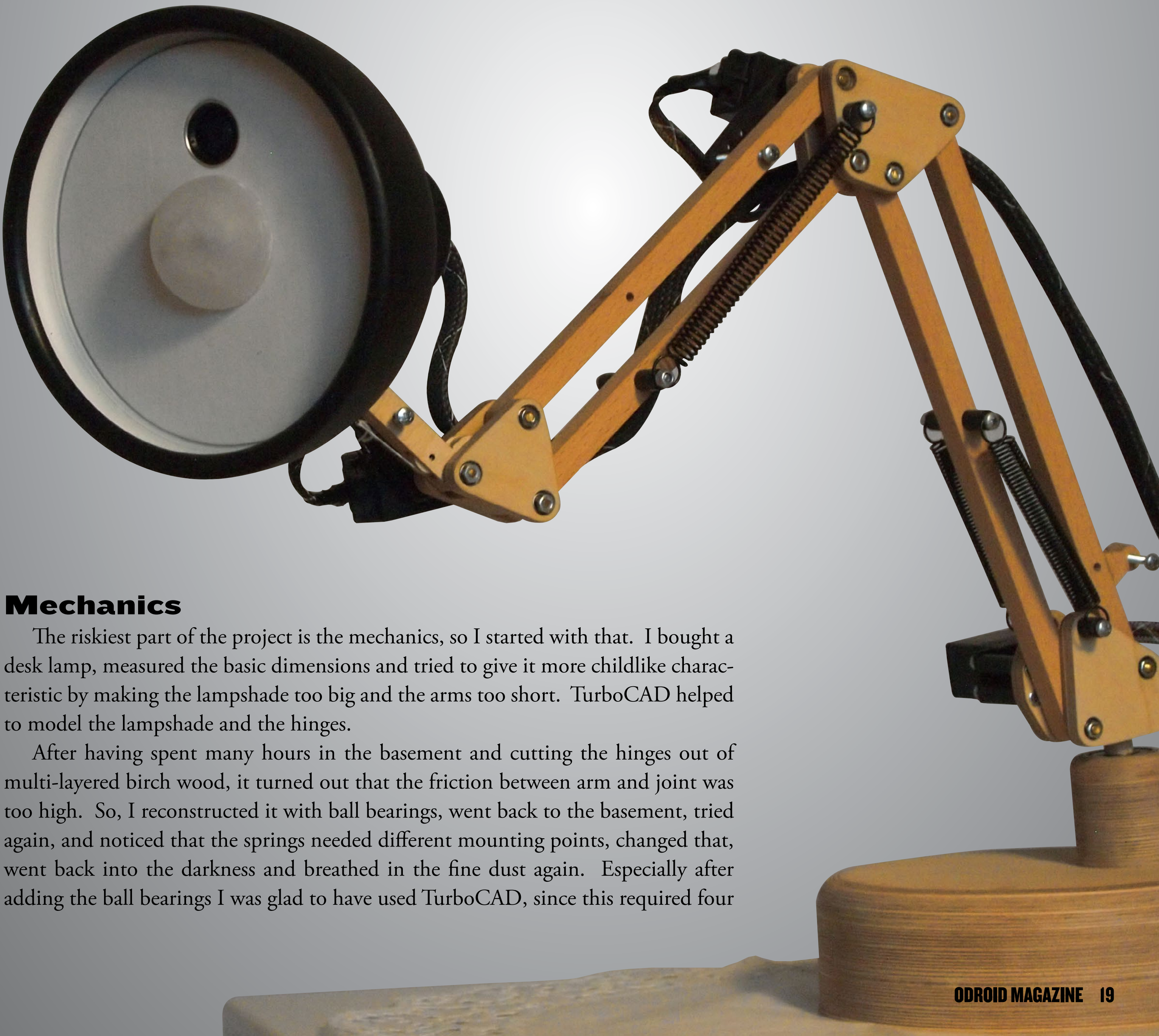
LUCI, MY LAMP

A SUPERCOOL ROBOTIC COMPANION POWERED BY AN ODROID-U3

by Jochen Alt

A lamp is nearly useless in the daytime. By itself, it does not talk, cannot dance, and is boring. This is where I try to help with my new project called Luci. Luci is an autonomous lamp with a webcam, a high power LED in the lampshade, and five servo motors. She is controlled by an ODROID-U3. Once switched on, she looks around, checks the environment for any sentient beings, and then does what she wants. Check out her video at http://bit.ly/1hLVtOt.



