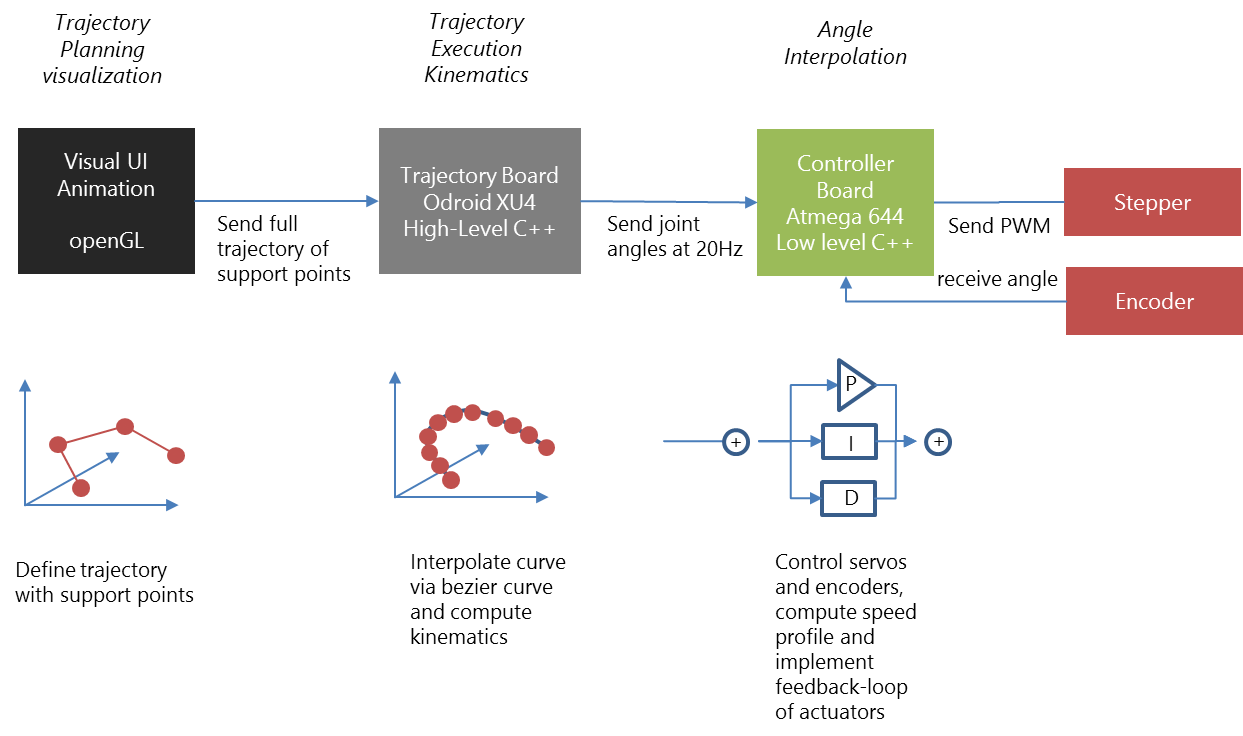


Building a bad Robot

Jochen Alt

# Architecture

In general, the data flow looks like this:



The visualizer is mainly the UI-based tool to plan trajectories by defining singular points. These points are sent to the Trajectory Execution, where the trajectory is interpolated using Bézier curves. Out of each point of the interpolated curve the angles are computed and send to the controller board, where a PID controller takes care that each actuator actually follows the curve.

On the mechanical side, we have two actuators driven by a servo (mainly due to space restrictions) and four actuators driven by a stepper/rotary encoder combination.

The servos (Herkulex DRS-101) are controlled directly by the controller board with an ATMega644 on board via a serial interface. The steppers do not have an internal feedback loop, so we need rotary encoders detecting the absolute angle of the joint and allowing to implement feedback controllers. Depending on the actuator, the steppers provide a torque between 26Ncm (turning the elbow) and 3,1Nm (upperarm). For space reasons, the gripper and the wrist is driven by a servo.

