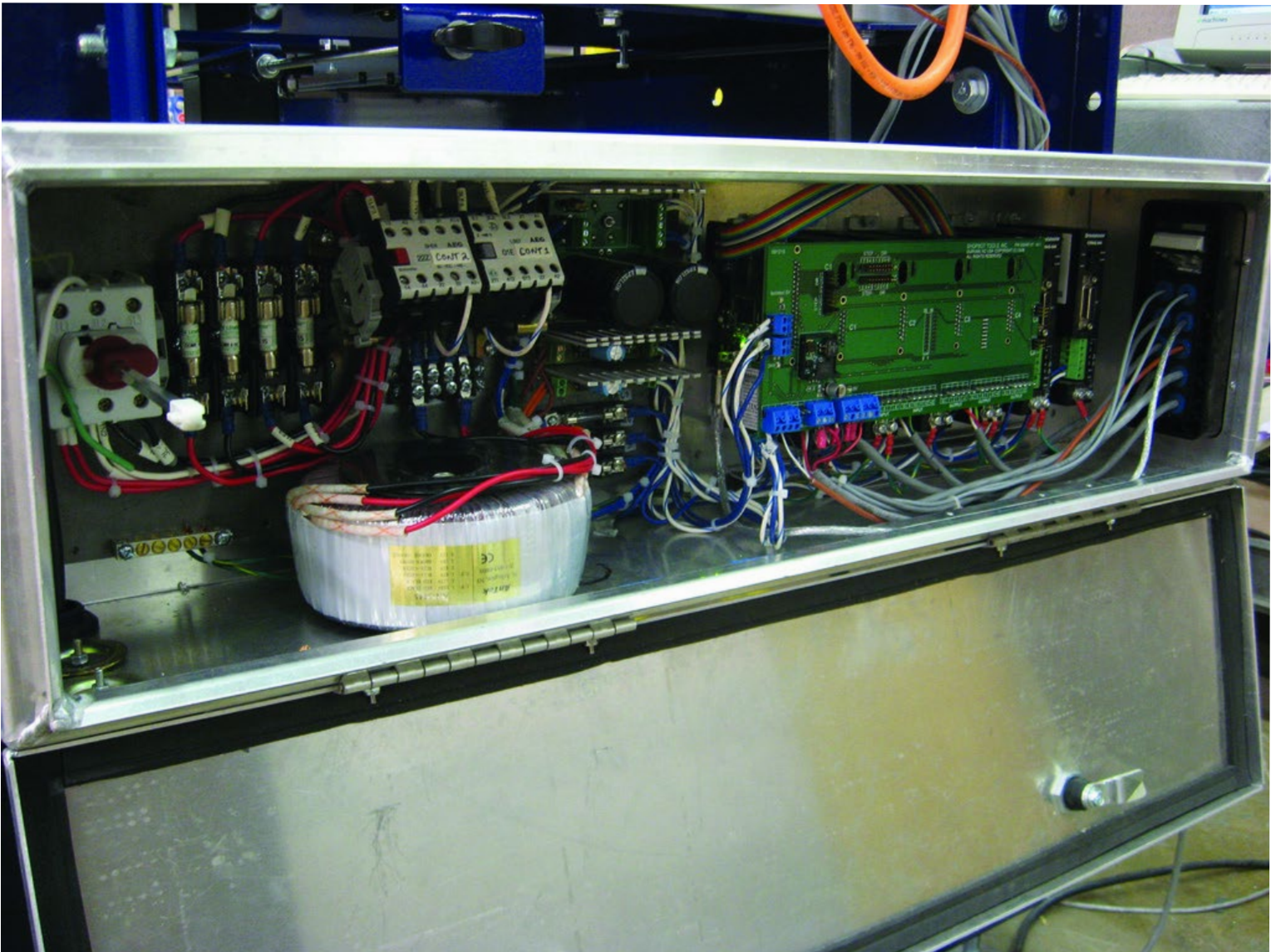
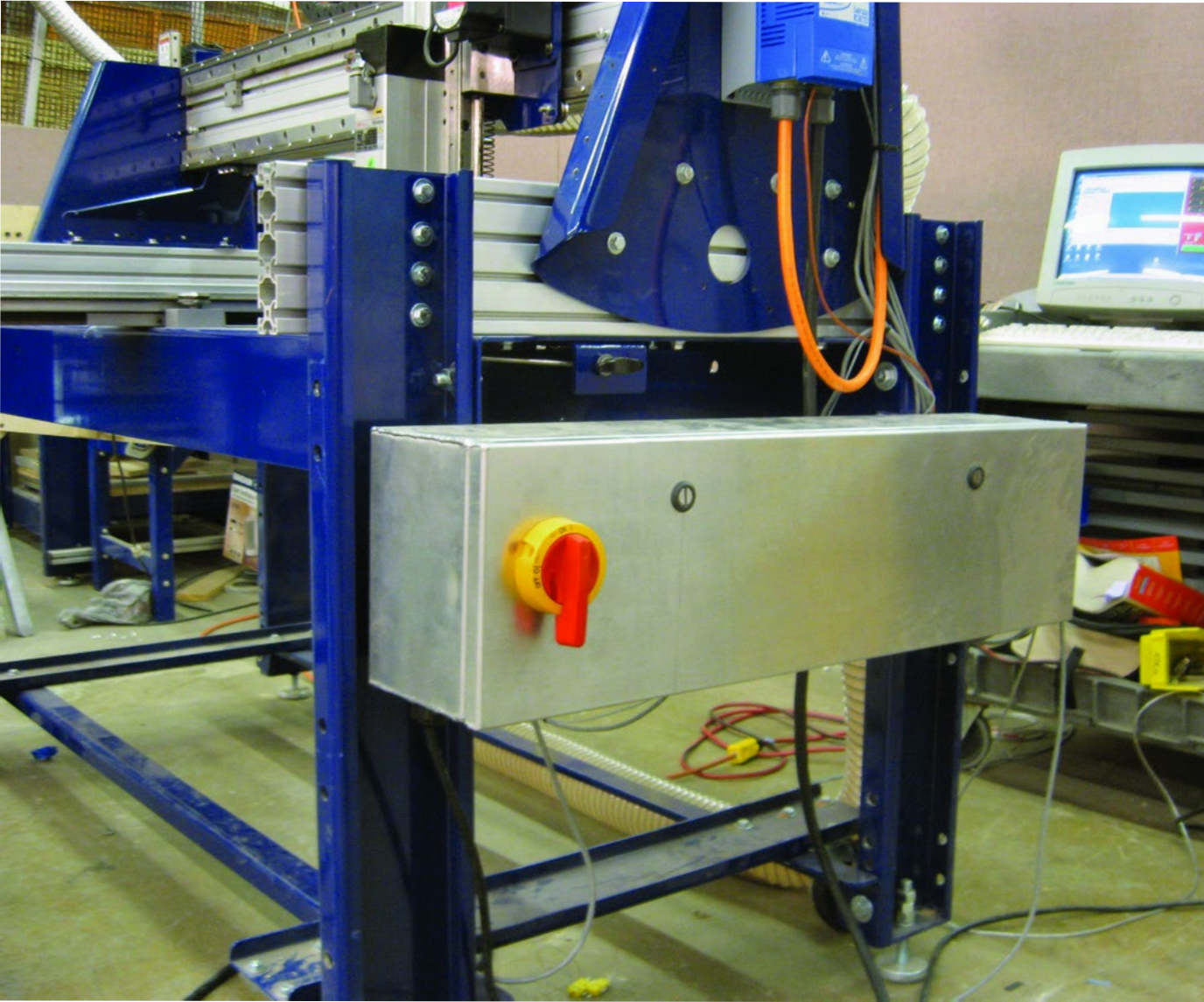
**MAKING
MACHINES THAT MAKE**



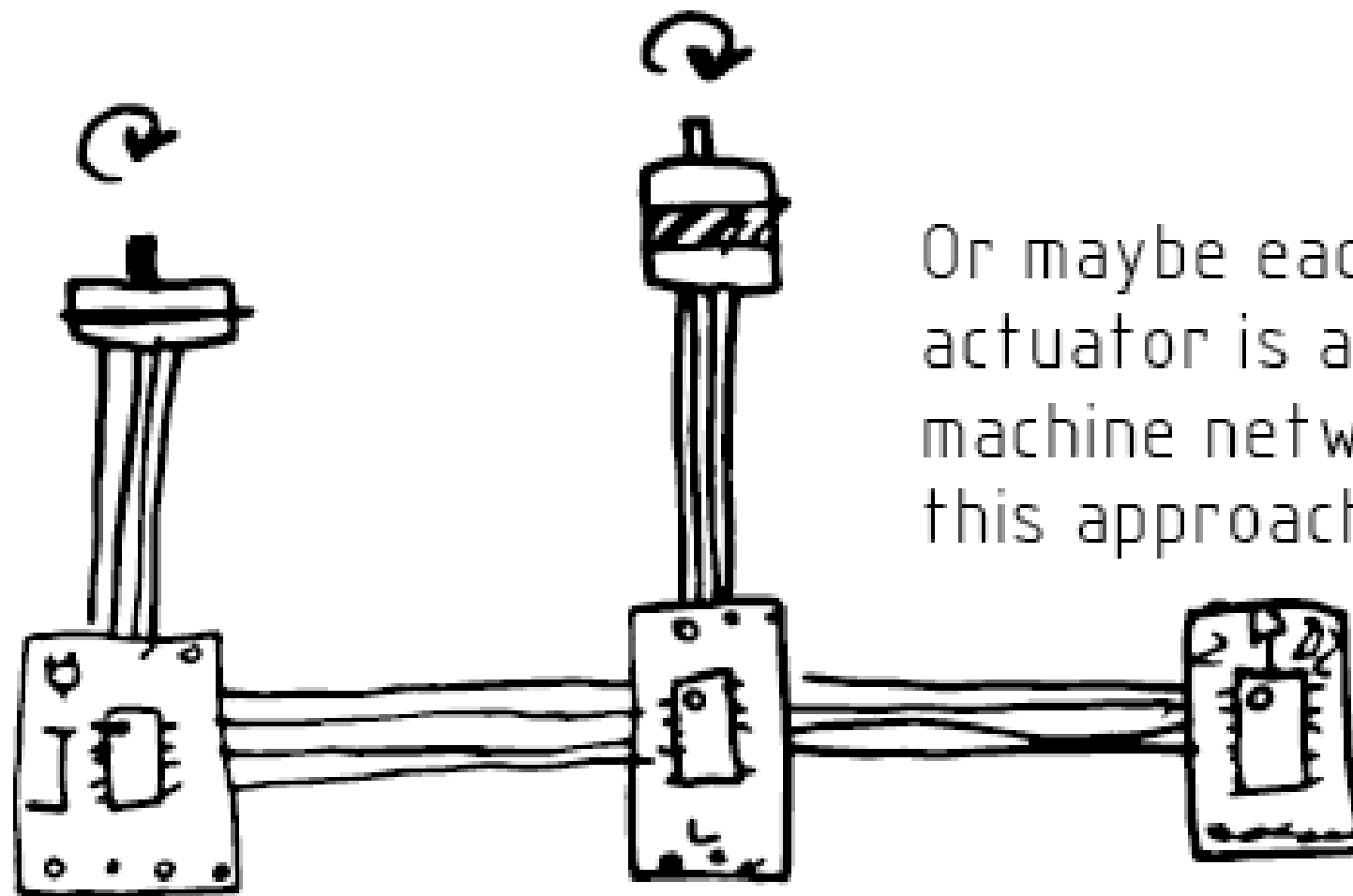
Maybe your control system is one electronic brain that interprets machine coordinates, (e.g. G-code).



Machine control box

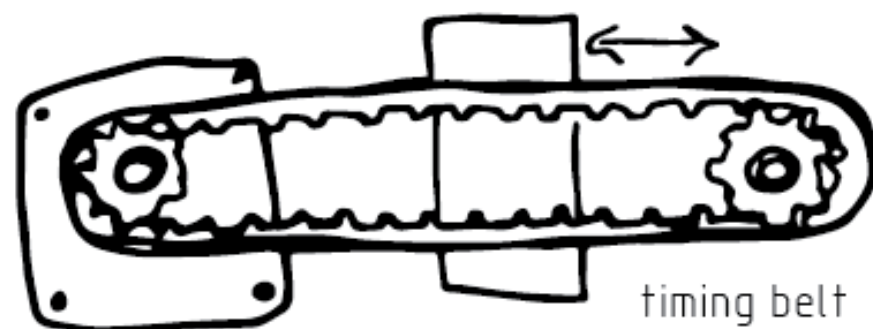
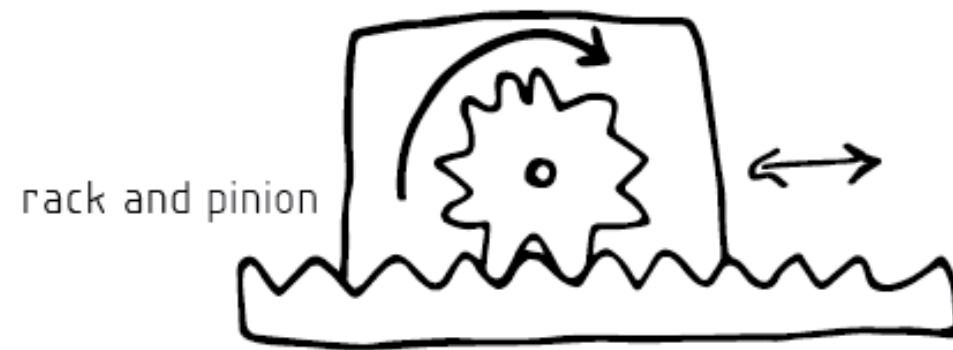
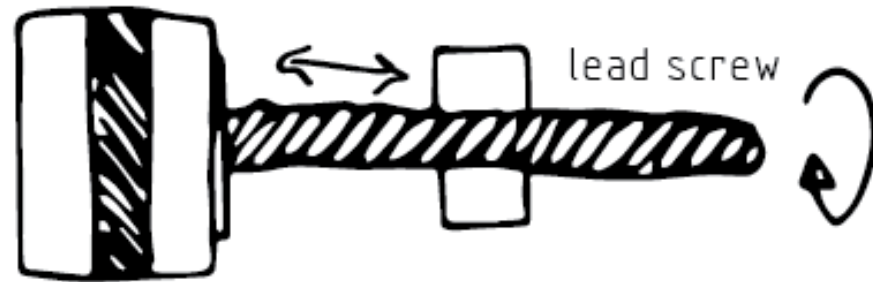


Machine control box



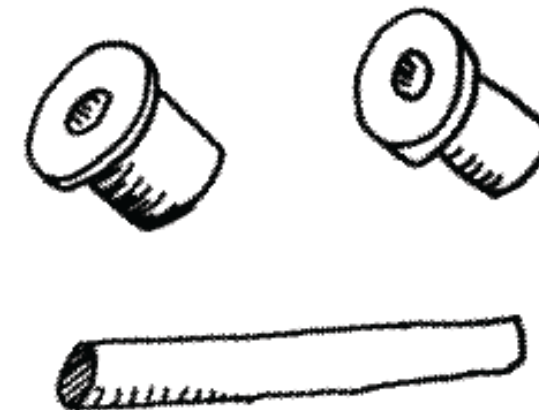
Or maybe each sensor and actuator is a node on a machine network! We like this approach.