Mechanics

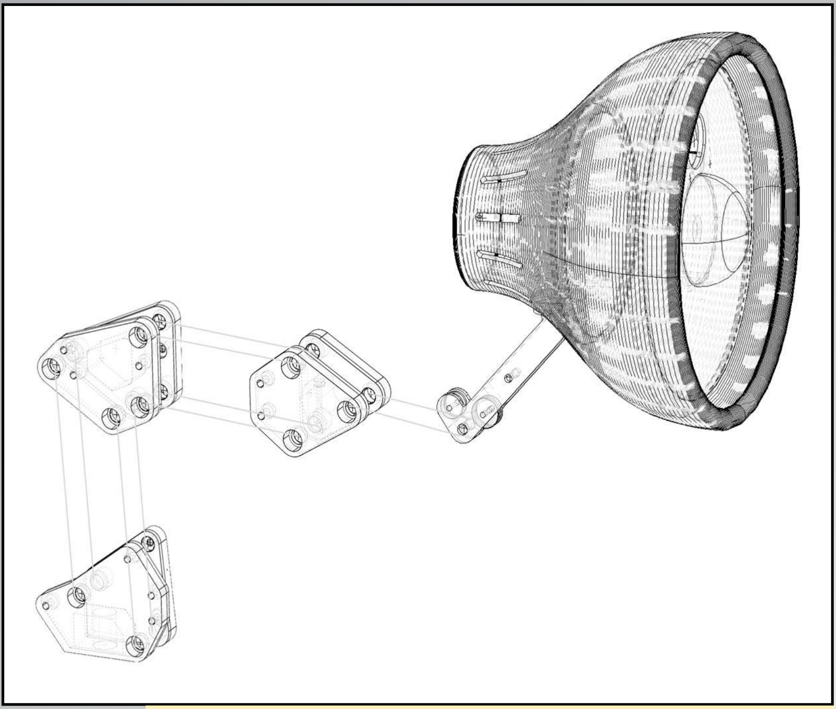
The riskiest part of the project is the mechanics, so I started with that. I bought a desk lamp, measured the basic dimensions and tried to give it more childlike characteristic by making the lampshade too big and the arms too short. TurboCAD helped to model the lampshade and the hinges.

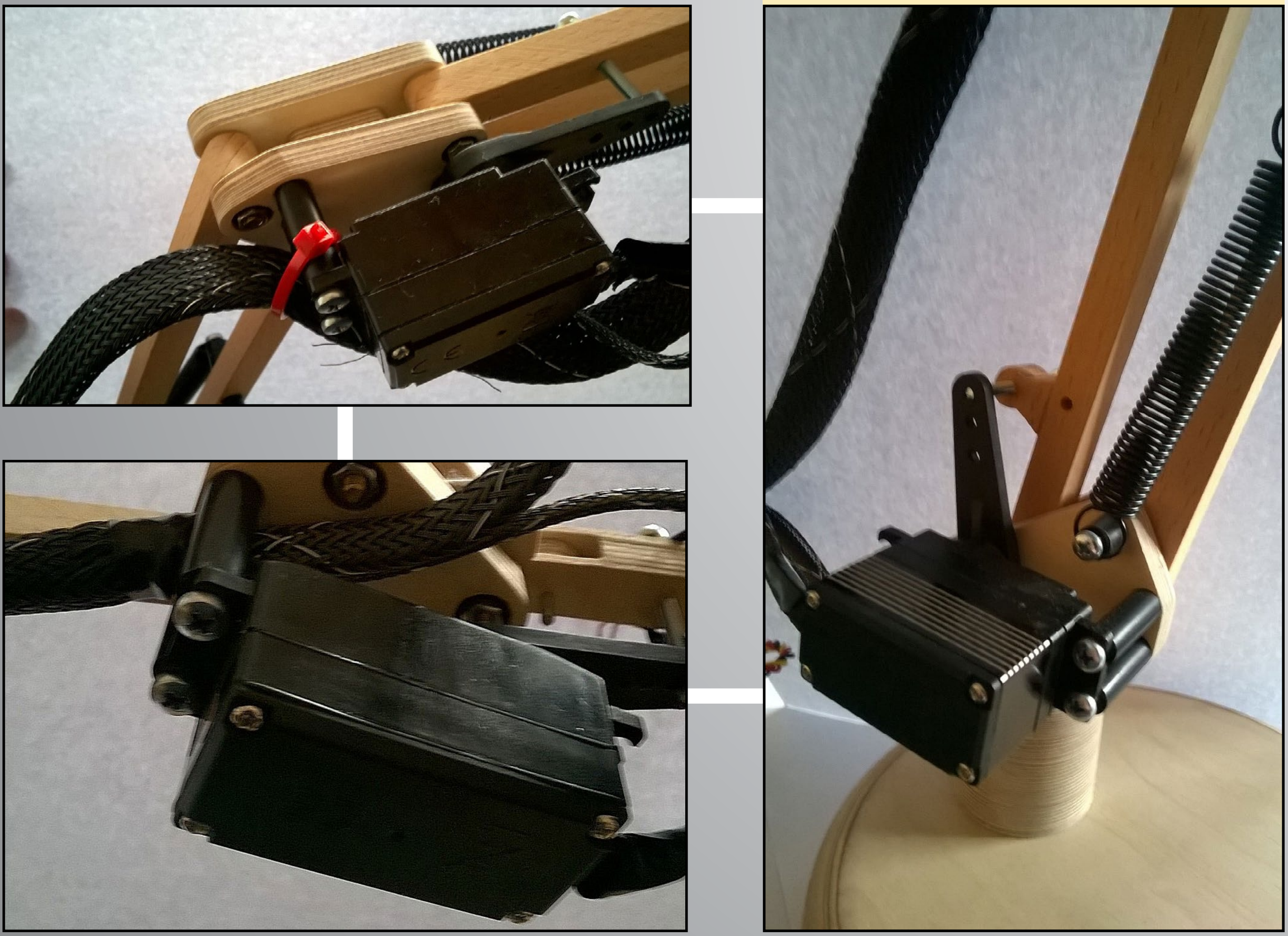
After having spent many hours in the basement and cutting the hinges out of multi-layered birch wood, it turned out that the friction between arm and joint was too high. So, I reconstructed it with ball bearings, went back to the basement, tried again, and noticed that the springs needed different mounting points, changed that, went back into the darkness and breathed in the fine dust again. Especially after adding the ball bearings I was glad to have used TurboCAD, since this required four slices per hinge with very precise drill holes.