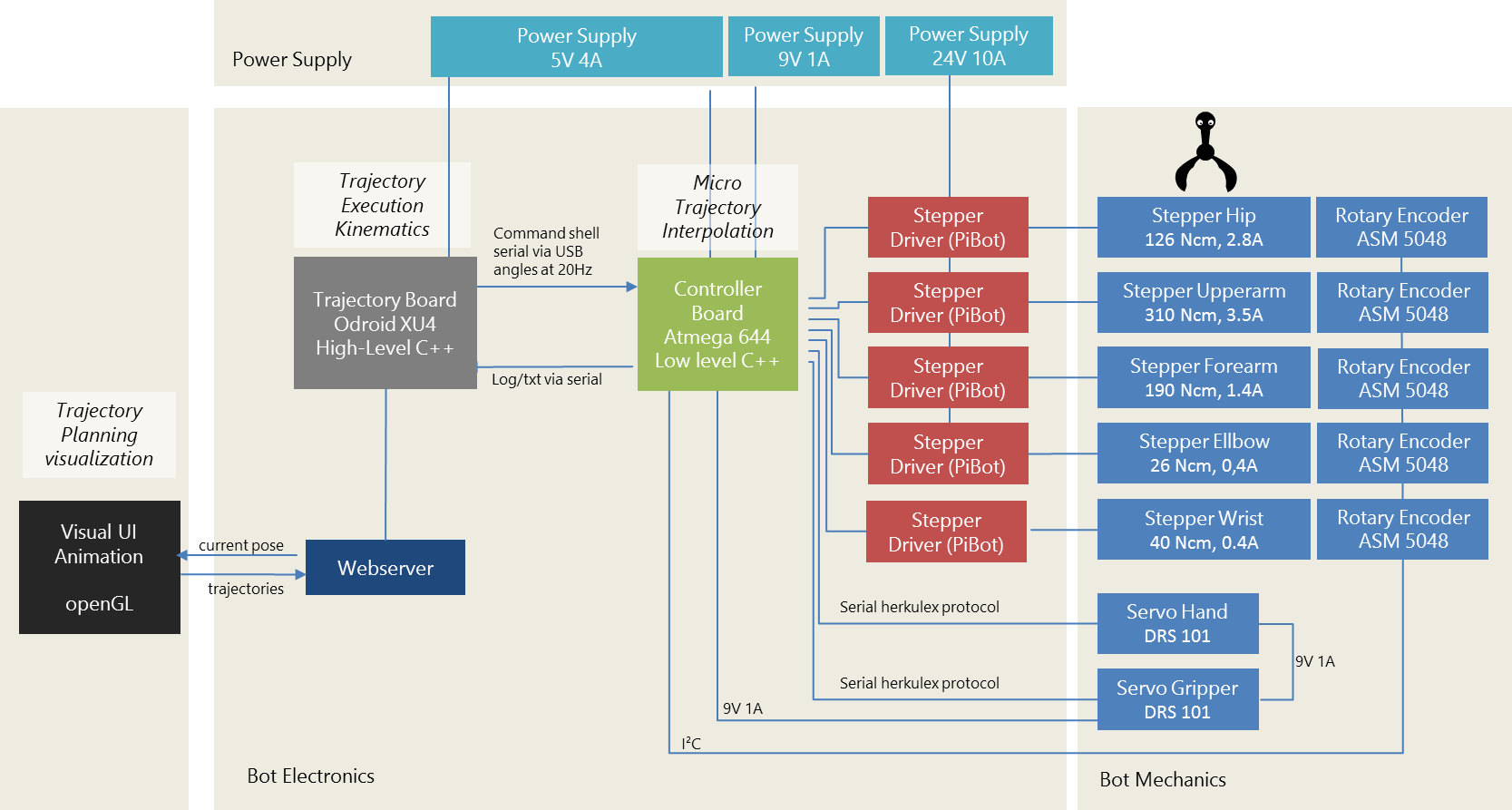


Figure 1‑1 Architecture

The steppers are driven by retail stepper drivers (PiBot Stepper Driver) around the popular PWM stepper driver Toshiba 6600 (4.5A max). The stepper drivers are directly controlled via the Controller Board. It receives joint angles at a sample rate of 20Hz, interpolates in between, and sends the according PWM signal to the stepper drivers and to the servos.. Besides micro interpolation of the trajectory, the controller board takes care of the speed profile, i.e. it limits the acceleration and speed of each actuator. The controller board is a DIY board around an ATMega 644 (for the sheer number of pins) running firmware in low level C++ on base of the Arduino library.

The controller board is fed by the trajectory board, which is an octa core board (Odroid XU4).

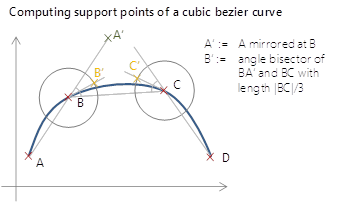
The trajectory controller board is encapsulated by a webserver exposing the current movement and accepting commands like new trajectories.

# Trajectories

Planning a trajectory means defining a sequence of poses in 3D space. These defined poses are interpolated in order to result in a smooth and continuous curve. Most beautiful are cubic Bézier curves for that purpose.

Bézier curves are polynoms of 3rd grade using a start and an end point and two support points defining the curvature at the start and end point. The computation is done on base of a parameter *t=0..1* defining the ratio of the current position and four points P0..P3, of which P1 and P2 are support points the curve does not touch.

|  |  |
| --- | --- |
|  | (2‑1) |

This computation is done for x,y, and z coordinates. Normally being beautiful, but Bézier curves have a tendency to “bounce”, if the support points P1 and P2 differ too much in terms of the distance to the trajectory points P0 and P3. So, it is necessary to normalize support points by a small trick:

The picture illustrates a trajectory defined by A, B, C, and D. We want to model the piece between B and C with a cubic Bézier curve.