Machine control network



Different drive trains are better at different things, like timing belts are fast, racks and pinions are stiff, lead screws are strong. Some systems are cheaper, some are easier to assemble. These are all things to take into consideration for your machine!

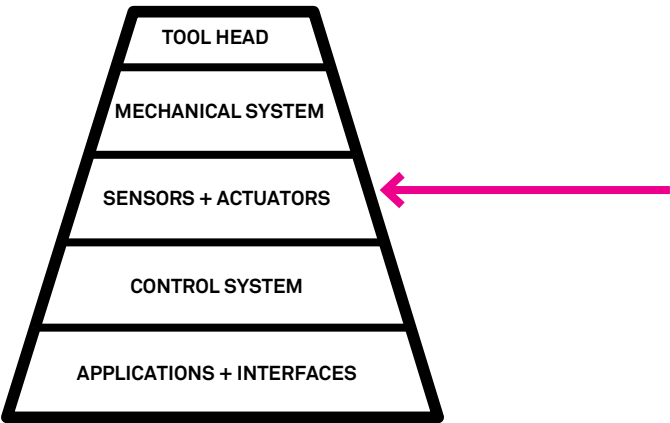
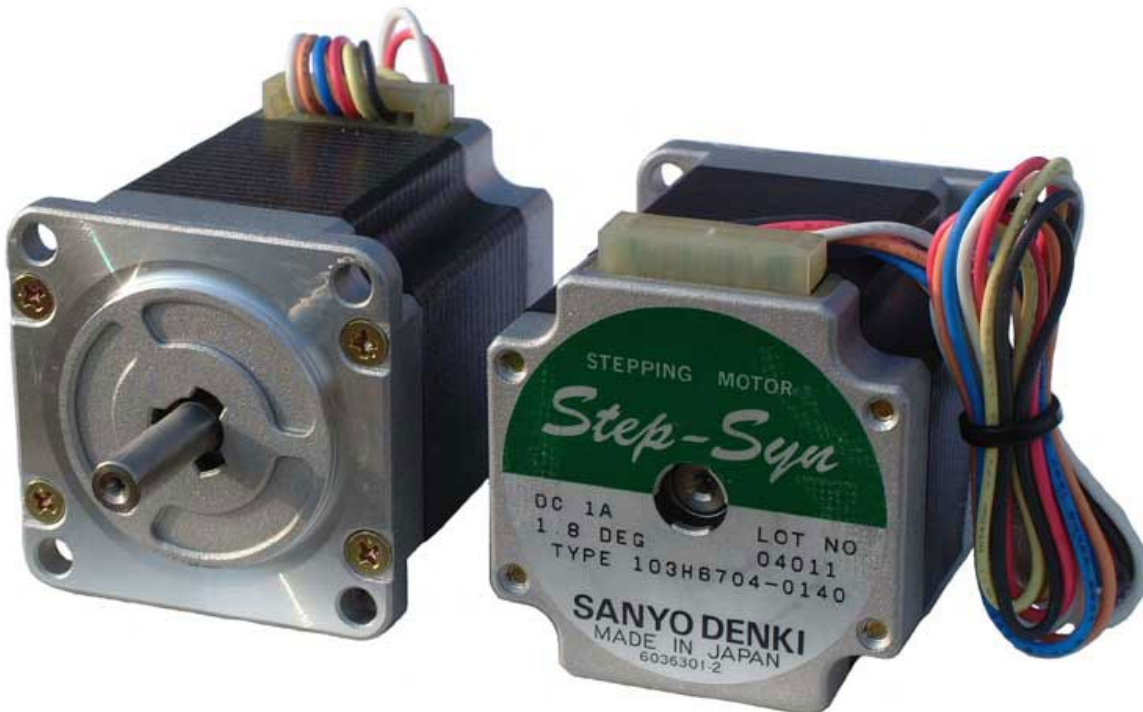
What kind of machines do you know that use these different kinds of drive trains?



The drive trains motion needs to be restricted in the axes you want to move in. You can do this with guide shafts, tracks, cable guides, linkages, and many other ways.

Moving things

HTMAA MACHINE DESIGN



Moving things

Machines that Make

What does it do?



cut?



mill?



extrude?



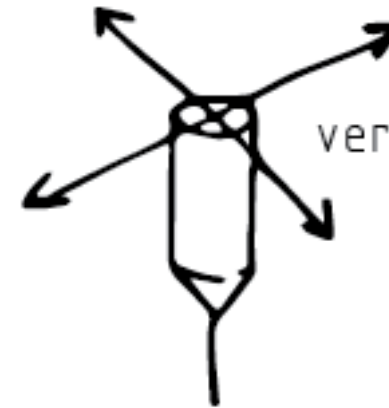
burn?



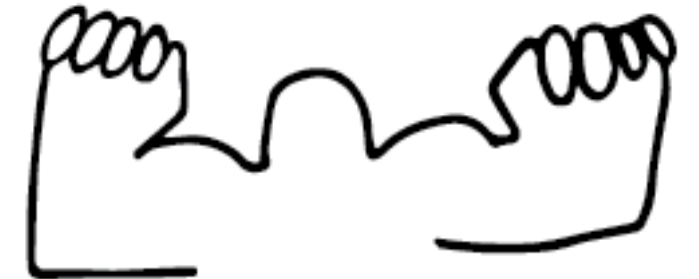
dispense
candy?



slow and
steady?



very precise?

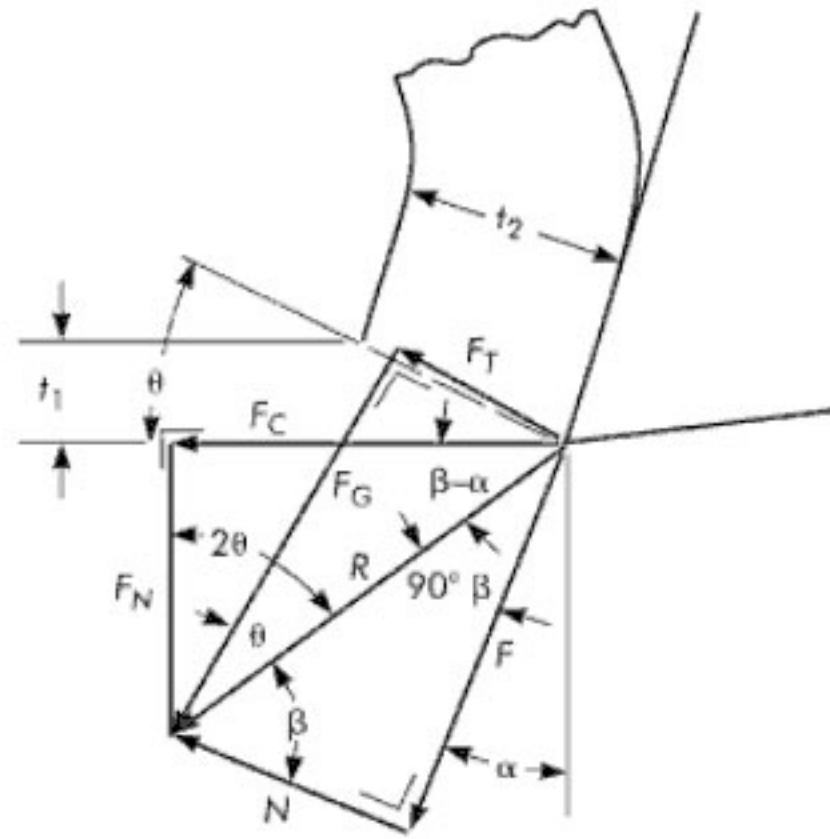


lots of force?

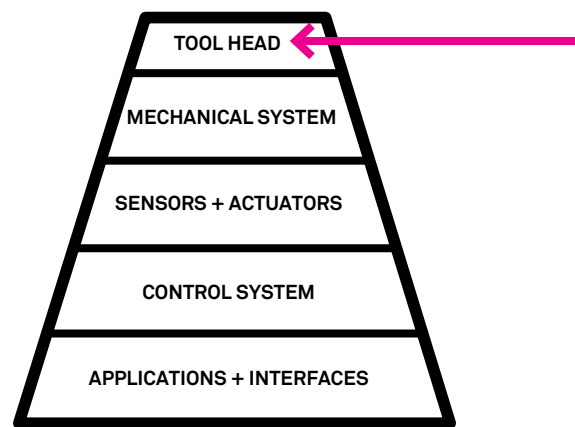
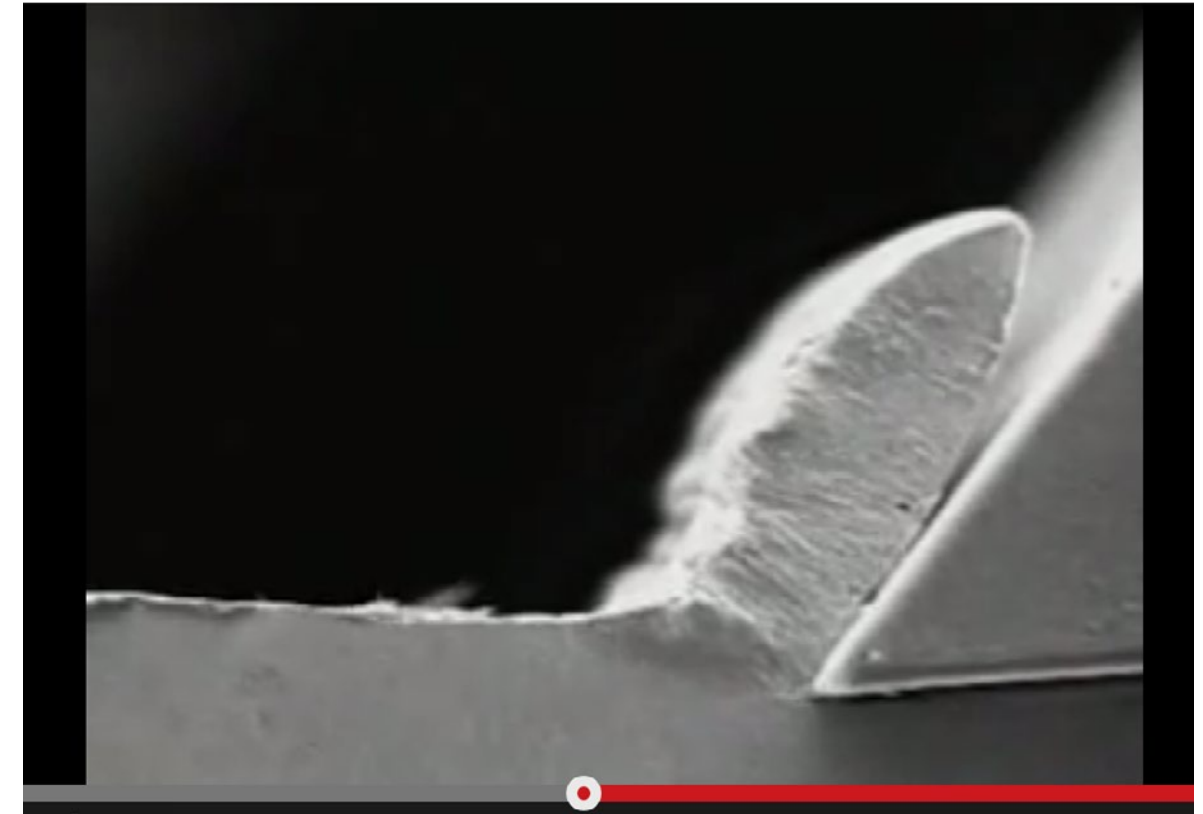
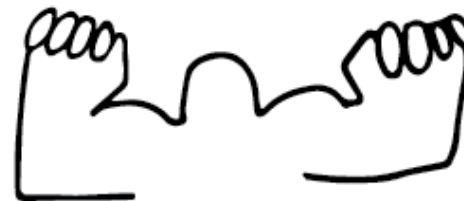
superfast?



Tool force, precision, speed



- R = resultant cutting force, lbf (kN)
- F_T = shear force, lbf (kN)
- F_G = normal to shear force, lbf (kN)
- F_C = cutting force, lbf (kN)
- F_N = normal to cutting force, lbf (kN)
- F = friction force, lbf (kN)
- N = normal to friction force, lbf (kN)
- t_1 = depth of cut or feed, in. (mm)
- t_2 = chip thickness, in. (mm)
- θ = shear angle, $^\circ$
- β = friction angle, $^\circ$
- α = rake angle, $^\circ$




Machining parameters

Linear Motion Carriages

Ball-Bearing Carriages and Guide Rails
Move high-capacity loads at high speeds with low friction.



Sleeve-Bearing Carriages and Guide Rails
An economical alternative to ball-bearing carriages, these are less likely to be hindered by dirt and debris.



Track Roller Carriages and Guide Rails
Great for light loads and applications that do not require high levels of precision.



Shaft Supports



Hardened Shafts



Miniature Hardened Shafts



Aluminum Shafts



Hardened Shafts with Machinable Ends



Hardened Shafts with Tapped Ends



Hardened Shafts with Threaded Ends



Tubular Shafts



Hardened Shafts with Support Rail



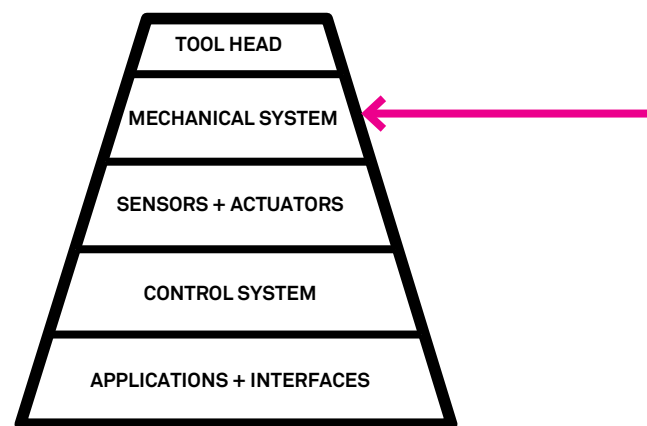
Ceramic-Coated Aluminum Shafts with Support Rail