I did not want to repeat that dusty experience with the lampshade, so 3D printing became very tempting, and I wanted to try it out anyway. I can't report too much on that, since I just gave the CAD model to a 3D printer shop. The biggest challenge of that step was to withstand their desperate attempts to sell me a printer.

Hardware

I started with a BeagleBoard, but it became clear soon that it is too slow for facial recognition. So I ordered an ODROID-U3. Unfortunately, it does not provide five PWM outputs for the servos, so I needed a separate µC to produce the PWM signal for the servos and the High Power LED. In addition, the ODROID is sensitive to voltage flaws, so after having connected the servos to a separate board with its own power supply, sudden resets of the ODROID did not occur anymore. The ODROID with the self-made Arduino shield was supposed to be placed in the base housing of the lamp but, in the end, I did not place it there, since I was wary of having a hot ODROID under full load within a wooden box full of splints.

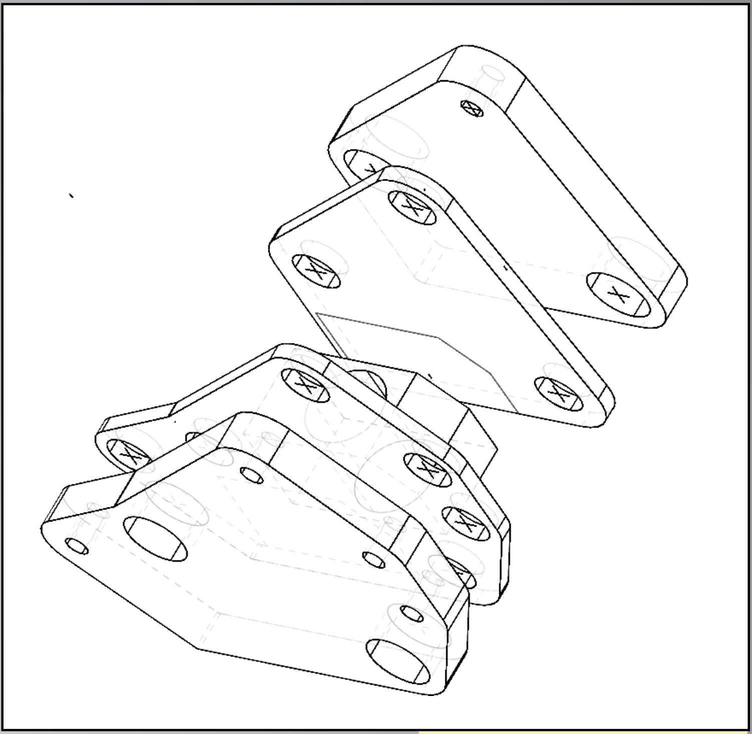

CAD diagram of Luci


Hinge closeups

Software

The program runs under Ubuntu using C++ with multiple threads in order to leverage most of the four cores. The first thread grabs the image from the webcam; a second thread takes these images and tries to find all faces currently looking at Luci. The result is a list of faces, and Luci focuses on the biggest one.