The support point B’ is computed by taking point A, mirroring it at B (A’), and moving along the angle bisector of A’ B C by a 3rd of the length of BC, C’ is computed in an analogous manner.

This approach is rather arbitrary, but results in a smooth and non-oscillating curve.

Having a nice trajectory, we need to convert the coordinates into angles of the joints. This is done by inverse kinematics in the following chapter.

# Kinematics

Kinematics is about computation of the tool-centre-point (TCP) out of joint angles and vice versa. First is demonstrated in chapter 3.1 (simple), latter in chapter 3.2 (tricky).

But before starting any kinematics, it is necessary to define all coordinate systems.

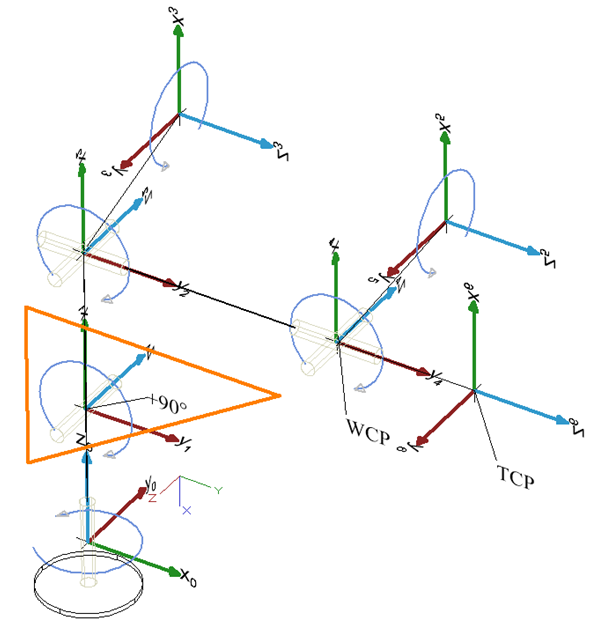


Figure 3‑1 Coordinate Systems in default position

The most important design decision is to let the three upper axis’ intersect in one point, the so-call wrist-center-point (WCP). This decision makes the computation of the inverse kinematic feasible.

The picture shows the used coordinate systems in the default position of the bot, having all angles at 0°, starting from the base (angle0) and ending with the coordinate system of the hand (angle6). For convenience the forearm (angle1) adds +90° to the real angle in order to have the base position at 0°of the bot, although the illustrated actually is -90°. The coordinate systems have been are arranged according to the Denavit Hardenberg convention, which is:

* The angle rotates along the z-axis
* The z-axis points on the direction of the next joint
* The transformation from anglei to anglei+1 is given via

1. rotating around the x-axis by 
2. translation along the x-axis by *a*
3. translation along the z-axis by *d*, and
4. rotation around the z-axis by joint angle

So, the Denavit Hardenberg parameters are:

|  |  |  |  |
| --- | --- | --- | --- |
| Joint | a[°] | a[mm] | d[mm] |
| hip | -90° | 0 | d0 |
| upperarm | 0 | a1 | 0 |
| forearm | -90° | 0 | 0 |
| ellbow | 90° | 0 | d3 |
| wrist | -90° | 0 | 0 |
| hand | 0 | 0 | d5 |

Table 3‑1 Denavit Hardenberg parameters

The general definition of a Denavit Hardenberg transformation is

|  |  |
| --- | --- |
|  | (3‑1) |

Combined with the DH parameters, the following DH matrixes define the transformation from one joint to its successor:

|  |  |
| --- | --- |
|  | (3‑2) |
|  | (3‑3) |
|  | (3‑4) |
|  | (3‑5) |
|  | (3‑6) |
|  | (3‑7) |

## Forward Kinematics

With the DH transformation matrixes at hand, computation of the bot’s pose out of the joint angles is straight forward. The matrix representing the gripper’s pose is

|  |  |
| --- | --- |
|  | (3‑8 |

By multiplying the transformation matrix with the origin (as homogeneous vector), we get the absolute coordinates of the tool centre point in world coordinate system (i.e. relative to the bot’s base).

|  |  |
| --- | --- |
|  | (3‑9) |

The orientation in terms of roll/nick/yaw of the tool centre point can be derived out of by taking the part representing the rotation matrix (). [[1]](#footnote-1)

|  |  |
| --- | --- |
|  | (3‑10) |
|  | (3‑11) |
|  | (3‑12) |
|  | (3‑13) |

Due to singularities, we need to consider and use

|  |  |
| --- | --- |
|  | (3‑14) |
|  | (3‑15) |

instead. if

|  |  |
| --- | --- |
|  | (3‑16) |
|  | (3‑17) |

**Note:** Unfortunately, the gripper’s coordinate system is not appropriate for human interaction, since the default position as illustrated in Figure 3‑1 is not. So, it is handy to rotate the gripper matrix such that the default position becomes. The according rotation matrix represents a rotation of -90° along x,y, and z, which results in a simple rotation matrix of

|  |  |
| --- | --- |
|  | (3‑18) |

In the following, this is not considered, but we stop at the coordinate system to simplify the computation.