Hardened Shafts with Tapped Mounting Holes



D-Profile Shafts



Linear Motion guides



Bushings



Thrust



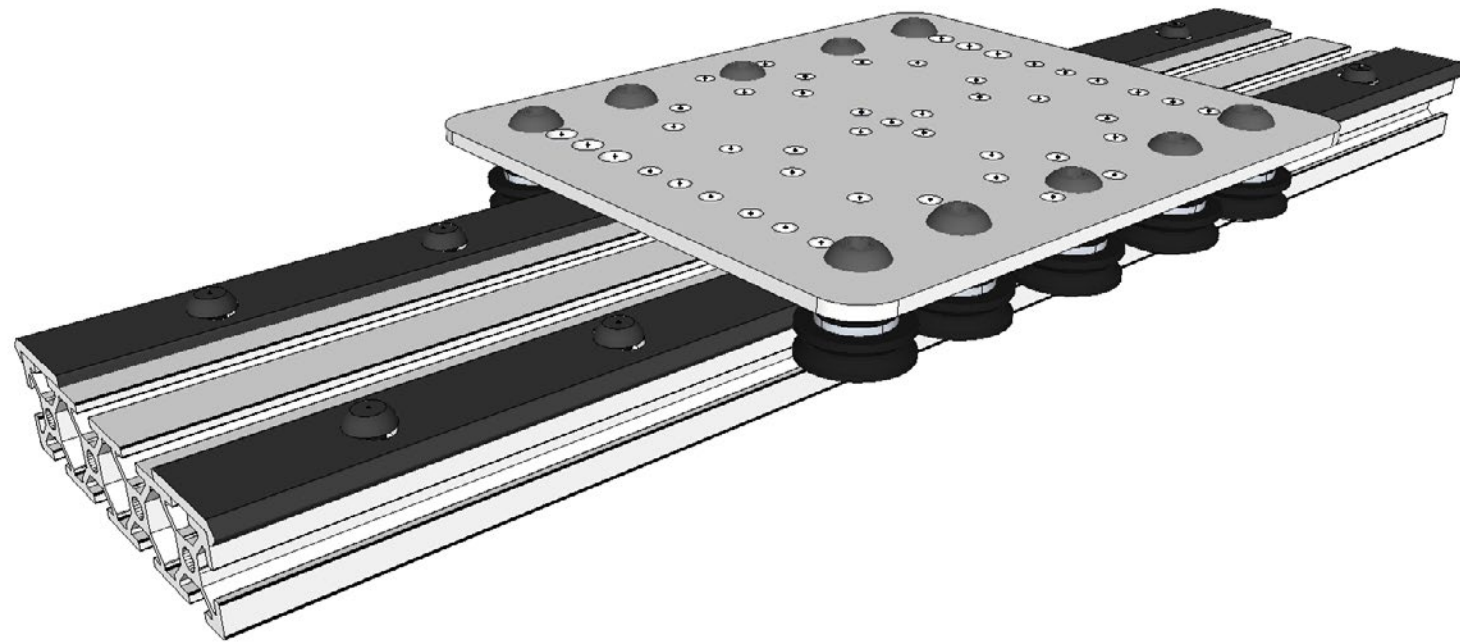
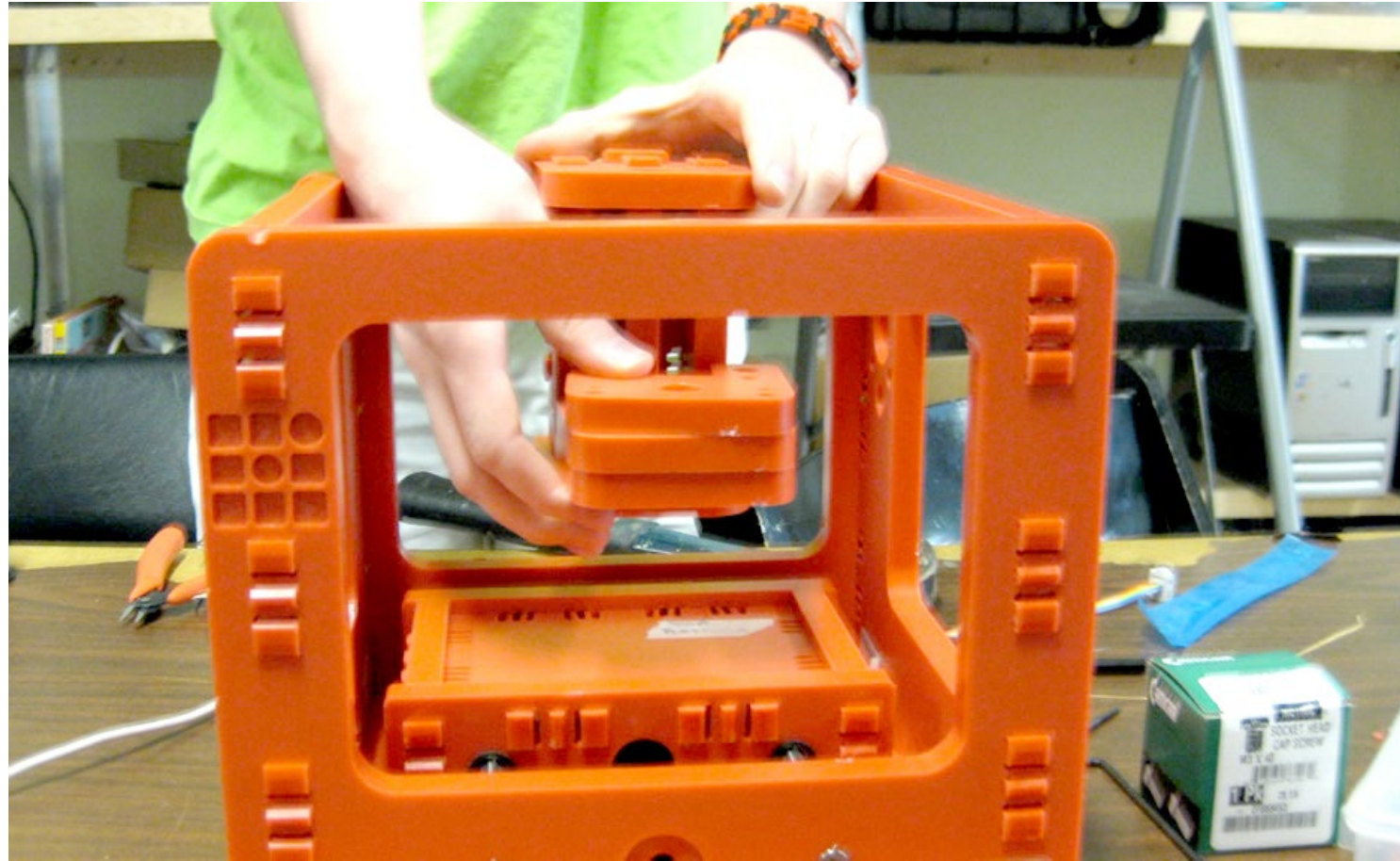
Taper



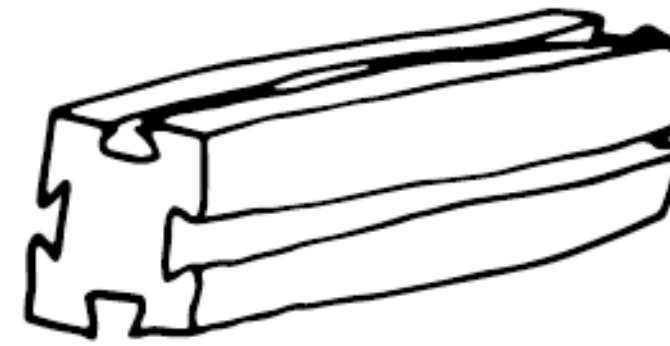
Roller/Radial

Bearings

HTMAA MACHINE DESIGN



routed plastic?
laser cut wood?



aluminum extrusion?



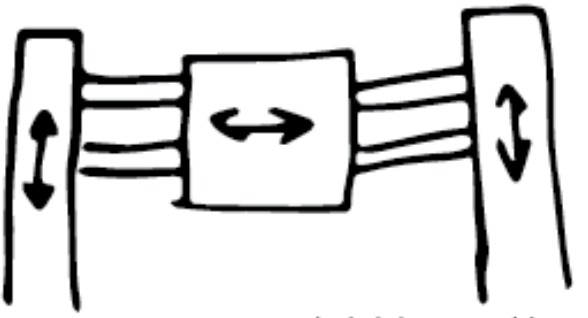
t-slot nuts?

Laser cutting is fast, but not super precise. Aluminum extrusion is expensive, and you have to assemble each piece separately. How do you hold the frame together? Glue? Machine screws? T-slot nuts? All things to consider!