A third thread runs the trajectory planning algorithm, which produces a sequence of points and orientations in 3-dimensional space generated by certain patterns. When no face is detected, Luci runs a search pattern looking for faces by sampling the environment until a face has been found. Then, Luci carries out a pattern simulating real emotions like nodding without knowing why, pretending to listen, coming closer or retreating depending on the movements of the face. It's like in real life at home.


Hinge explosion diagram

Trajectory Planning

The implementation of the trajectory patterns is rather simple: whenever Luci runs short of queued points to be approached, she invokes a pattern point generator which is parameterized with the current pattern. There, the next point of a predefined sequence of movements is generated. In case of the pattern that interacts with a face, this is:

Move back quickly (surprise, recognizing a familiar face)
Move slowly towards the face (first shy contact)
Watch face from various angles (closer inspection)
Move down and watch the face from a frog's perspective (looking cute)
Go up again and nod (pretending an agreement)

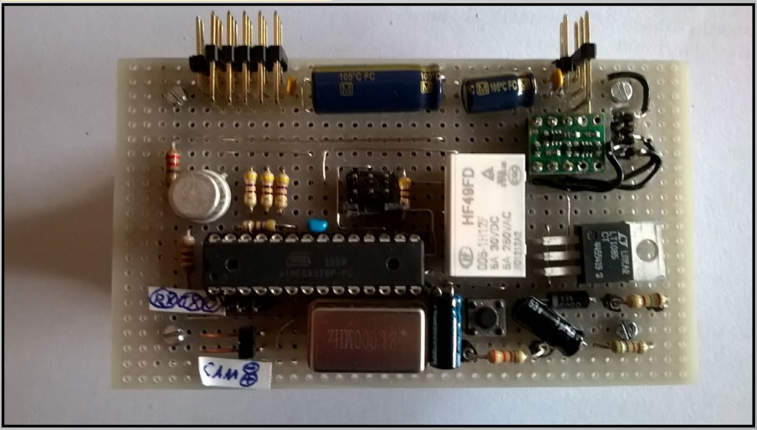
The ideas were borrowed from Eliza (http://bit.ly/1Co34ab), but coded in a robot instead of Emacs. Some patterns with special movements are hard-coded. For example, when Luci pushes a box from the table or looks guilty for watching dirty pictures (1:34 and 2:00 in the video).

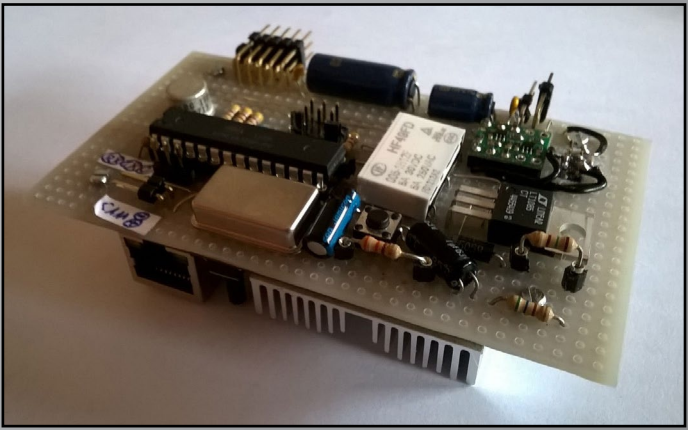
Finally, the main loop takes the previous and next point of the trajectory and interpolates all intermediate points with 60Hz using a cubic Bézier curve to smooth the movement. The support points of the Bézier curve are geometrically derived from the trajectory's prevprev (A) and nextnext (D) point by the rule shown in the picture: Since any polynomial with higher grade tends to oscillate when support points are too far away, I kept them in a constant distance of |BC|/3 to B resp. C.