## Inverse Kinematics

Inverse kinematics denotes the computation of all joint angles out of the tool-centre-point’s position and orientation. In general it is hard to give non-numeric solution, in this case it is possible since the upper three joint angles point to one point, the so-called wrist centre point (Figure 1‑1).

We know the TCP’s position and orientation in terms of roll, nick, yaw (.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | (3‑19 |

First, we need to compute the wrist-centre-point out the tool-centre-point. This is possible by taking the TCP and moving it back along the TCP’s orientation by the hand length. For doing so, we need the transformation matrix from the base to the last joint , which we can derive out of the TCP’s position and orientation.

To build the transformation matrix we need the rotation matrix defining the orientation of the tool-centre-point. This is given by multiplying the rotation matrixes for all axis which gives

|  |  |
| --- | --- |
|  | (3‑20) |

Now we can denote the transformation matrix of the TCP

|  |  |
| --- | --- |
|  | (3‑21) |

From the TCP’s perspective, WCP is just

|  |  |
| --- | --- |
| = | (3‑22) |

Furthermore, , so we get the wrist-centre-point by

|  |  |
| --- | --- |
|  | (3‑23) |

in world coordinates. Having a top view on the robot shows how to compute the first angle:

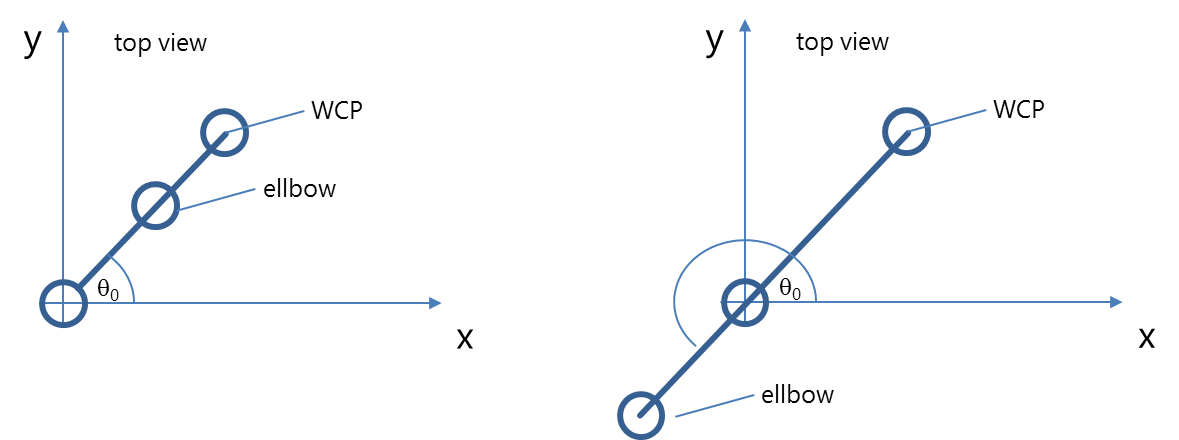


Figure 3‑2 Two solutions for viewed from above

|  |  |
| --- | --- |
|  | (3‑24) |

Actually, this angle exists in two variants: if the bot looks backwards, another valid solution is