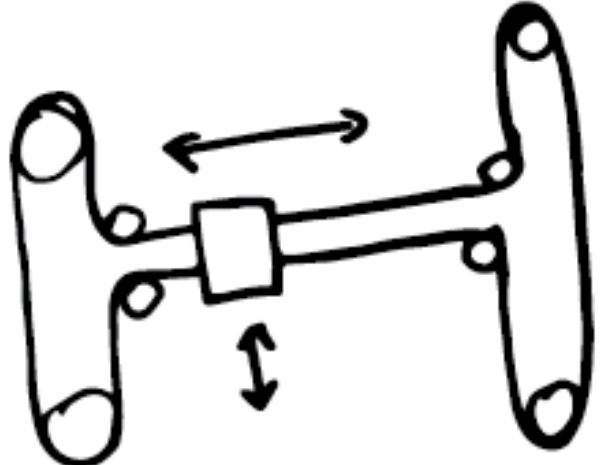
Frames

Machines that Make

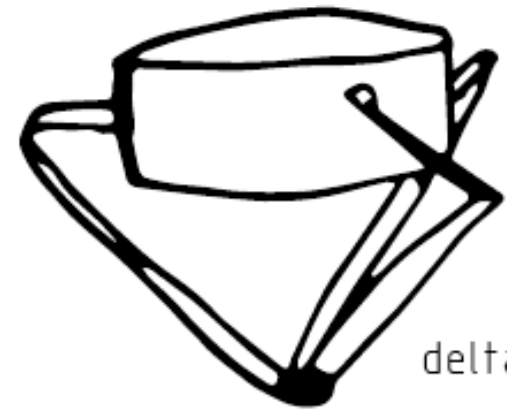




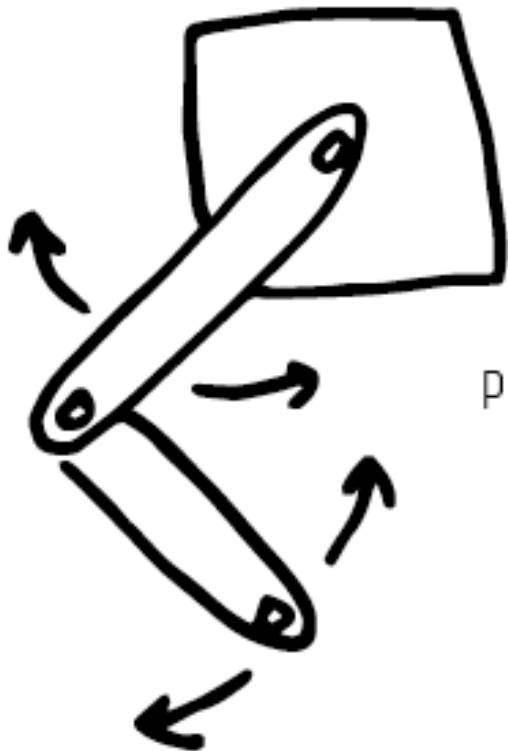
serial kinematics



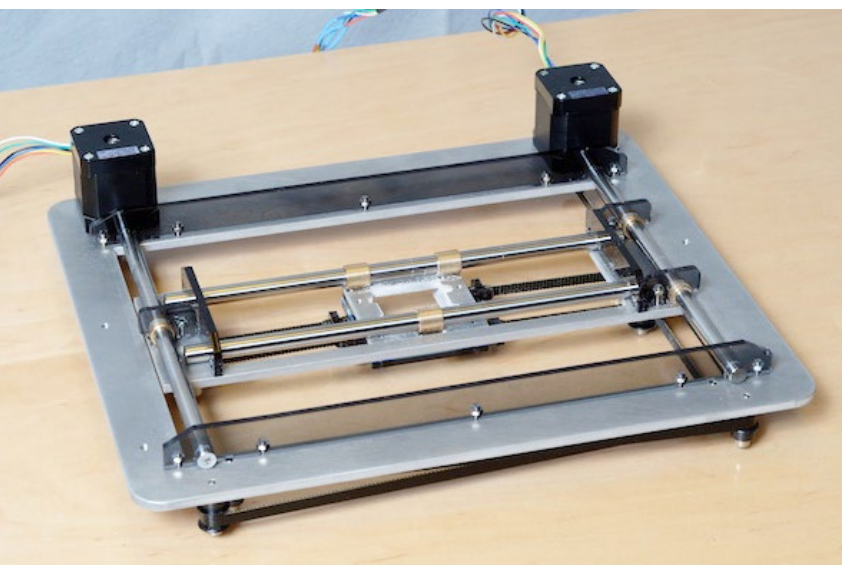
h-bot



delta bot



polar coordinates



Kinematics

HTMAA MACHINE DESIGN

ball screw support with bearing

Stepper motor

rollers

anti backlash nut

Precision guides

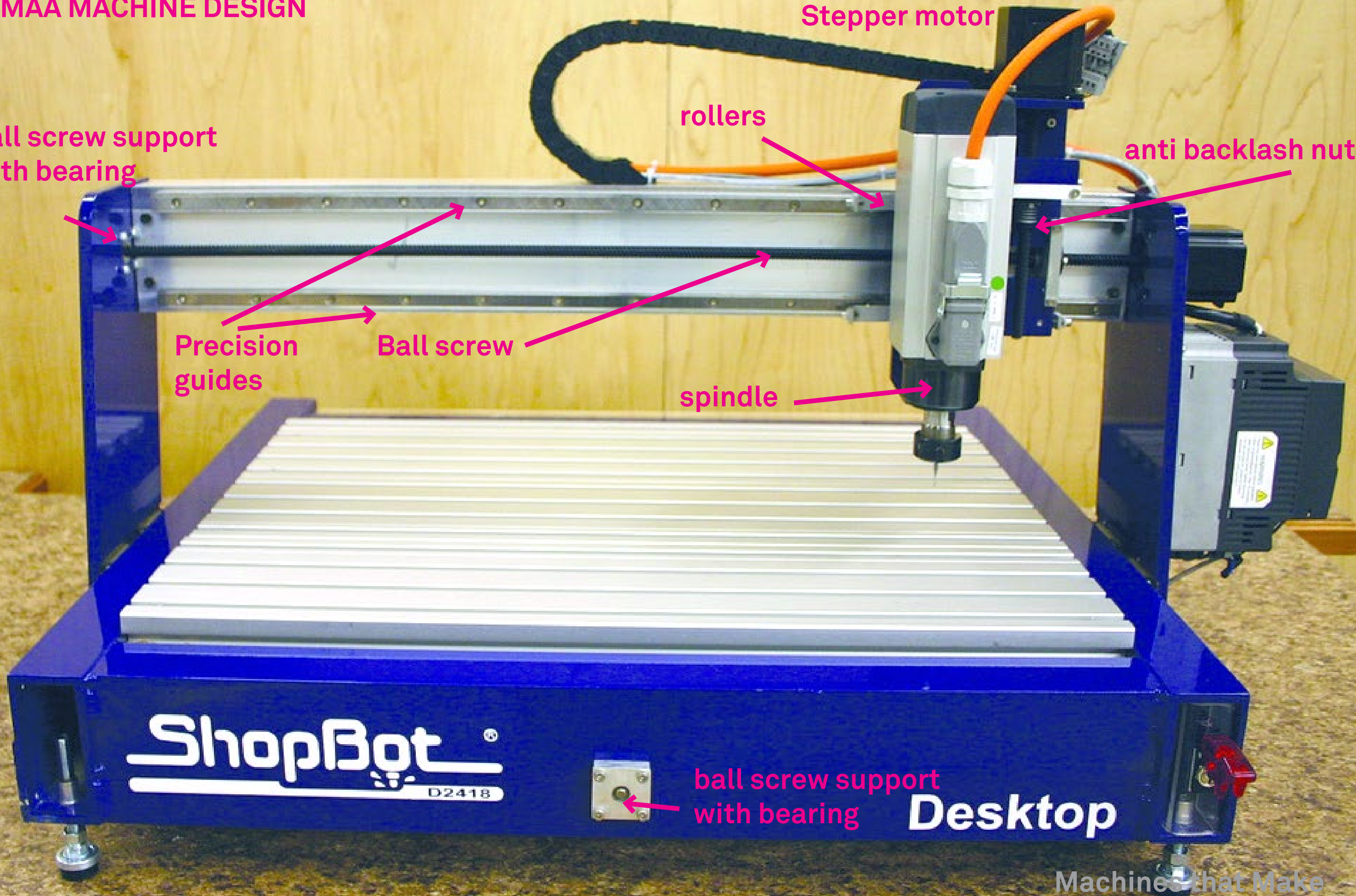
Ball screw

spindle

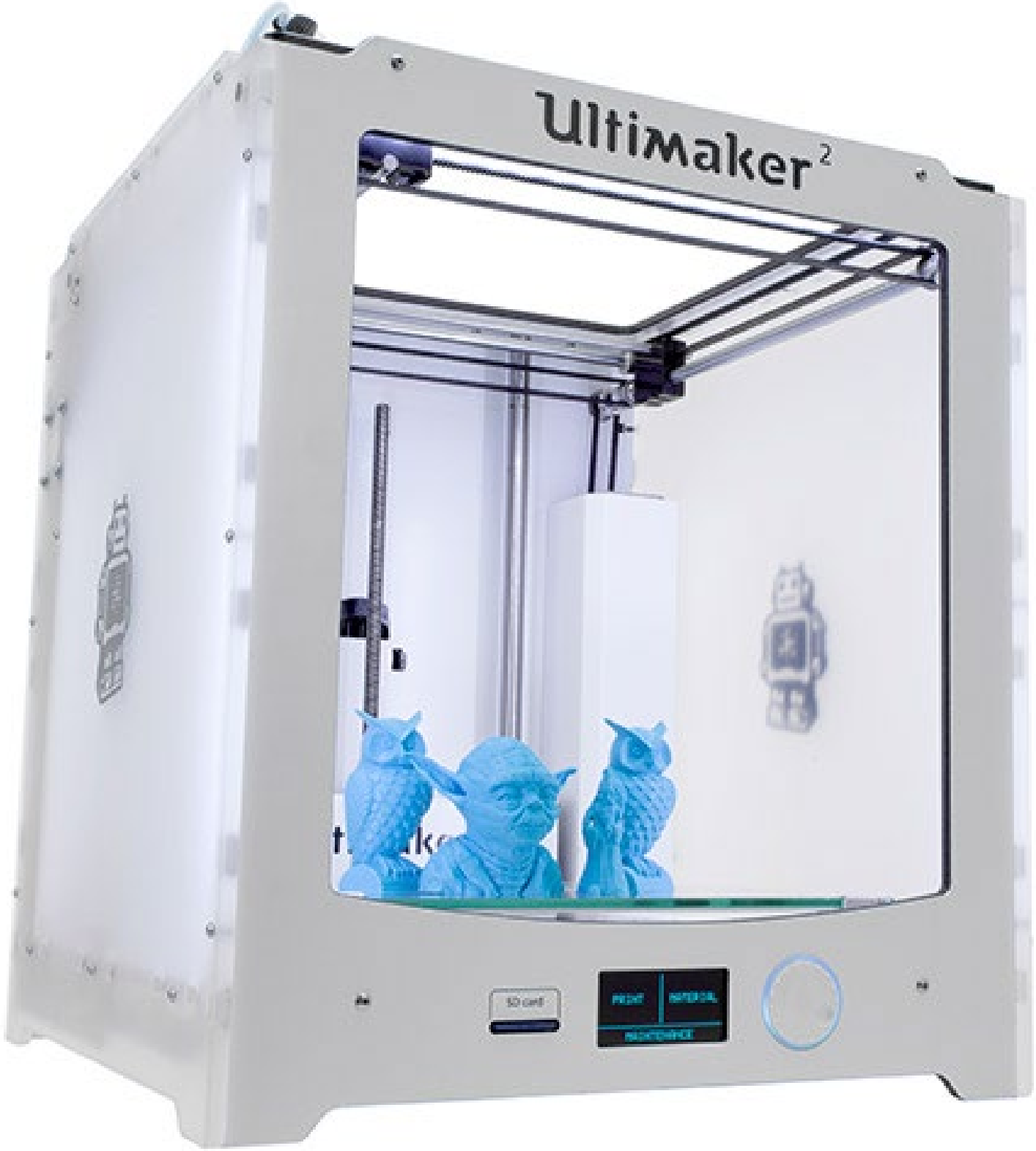
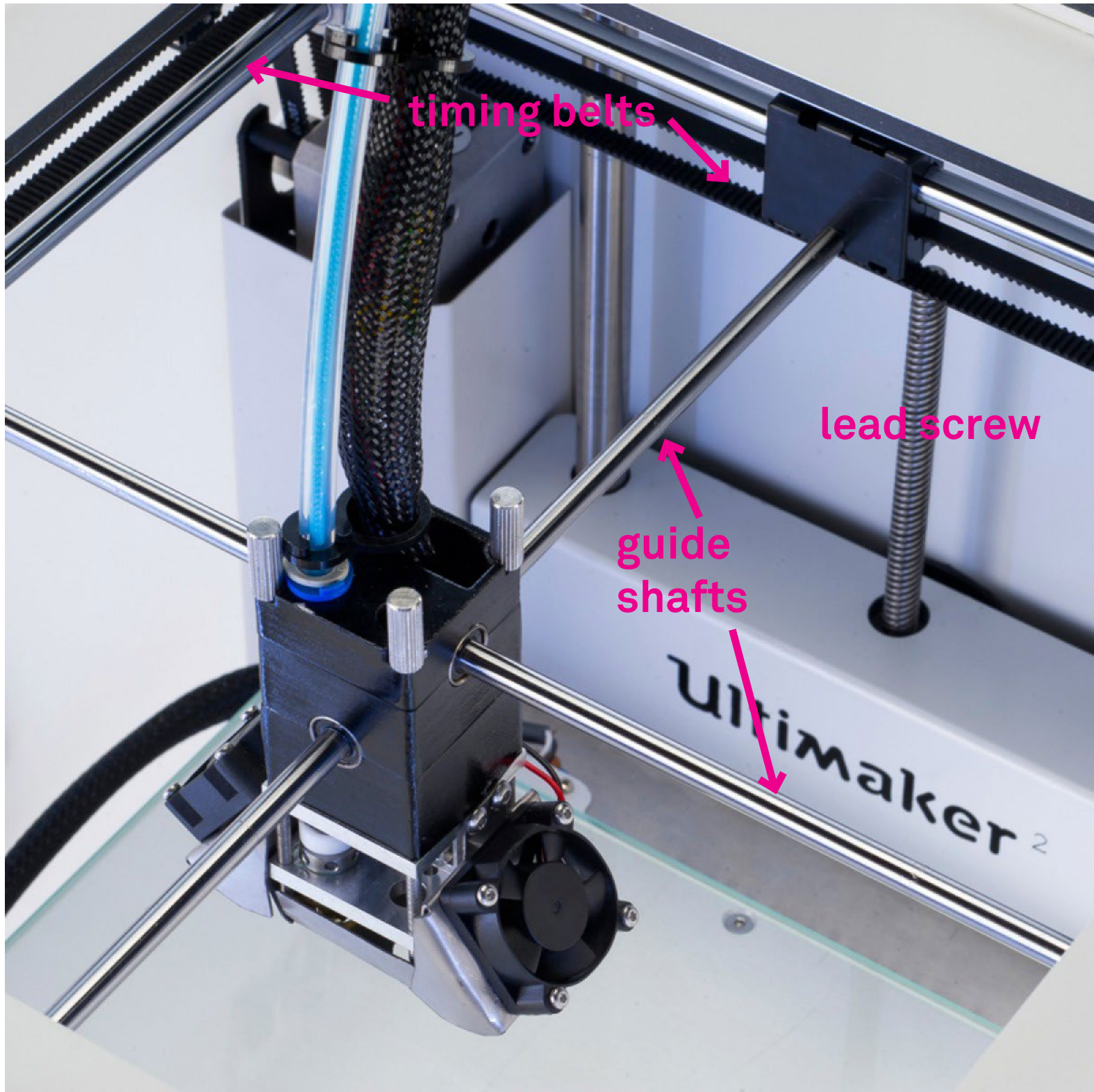
ShopBot
D2418

ball screw support with bearing

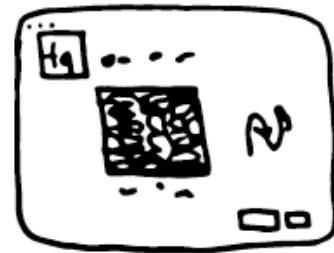
Desktop



HTMAA MACHINE DESIGN

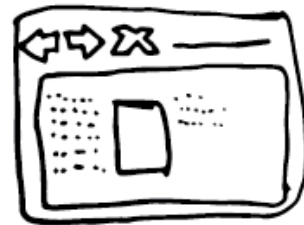


Finally, how are you going to control the machine? There are different kinds of software to stream machine code to machines with, how do you want yours to work?



Maybe a drawing program interfaces directly to the machine

Maybe the machine is controlled from a browser...



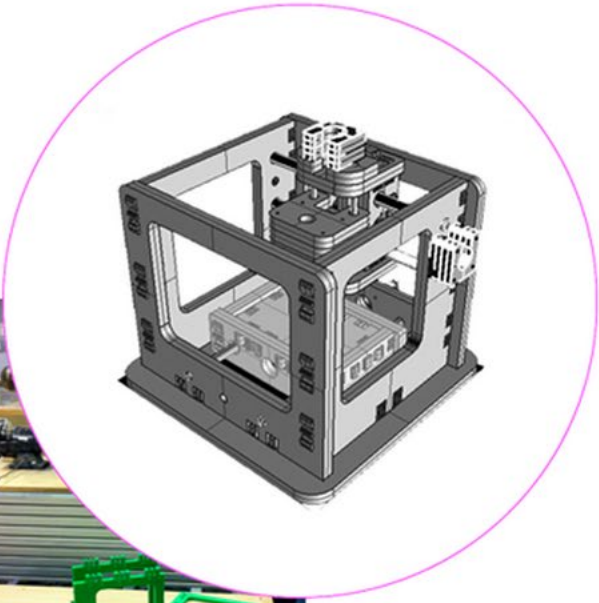
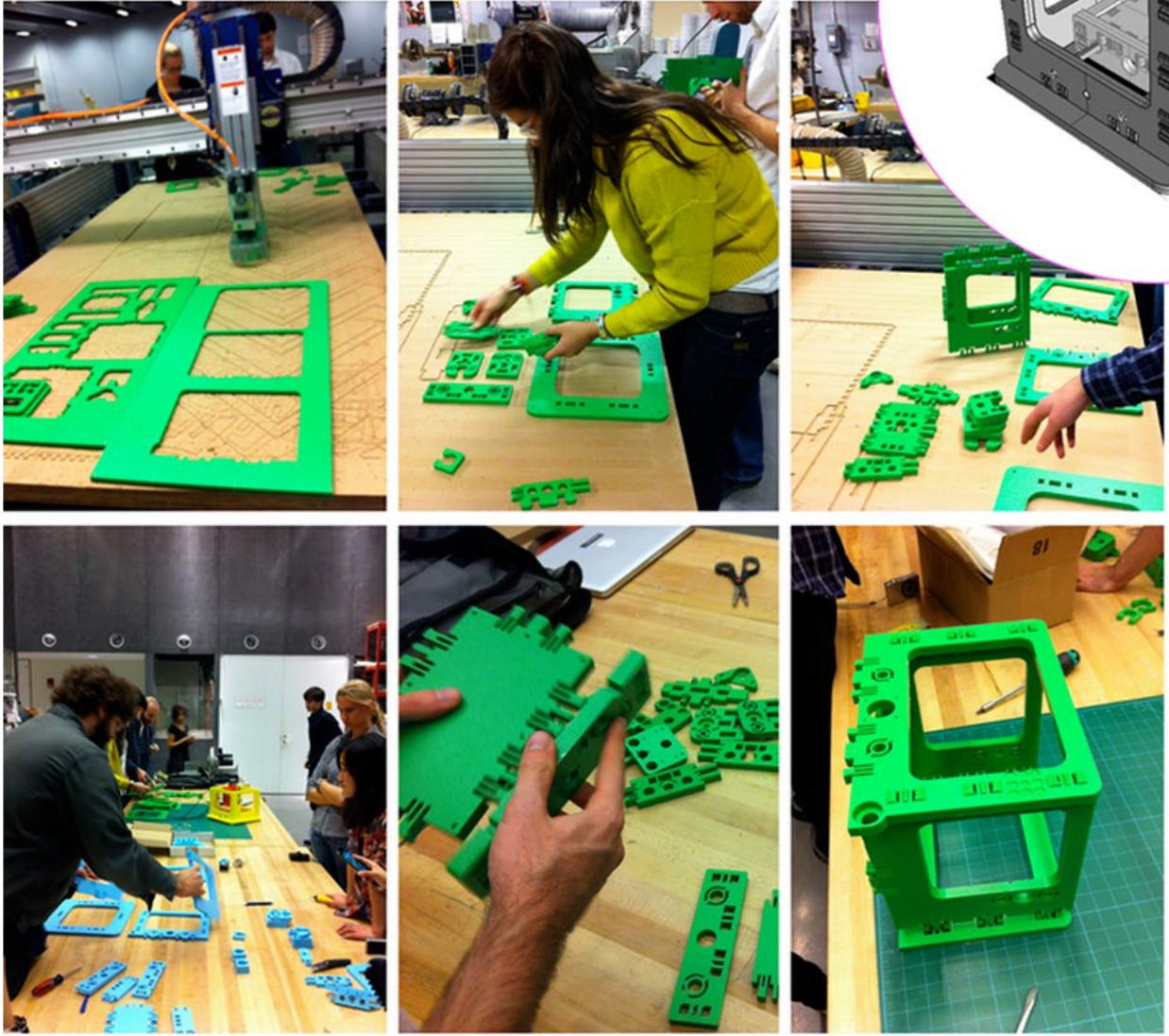
Those were some tips for making your own machines, we hope they were helpful. Now go on, cook up some machines, and have fun!