|  |  |
| --- | --- |
|  | (3‑25) |

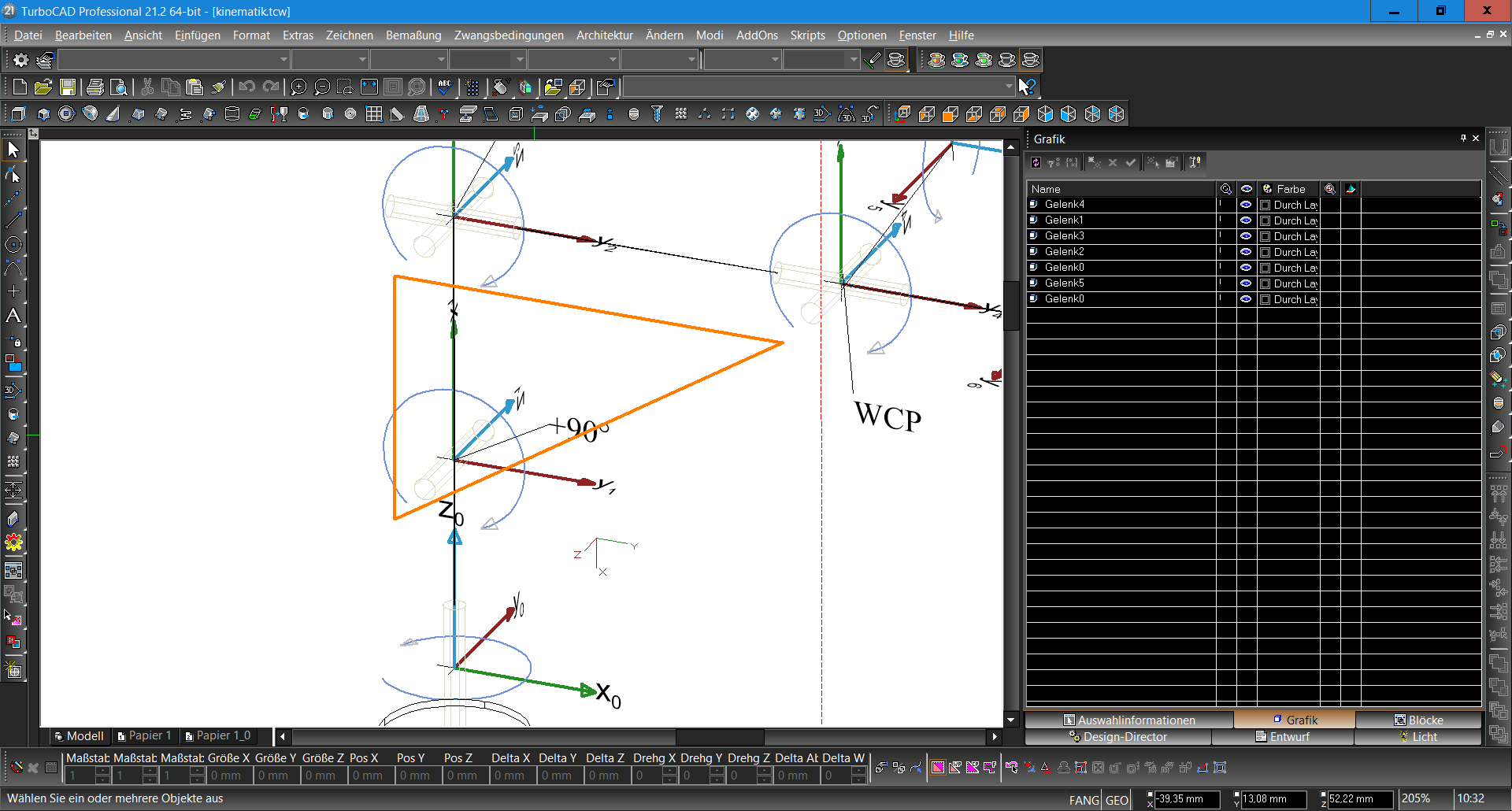
Thanks to the design having a wrist-centre-point where the axes of the three upper actuators intersect, the next two angles can be computed by a triangle:

Figure 3‑3 Triangle to compute and

Again, there are two solutions representing, one configuration corresponds with a natural pose of the elbow, solution II is a rather unhealthy position:

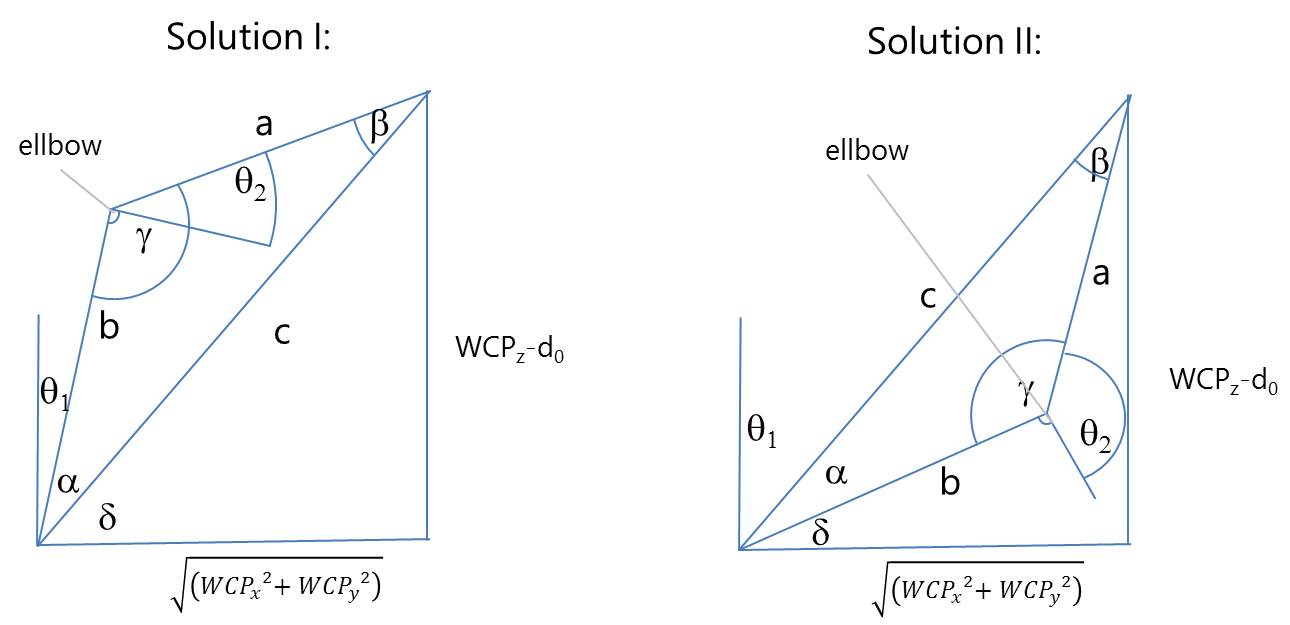


Figure 3‑4 Cosine sentence applied to compute and

a and b is given by the length of the actuators a1 und d3 (Table 3‑1). So, cosine sentence yields the angles and .