I made my machine,
it is just right for me

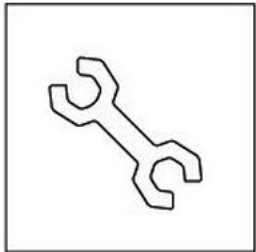


**Each section must
MAKE A MACHINE!!!**

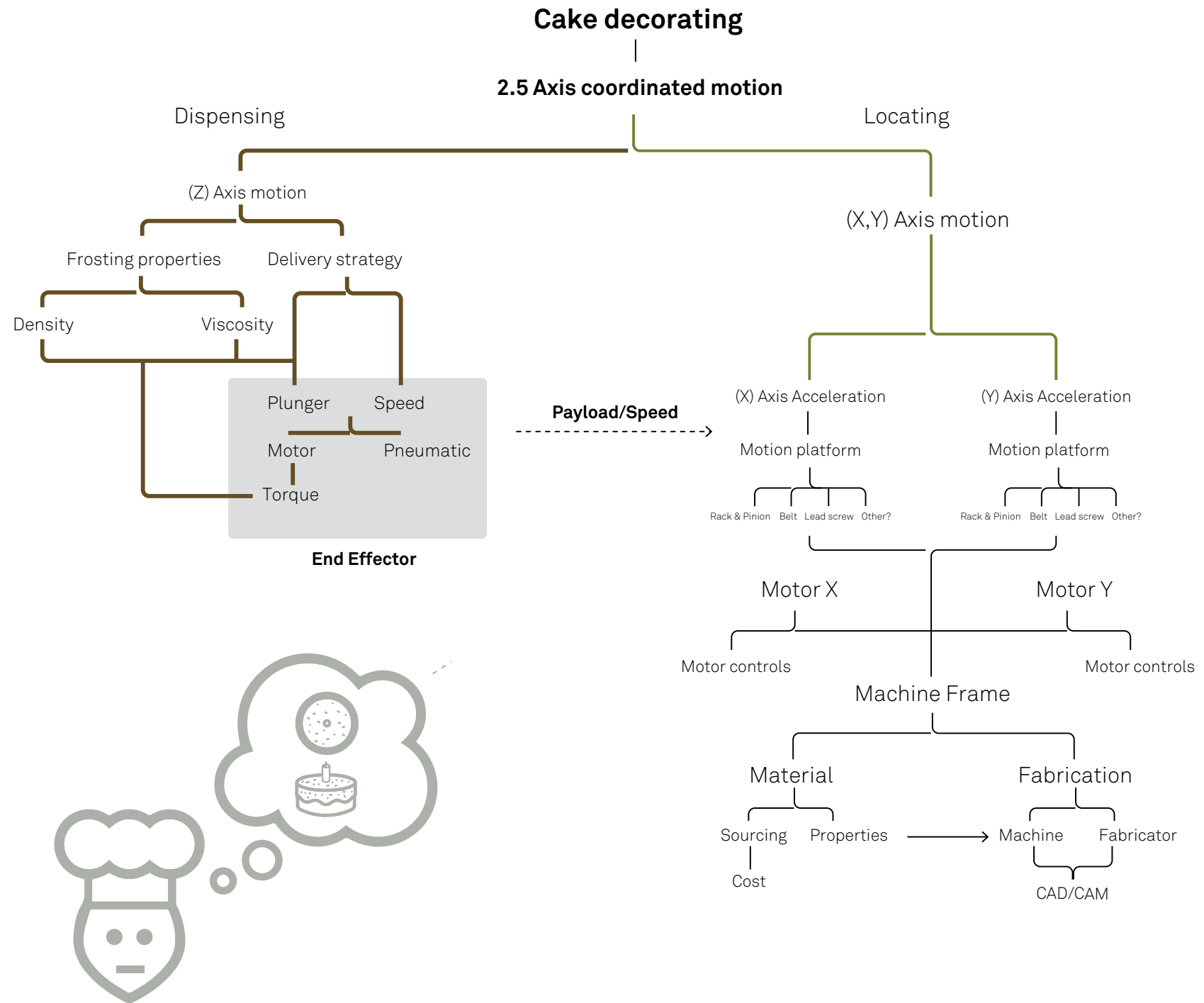
The Assignment



I spent most of my time preparing for the final project. [Click on the icon below to view my action plan.](#)

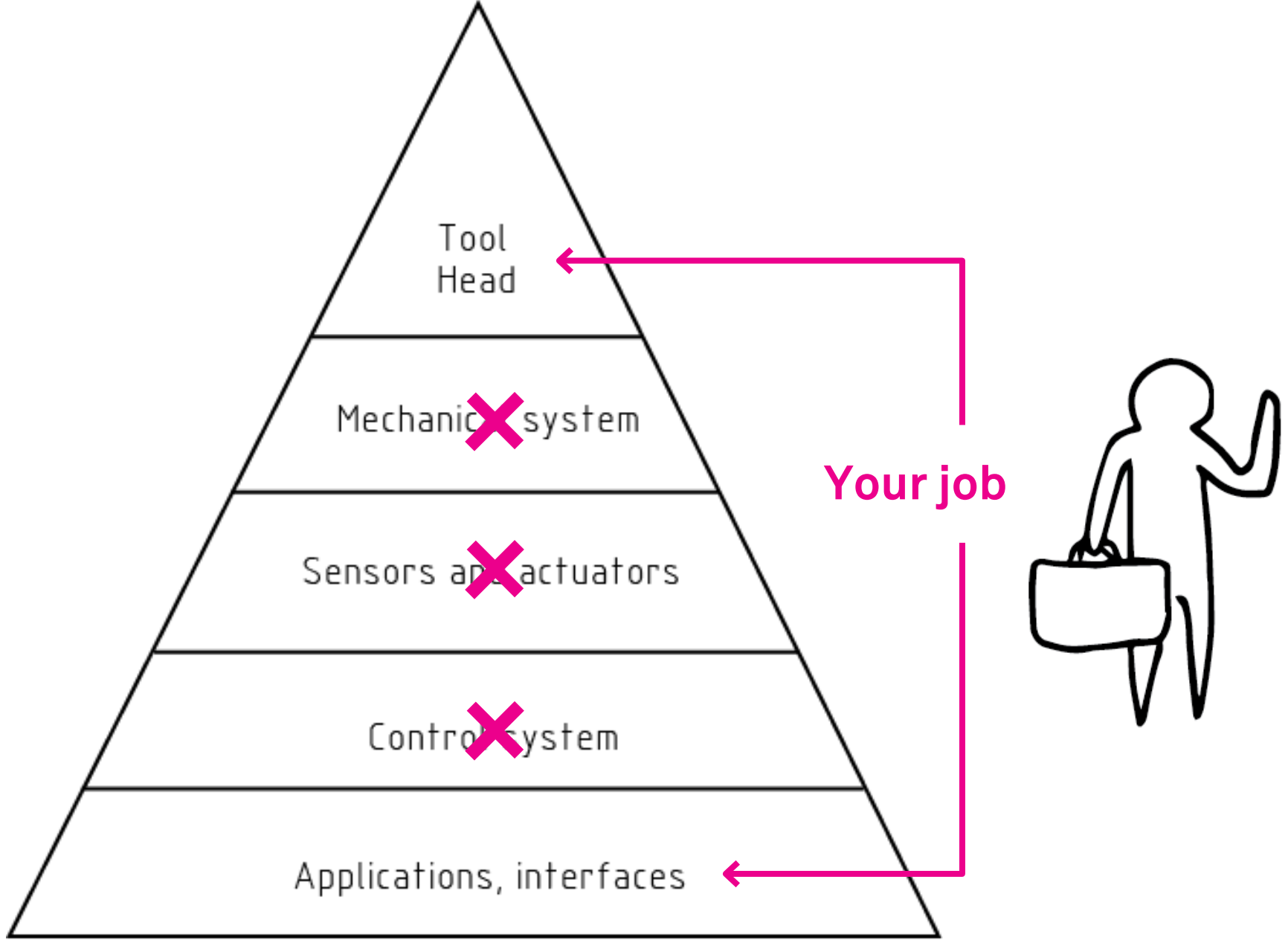


The trouble

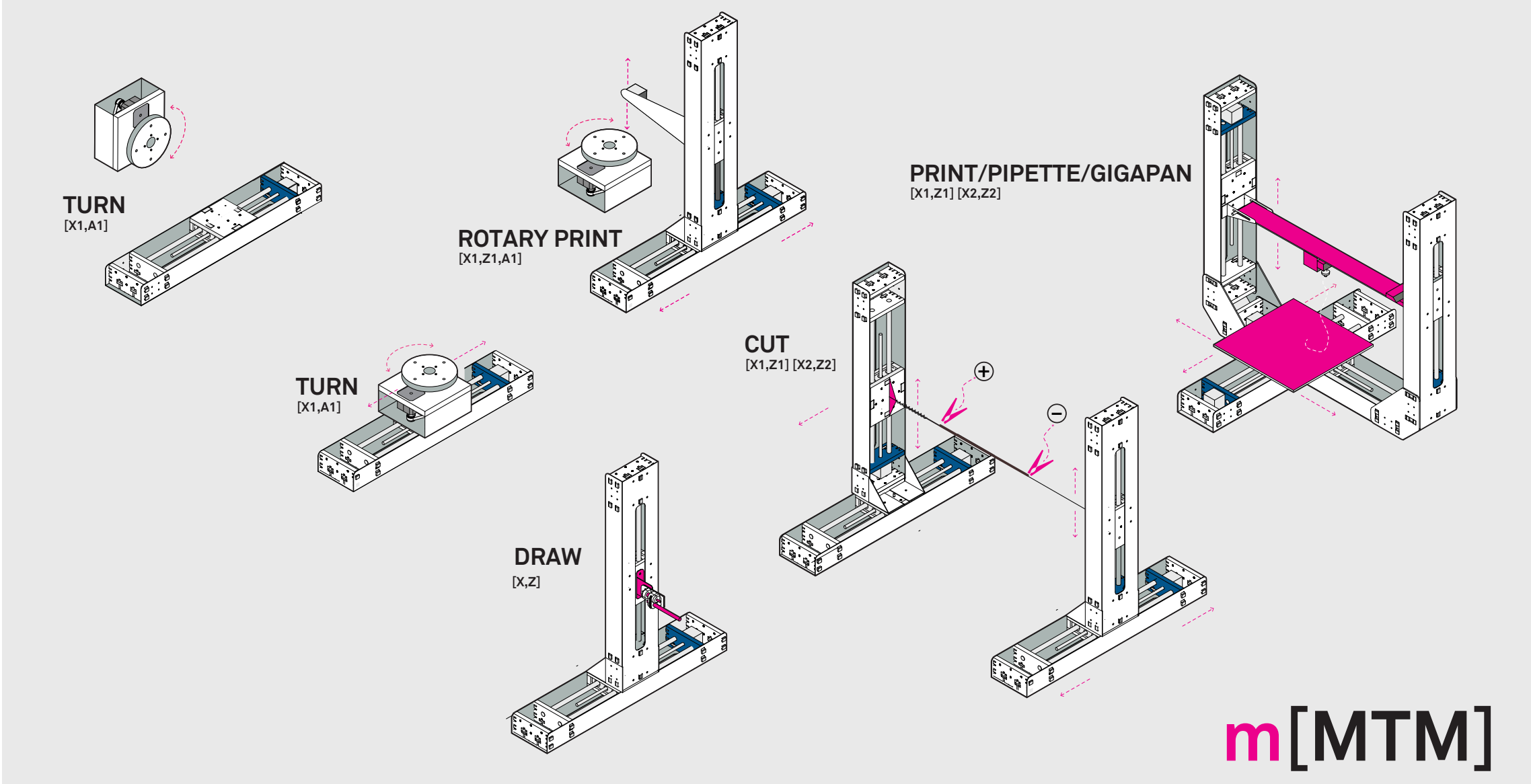


The strategy

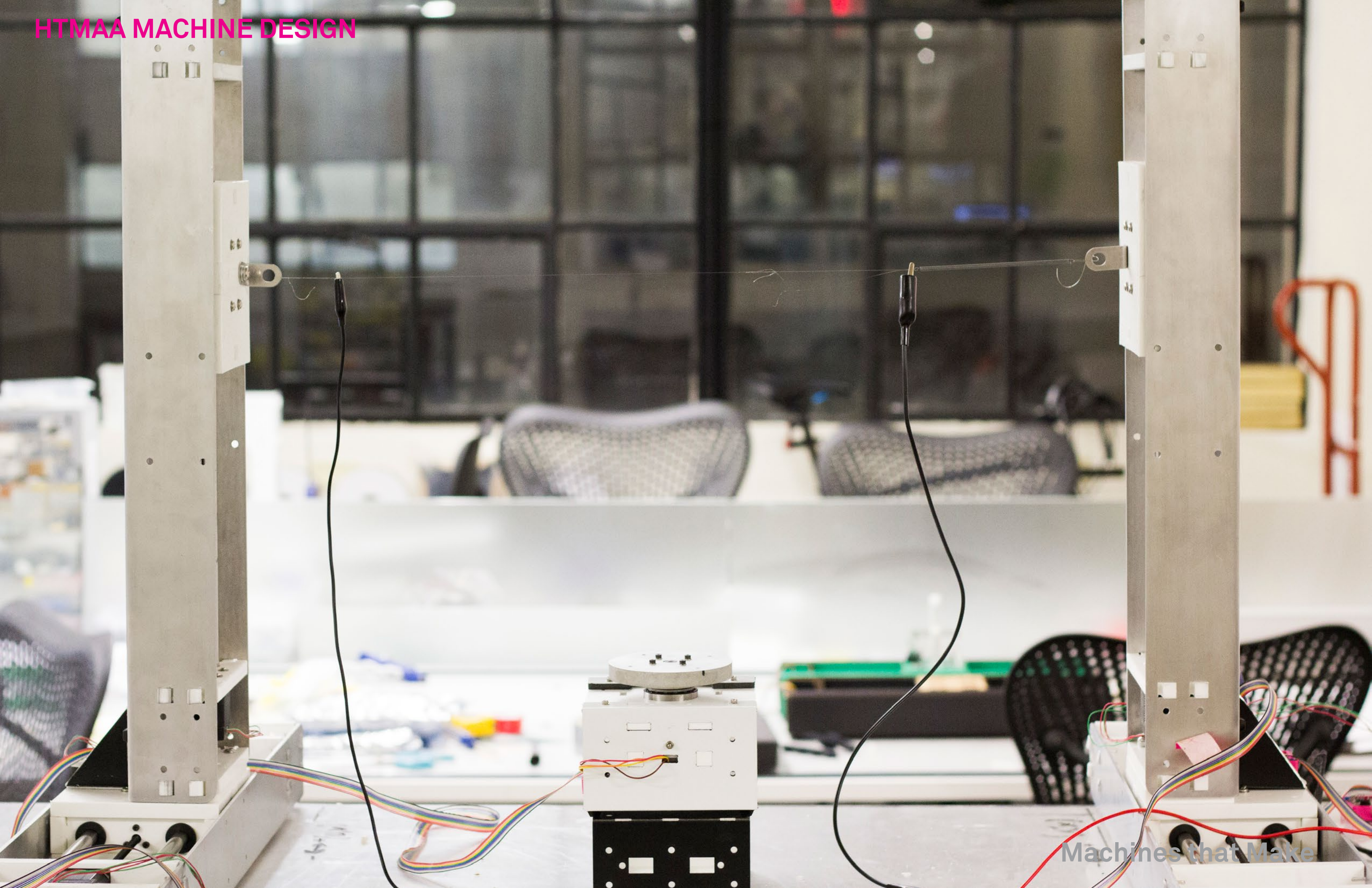
HTMAA MACHINE DESIGN

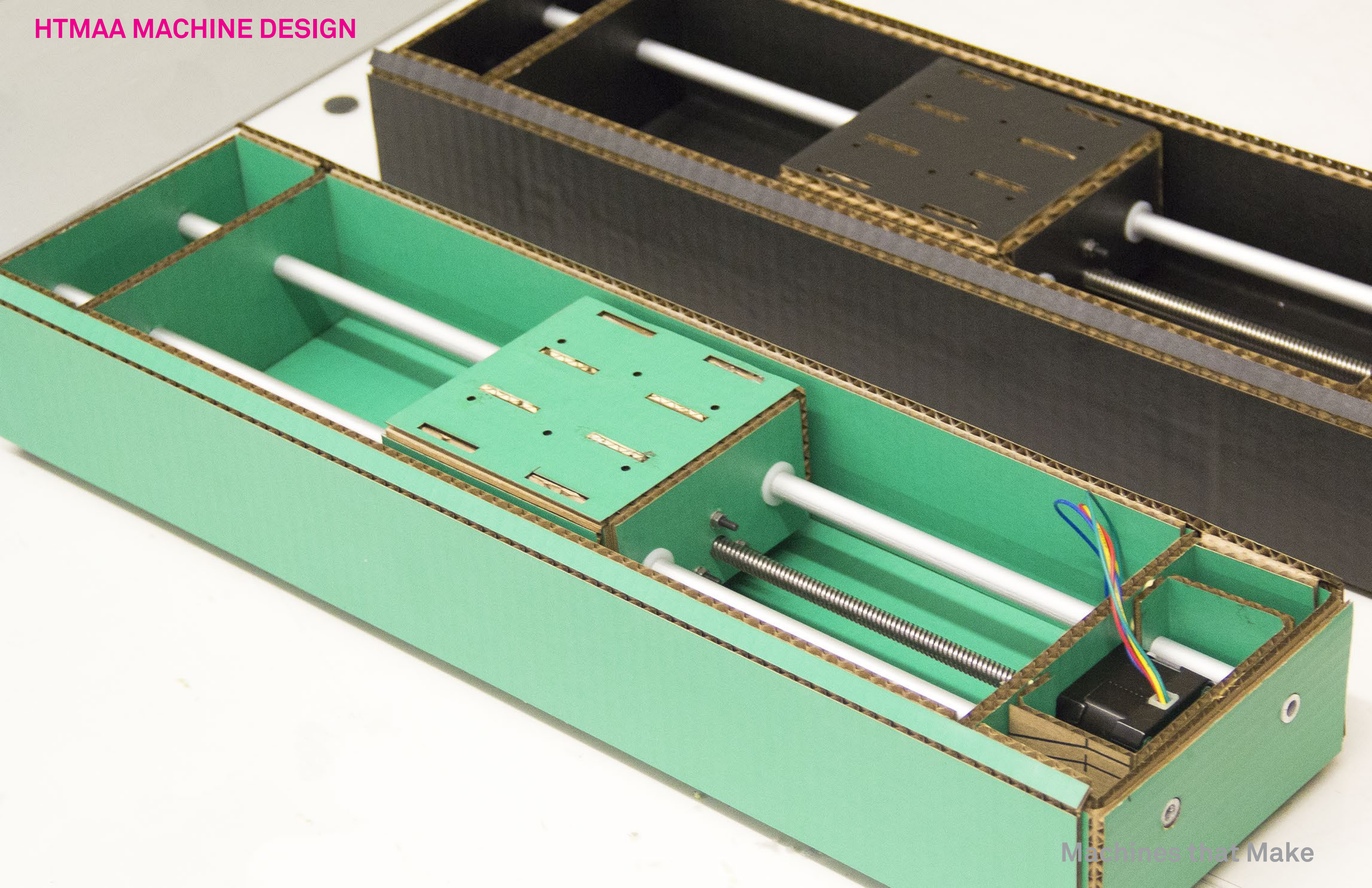


making machines that make

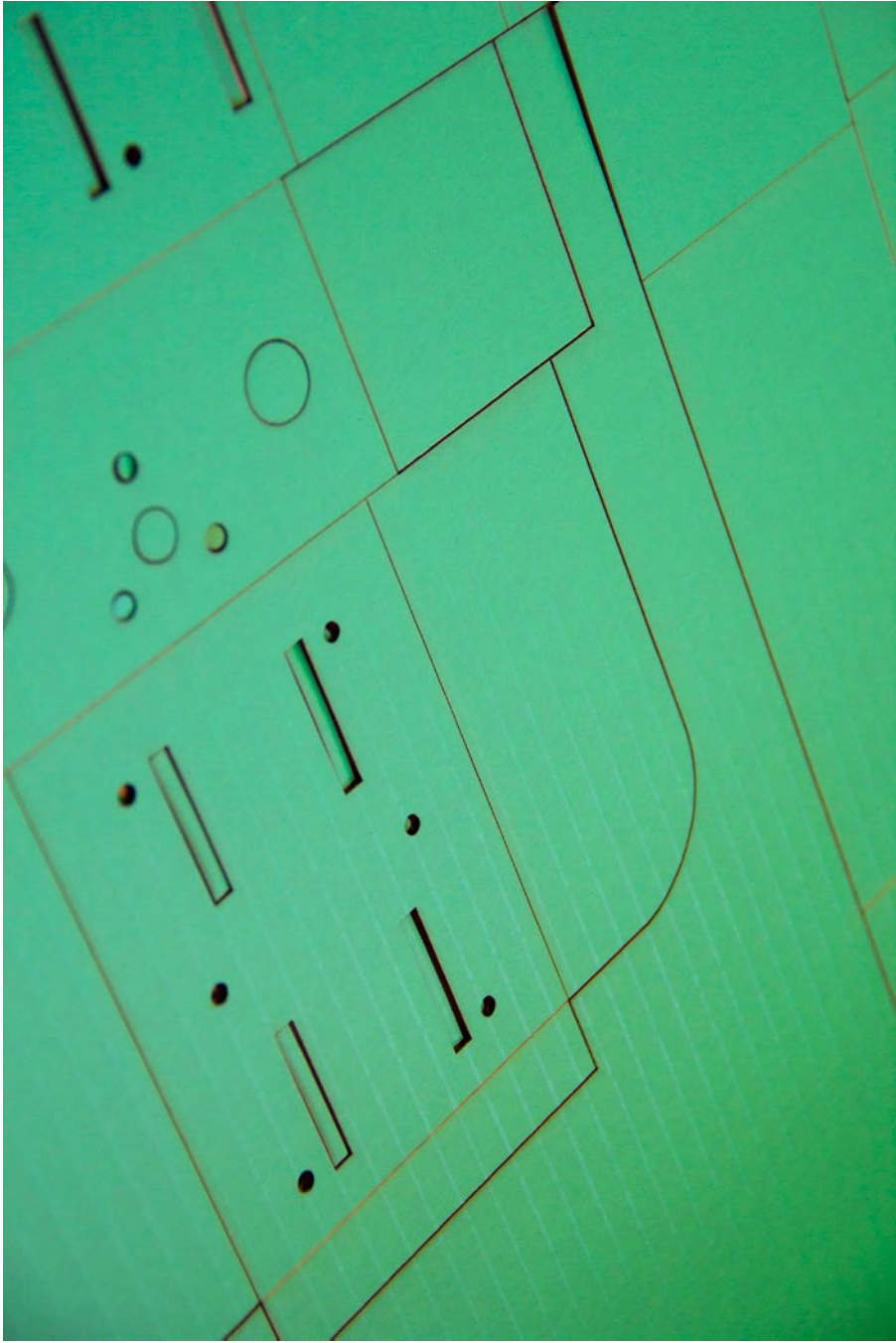


[modular] Machines that Make

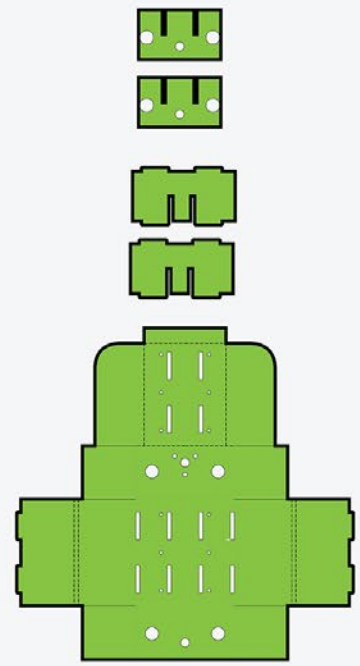
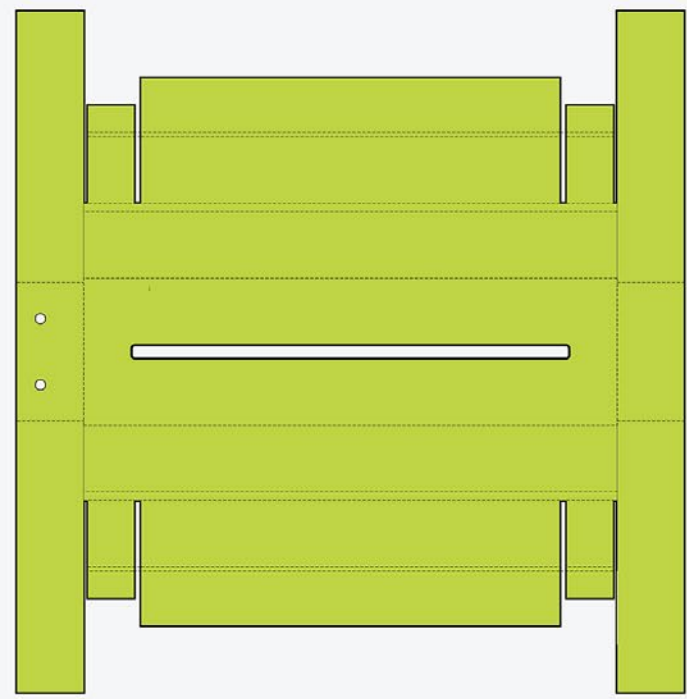
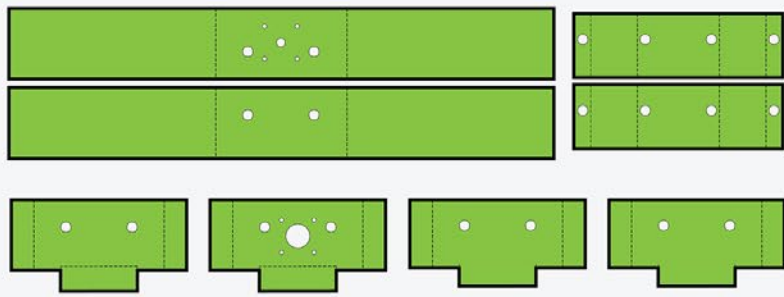