|  |  |
| --- | --- |
|  | (3‑26) |
|  | (3‑27) |

Finally, we the get

|  |  |
| --- | --- |
|  | (3‑28) |
|  | (3‑29) |

and – as second solution -

|  |  |
| --- | --- |
|  | (3‑30) |
|  | (3‑31) |

The upper angles, 4, 5 can be obtained by considering by the chain of transformation matrixes. With

|  |  |
| --- | --- |
|  | (3‑32) |

we get

|  |  |
| --- | --- |
|  | (3‑33) |

Although the need of the annoying multiplication out of (3‑5), (3‑6), and (3‑7) we only need to consider the rotation part of the homogenous matrixes, since just the orientation of the tool-centre-point is relevant here.

|  |  |
| --- | --- |
|  | (3‑34) |

- and therefore the rotation part -is already known by (3‑21).

resp. can be obtained out of the given angles ,1,2 by

|  |  |
| --- | --- |
|  | (3‑35) |

So, by equalizing the values of matrix with (3‑35) we get 4 :

|  |  |
| --- | --- |
|  | (3‑36) |

having two solutions

|  |  |
| --- | --- |
|  | (3‑37) |

For 4 there is no easy matrix element, but we can combine

|  |  |
| --- | --- |
|  | (3‑38) |
|  | (3‑39) |

to

|  |  |
| --- | --- |
|  | (3‑40) |

which ends up in

|  |  |
| --- | --- |
| ) | (3‑41) |

again having two solutions depending on 4 . Same is done on 5:

|  |  |
| --- | --- |
| ) | (3‑42) |

If 4=0, we have an infinite number of solutions 3 and 5 (gimbal lock). In that case, we consider

|  |  |
| --- | --- |
|  | (3‑43) |

since know the trigonometric addition theorem from school

|  |  |
| --- | --- |
|  | (3‑44) |

and get

|  |  |
| --- | --- |
| = - | (3‑45) |

We are free to choose and take the bot’s currentangle to not move unnecessarily.

|  |  |
| --- | --- |
|  | (3‑46) |

In the end, we get max. eight solutions by combining the possible pose configurations of (forward/backward), and (triangle flip), and (upper orientation turn).

The “right” solution is chosen by taking the one that differs the least from the current bot’s joint angles.

# Trajectory Execution

Now we have the trajectory in terms of a sequence of joint angles. This needs to be translated into movements of the motors. This translation should be done in a manner that no motors limits are violated, and with a speed profile that avoids vibrations. Typically, this is done with speed profile of constant acceleration. I.e. if the required acceleration is higher than an upper limit, the speed is increased with a limited acceleration until the to-be speed is reached.

Additionally, we need to control a feedback loop to ensure that the to-be angle of the motor is actually reached.

All this is done in the controller board. It receiving the joint angles at 20Hz and the angles of the magnetic encoders, and let that control with a PI controller producing the PWM signal for the steppers.