HTMAA MACHINE DESIGN



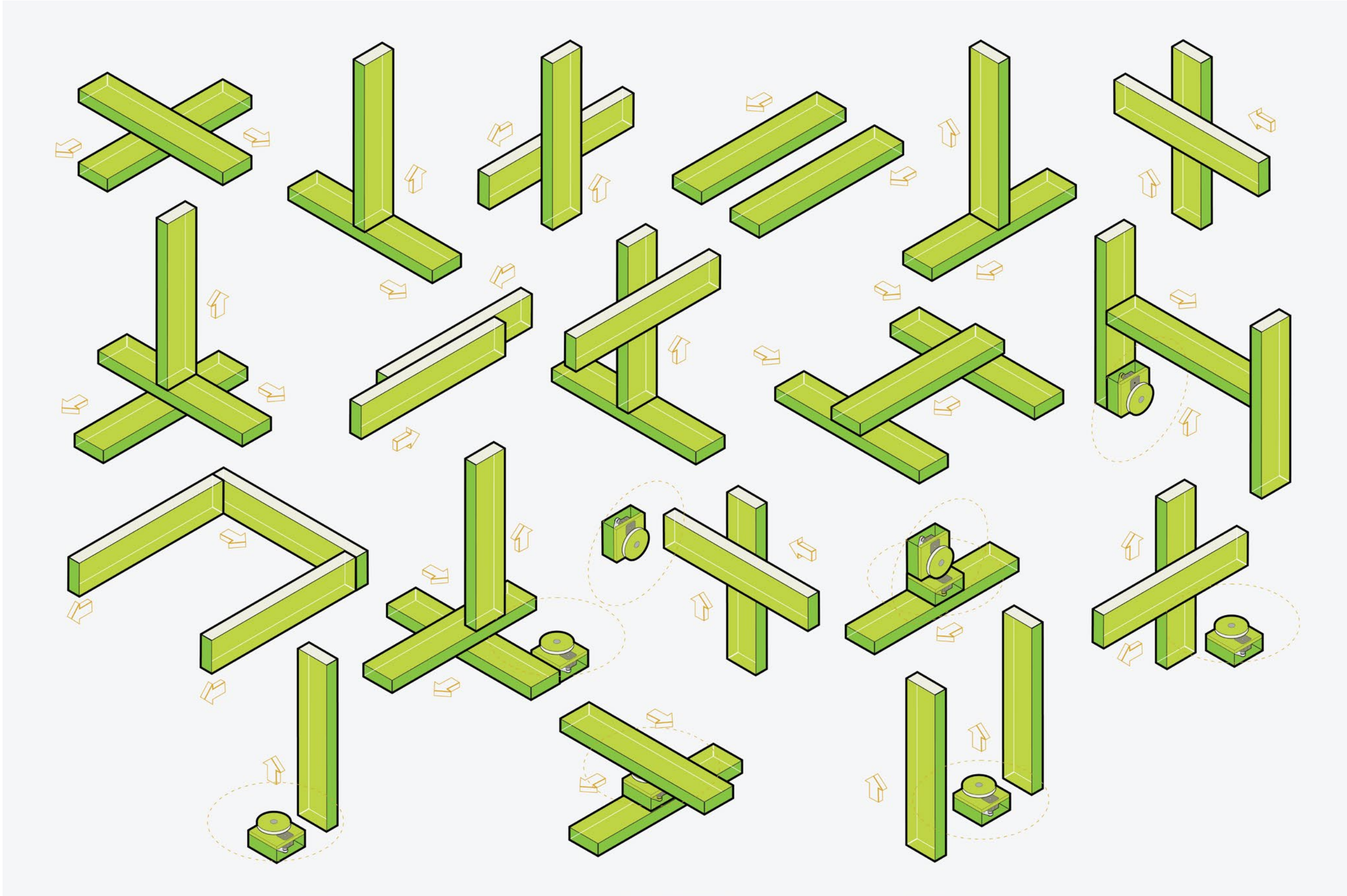
CARDBOARD PART LAYOUT



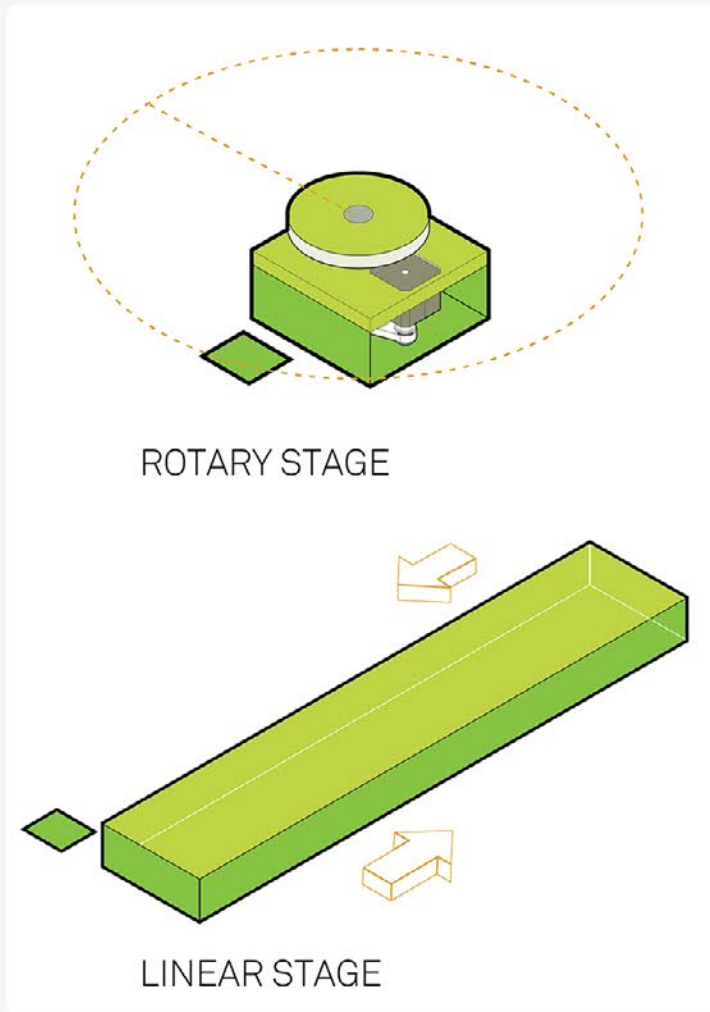
FRAME PARTS

STAGE PARTS

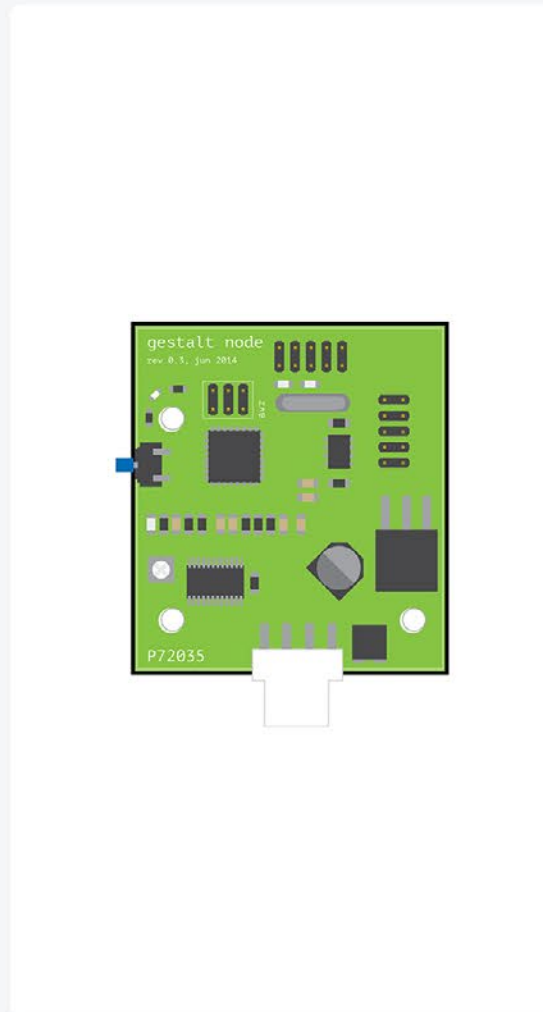
HTMAA MACHINE DESIGN



MODULAR MACHINE PARTS



GESTALT NODE



GESTALT FRAMEWORK

```
# Forked from DFUnivM Oct 2013
# set portname
# set location of hex file for bootloader
#
#---IMPORTS---
from gestalt import nodes
from gestalt import interfaces
from gestalt import machines
from gestalt import functions
from gestalt.machines import elements
from gestalt.machines import kinematics
from gestalt.machines import state
from gestalt.utilities import notice
from gestalt.publish import rpe #remote procedure call dispatcher
import time
import io

#---VIRTUAL MACHINE---
class virtualMachine(machines.virtualMachine):
    def __init__(self):
        if self.providedInterface: self.fabnet = self.providedInterface #providedInterface is defined in the virtualMachine class.
        else: self.fabnet = interfaces.gestaltInterface(FABNET, interfaces.serialInterface(baudRate = 115200, interfaceType = 'Tty', portName = "/dev/tty.usbserial-FYXPKBVB"))

    def __init__(self):
        self.xAxisNode = nodes.networkedGestaltNode('X Axis', self.fabnet, filename = '086-005a.py', persistence = self.persistence)
        self.yAxisNode = nodes.networkedGestaltNode('Y Axis', self.fabnet, filename = '086-005a.py', persistence = self.persistence)
        self.xyNode = nodes.compoundNode(self.xAxisNode, self.yAxisNode)

    def __init__(self):
        self.position = state.coordinate('mm', 'mm')

    def __init__(self):
        self.xAxis = elements.elementChain.forward([elements.micorstep.forward(4), elements.stepper.forward(1.8), elements.leadscrew.forward(6.096), elements.invert.forward(True)])
        self.yAxis = elements.elementChain.forward([elements.micorstep.forward(4), elements.stepper.forward(1.8), elements.leadscrew.forward(6.096), elements.invert.forward(False)])
        self.stageKinematics = kinematics.direct2 #direct drive on all axes

    def __init__(self):
        self.move = functions.move(virtualMachine = self, virtualNode = self.xyNode, axes = (self.xAxis, self.yAxis), kinematics = self.stageKinematics, machinePosition = self.position.planner = 'null')
        self.log = functions.log(self.move) #an incremental wrapper for the move function
        pass

    def __init__(self):
        self.machineControl.setMotorCurrents(aCurrent = 0.8, bCurrent = 0.8, cCurrent = 0.8)
        self.xyNode.setVelocityRequest(0) #clear velocity on nodes. Eventually this will be put in the motion planner on initialization to match state.
        pass

    def __init__(self):
        self.publisher.addNodes(self.machineControl)
        pass

    def __init__(self):
        return (position: self.position.future())

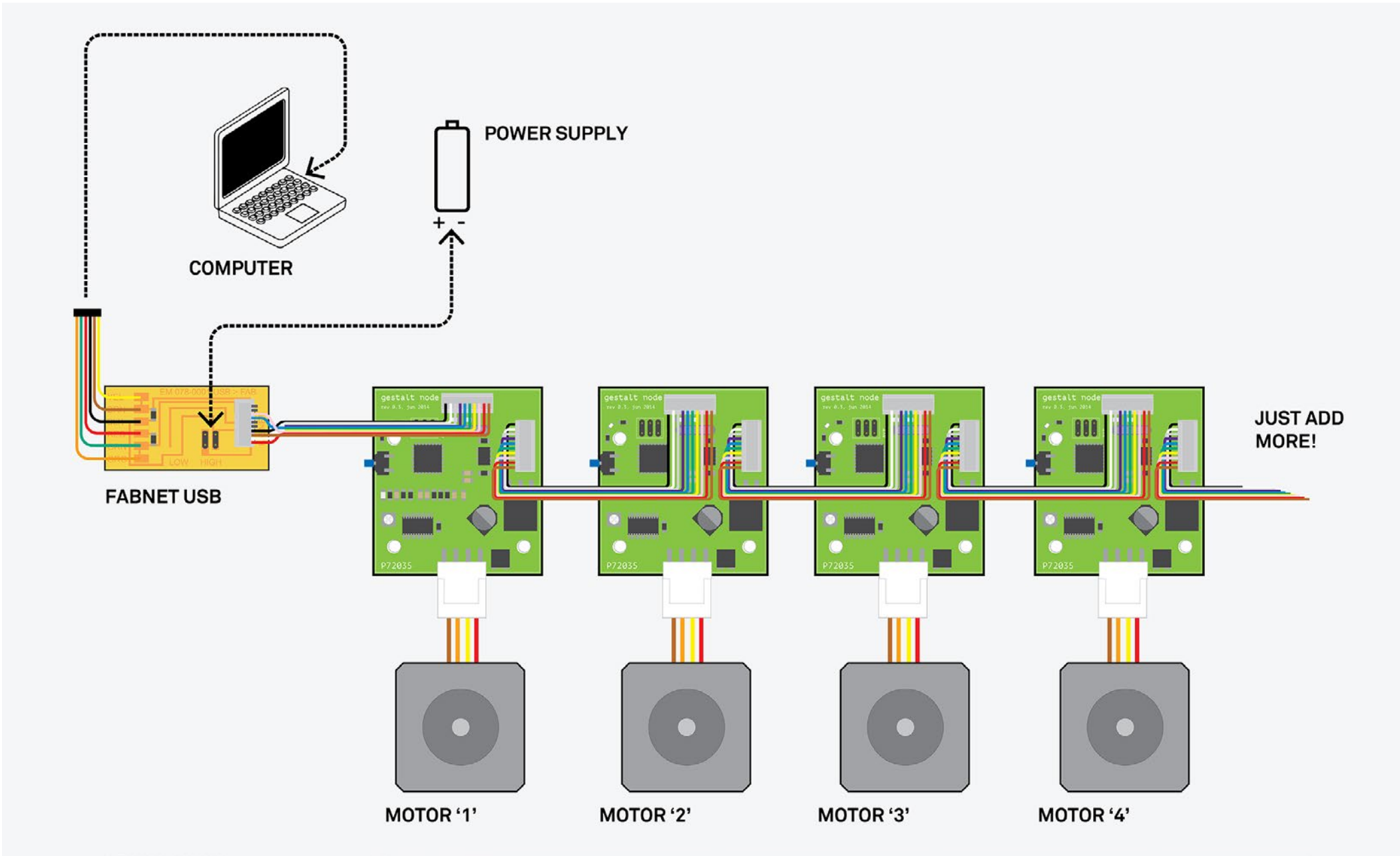
    def __init__(self, position = (None, None, None)):
        self.position.future.set(position)

    def __init__(self, speedFraction):
        self.machineControl.pwmRequest(speedFraction)
        pass

#---IF RUN DIRECTLY FROM TERMINAL---
if __name__ == '__main__':
    stages = virtualMachine(persistenceFile = 'test.vmp')
    stages.xyNode.setVelocityRequest(3)

    f = open('MIT.csv')
    CIRCOORDS = []
    for line in f.readlines():
```

Gestalt Framework



Networked nodes



Per machine:

1: stages + gestalt nodes

2: power supply + power supply/usb board

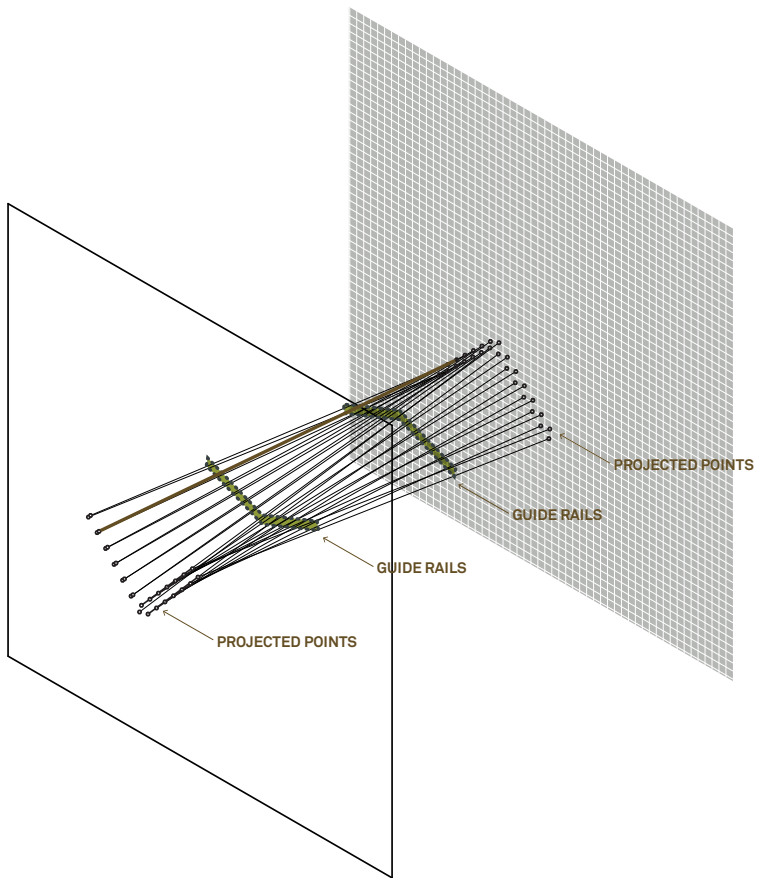
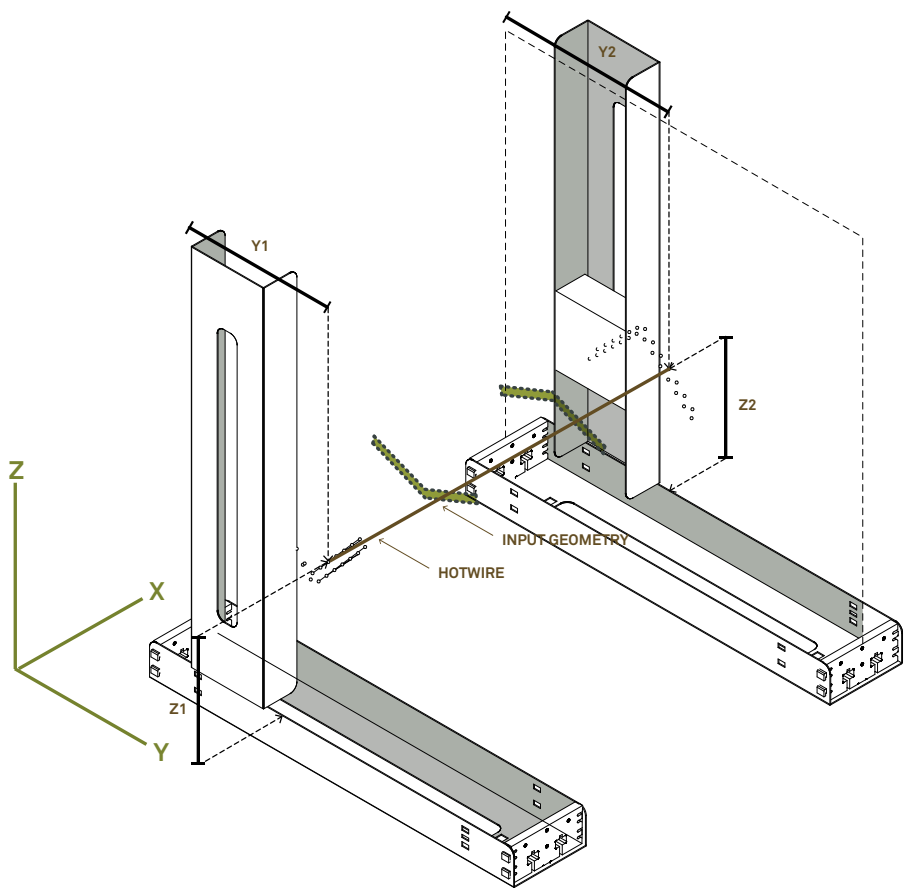
3: Virtual machine file

NO hot plugging

NO backdriving

NO drama!

HTMAA MACHINE DESIGN



OUTPUT

Point decomposition

Projected POINTS1

Swatch

Projected POINTS2

Y1	
0	57.350786
1	61.900312
2	65.553723
3	69.207134

Z1	
0	42.697303
1	44.595054
2	48.130075
3	51.665097

Y2	
0	54.527661
1	55.753626
2	59.407036
3	63.060447

Z2	
0	135.839191
1	136.179694
2	132.085068
3	127.990441

Move coordinates

0	Y1,57.350786,Z1,42.697303,Y2,54.527661,Z2,135.839191
1	Y1,61.900312,Z1,44.595054,Y2,55.753626,Z2,136.179694
2	Y1,65.553723,Z1,48.130075,Y2,59.407036,Z2,132.085068
3	Y1,69.207134,Z1,51.665097,Y2,63.060447,Z2,127.990441
4	Y1,72.860545,Z1,55.200118,Y2,66.713858,Z2,123.895815
5	Y1,76.513956,Z1,58.73514,Y2,70.367269,Z2,119.801188
6	Y1,80.167367,Z1,62.270161,Y2,74.02068,Z2,115.706562
7	Y1,83.820778,Z1,65.805183,Y2,77.674091,Z2,111.611935
8	Y1,87.474189,Z1,69.340205,Y2,81.327502,Z2,107.517309

SIMULATION

Index 111

Position Index

Toolpathing

Machines that Make

$$\text{END EFFECTOR POSITION} = \begin{pmatrix} x_E \\ y_E \\ z_E \end{pmatrix}$$

FORWARD KINEMATICS

$$x_E = (l_2 \cdot \cos \theta_2) \cos \theta_1 + dx$$

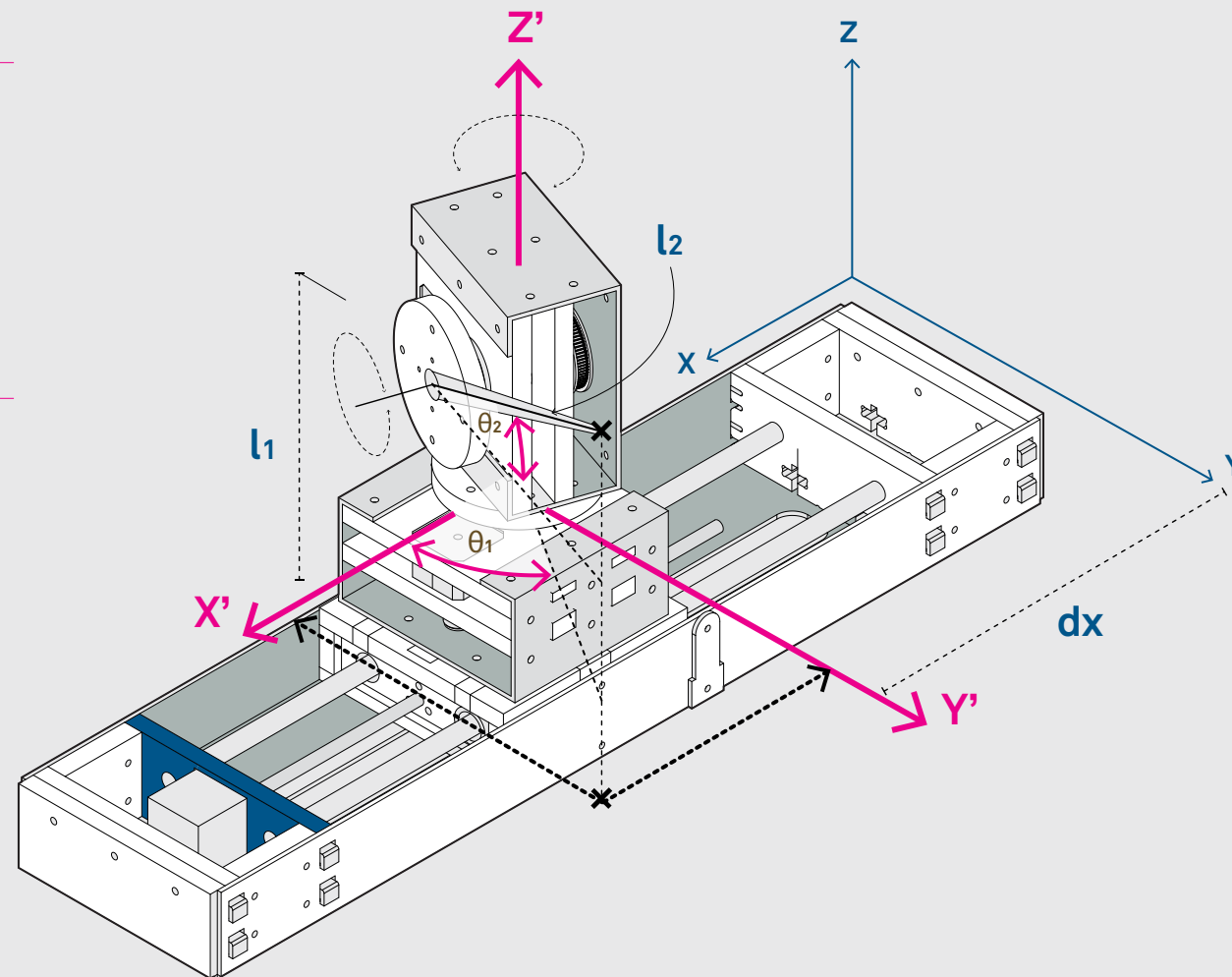
$$y_E = (l_2 \cdot \cos \theta_2) \sin \theta_1$$

$$z_E = (l_2 \cdot \sin \theta_2) + l_1$$

INVERSE KINEMATICS

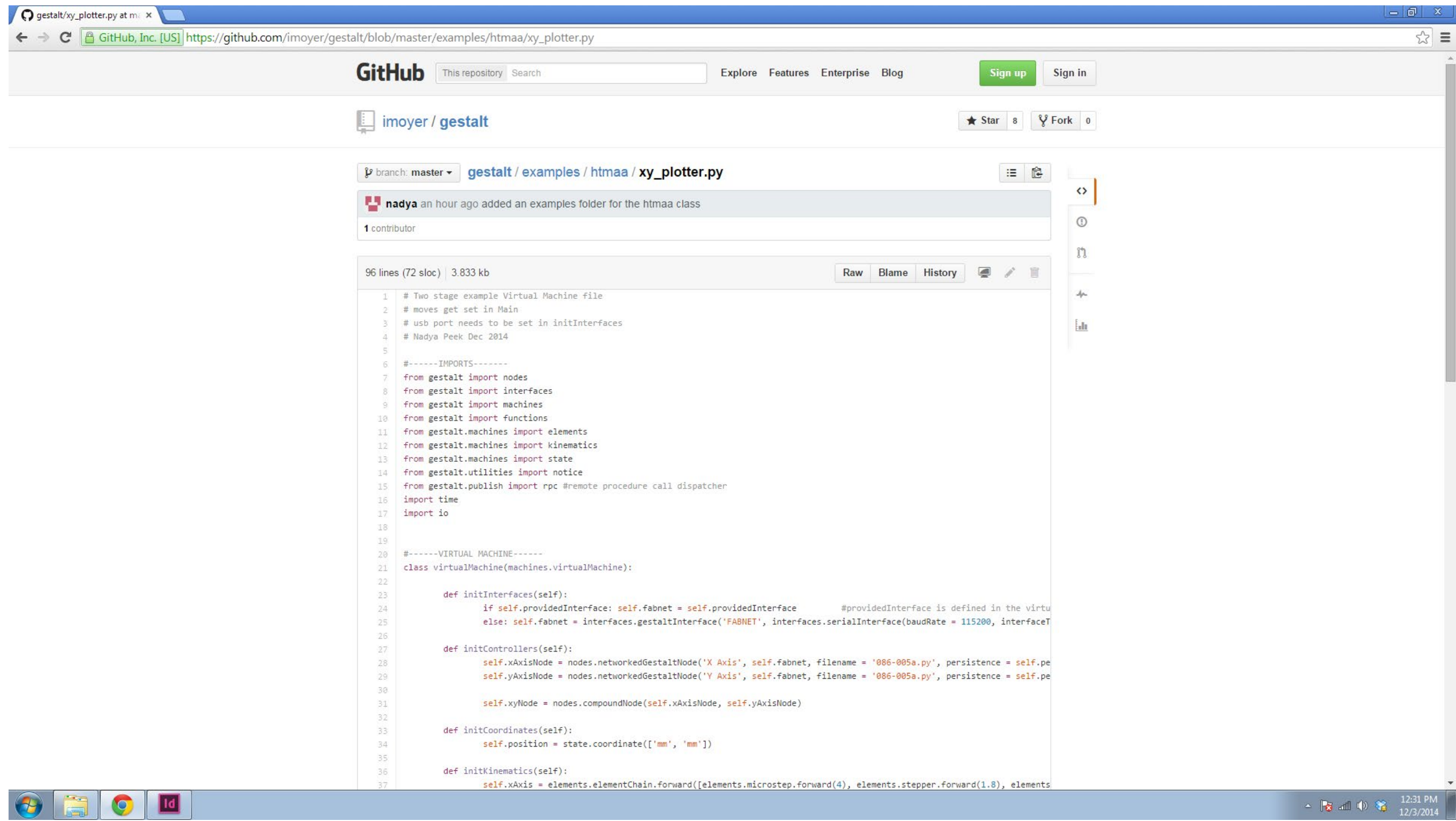
$$\theta_1 = \tan^{-1} \left(\frac{y_E}{x_E + dx} \right)$$

$$\theta_2 = \tan^{-1} \left(\frac{(z_E - l_1)^2}{\sqrt{(l_2)^2 - (z_E - l_1)^2}} \right)$$



Kinematics

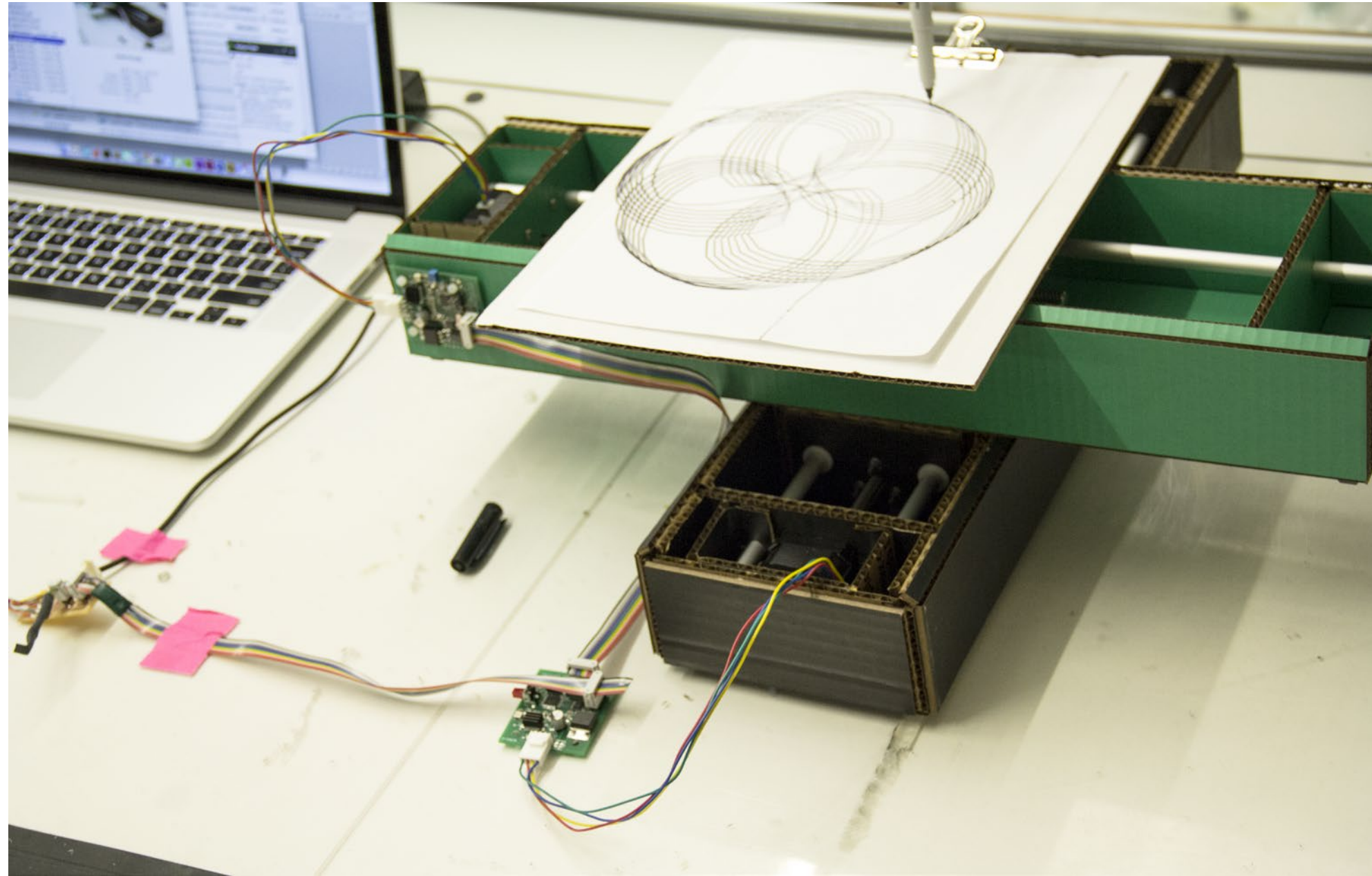
HTMAA MACHINE DESIGN



The screenshot shows a GitHub repository page for the file `xy_plotter.py` in the `examples/htmaa` directory of the `gestalt` repository. The page displays the commit history, with the most recent commit by `nadya` adding an `examples` folder. The code is shown in a diff view, with line numbers 1 through 37. The code is a Python file that defines a `virtualMachine` class, which inherits from `machines.virtualMachine`. The class includes several initialization methods: `initInterfaces`, `initControllers`, `initCoordinates`, and `initKinematics`. The `initInterfaces` method sets up a `fabnet` interface. The `initControllers` method creates `xAxisNode`, `yAxisNode`, and `xyNode` objects. The `initCoordinates` method sets the `position` attribute. The `initKinematics` method sets up the `xAxis` element chain.

```
1 # Two stage example Virtual Machine file
2 # moves get set in Main
3 # usb port needs to be set in initInterfaces
4 # Nadya Peek Dec 2014
5
6 #-----IMPORTS-----
7 from gestalt import nodes
8 from gestalt import interfaces
9 from gestalt import machines
10 from gestalt import functions
11 from gestalt.machines import elements
12 from gestalt.machines import kinematics
13 from gestalt.machines import state
14 from gestalt.utilities import notice
15 from gestalt.publish import rpc #remote procedure call dispatcher
16 import time
17 import io
18
19
20 #-----VIRTUAL MACHINE-----
21 class virtualMachine(machines.virtualMachine):
22
23     def initInterfaces(self):
24         if self.providedInterface: self.fabnet = self.providedInterface #providedInterface is defined in the virtu
25         else: self.fabnet = interfaces.gestaltInterface('FABNET', interfaces.serialInterface(baudRate = 115200, interfaceT
26
27     def initControllers(self):
28         self.xAxisNode = nodes.networkedGestaltNode('X Axis', self.fabnet, filename = '086-005a.py', persistence = self.pe
29         self.yAxisNode = nodes.networkedGestaltNode('Y Axis', self.fabnet, filename = '086-005a.py', persistence = self.pe
30
31         self.xyNode = nodes.compoundNode(self.xAxisNode, self.yAxisNode)
32
33     def initCoordinates(self):
34         self.position = state.coordinate(['mm', 'mm'])
35
36     def initKinematics(self):
37         self.xAxis = elements.elementChain.forward([elements.microstep.forward(4), elements.stepper.forward(1.8), elements
```

Virtual machine



Assignment: make your own machine w/ end effector

HTMAA MACHINE DESIGN