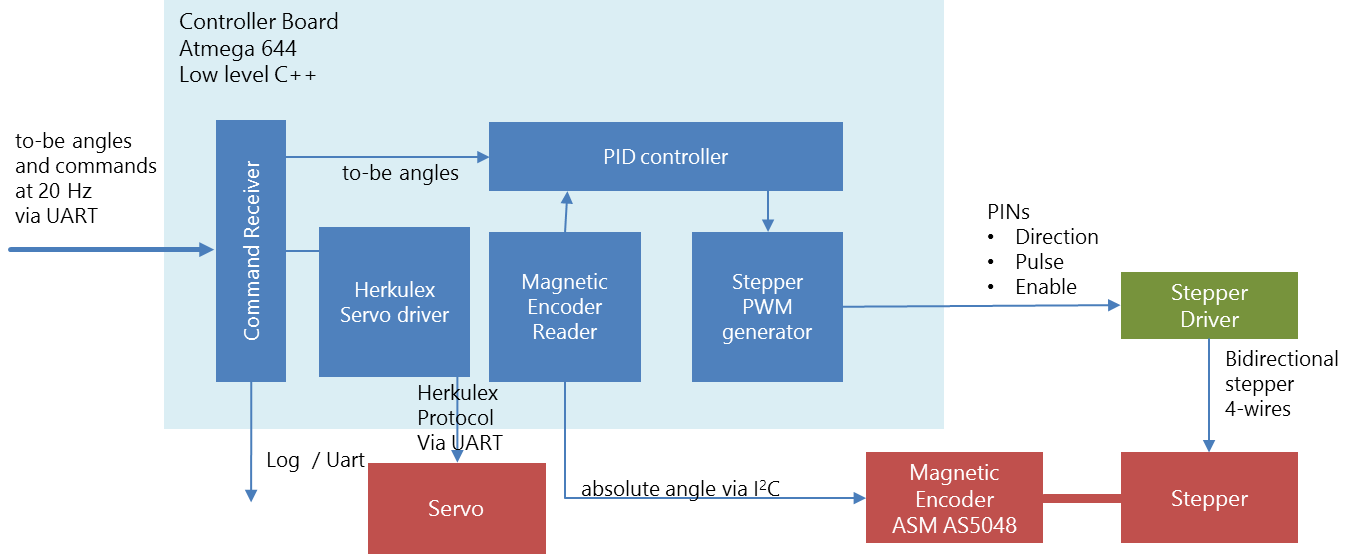


Figure 4‑1 Components of the Controller Board

The software runs on the basis of the Arduino library

Besides software, this requires just a few components, it is mainly an 8-bit ATMega 644 overclocked at 24 MHz[[2]](#footnote-3).

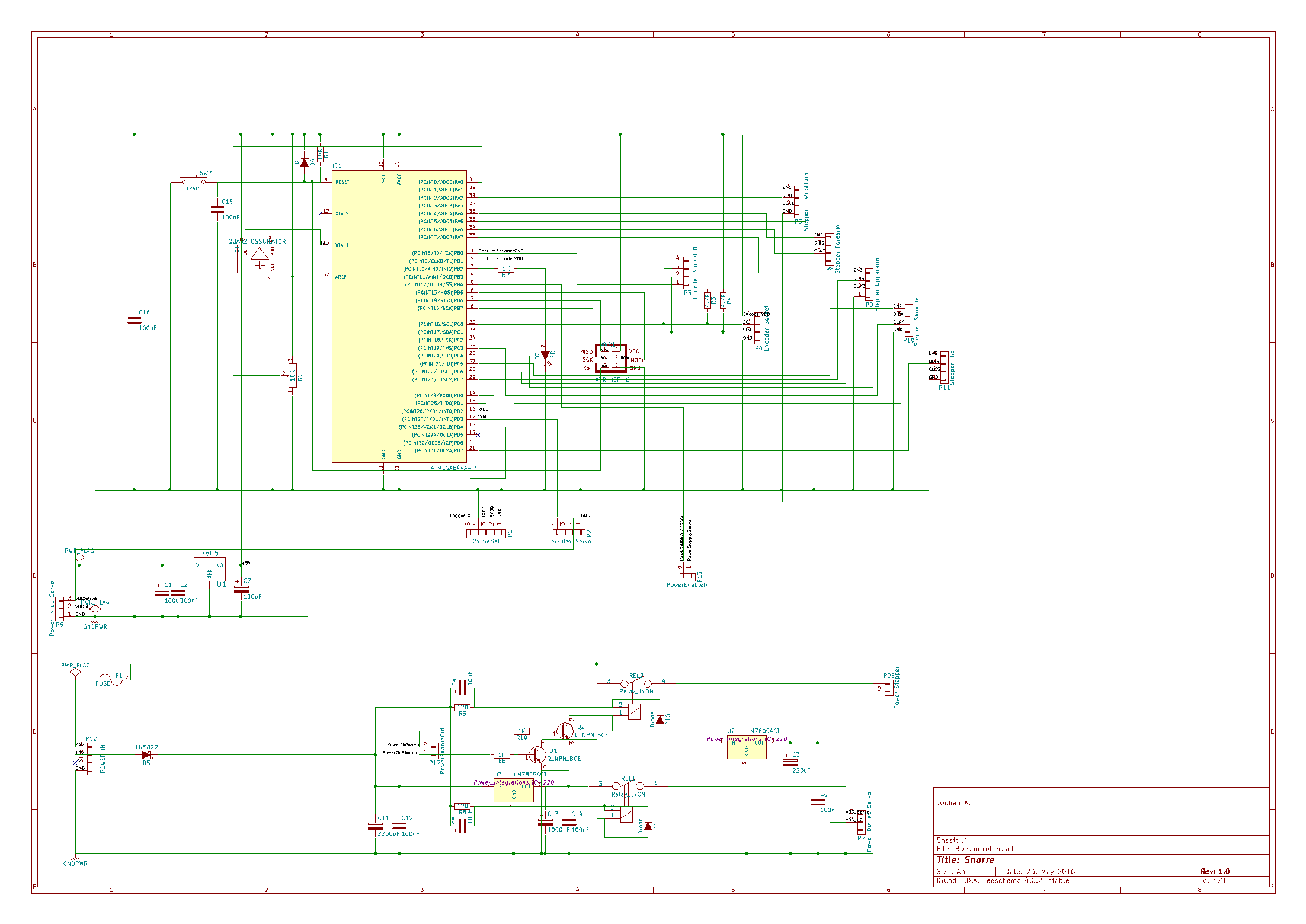
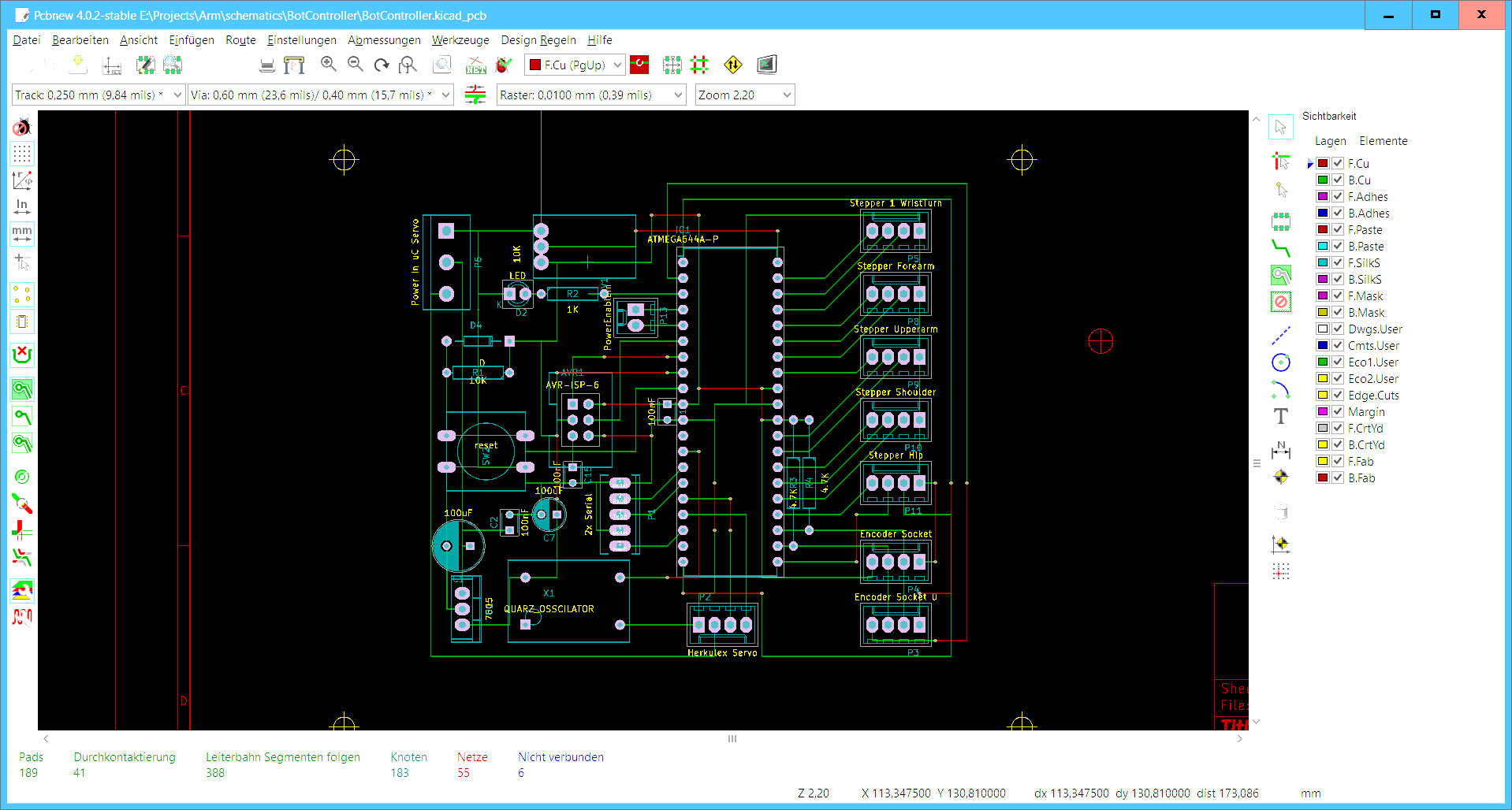


Figure 4‑2 Schematics of the controller board

Most of the layout is for the power supply at the bottom. The C gets 5V by a 7805, the two servos are driven by a 7809 providing 2A. Additionally, there are two relays for a proper start-up procedure switching the 240W power supply for the steppers and the power supply for the servos. This has been necessary to avoid impulses to the motors inducing annoying ticks when power is switched on.

On the top there is the ATMega controller with an integrated-oscillator (to ensure a properly running software-based UART) and lots of sockets for the stepper drivers, encoders and servos. All this fits on a small PCB:

Figure 4‑3 Layout of the controller board

# References

1. Controller board schematics   
   ./schematics/BotController.sch created with KiCad
2. Controller board layout  
   ./schematics/BotController.kicad-pcb

1. <https://de.wikipedia.org/wiki/Roll-Nick-Gier-Winkel#Berechnung_aus_Rotationsmatrix> [↑](#footnote-ref-1)
2. I regard overclocking of self-made PCBs as an obligation [↑](#footnote-ref-3)