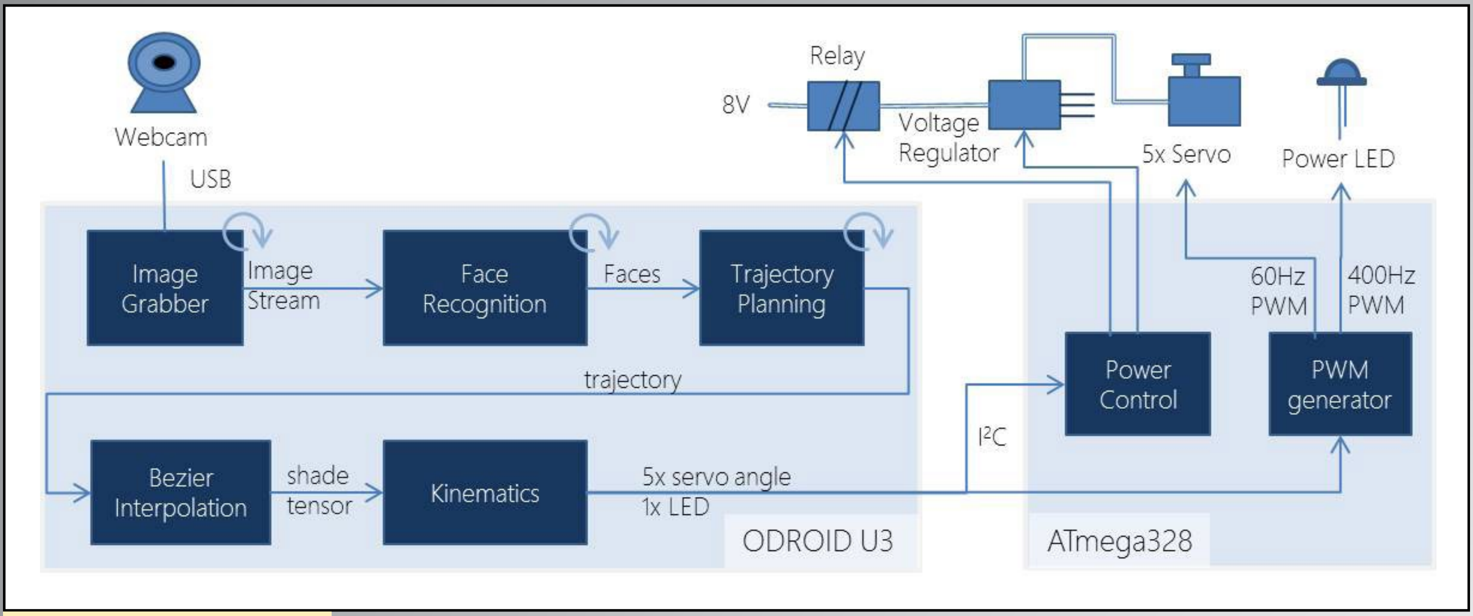
Mass inertia

The last step also computes the lampshade's acceleration, since the Bézier curve does not take into account that, in total, 400 grams are moved. As a consequence, I limited the mass acceleration by ½ g to prevent flapping caused by the elastic construction and the backlash of the servo motors. This is done by checking whether the next position can be reached without accelerating above the limit. If not, the new position is computed by taking the current position and adding the maximum dis-

PCB closeups







Software structure

tance (on the basis of the current speed and maximum acceleration capped by ½ g) along the current speed vector. In the end, the result curve leaves the Bézier curve where it is too sharp. Professional robots do this on the basis of fully modelled inertia, but at this point the methematics tired me out, so I did not try that.

Kinematics

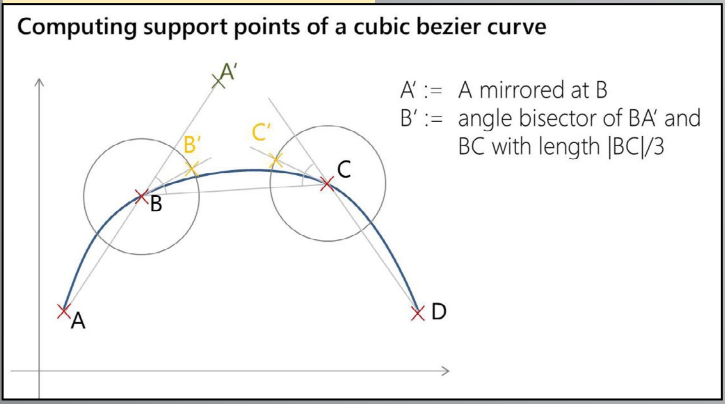
The output of all this is a 3D point which is passed to the kinematics module that computes the angles of all servo motors. This part is textbook robotics, it works as follows:

The algorithm starts with the point / orientation (=tensor) of the head's centre A. First step is to compute position B and C out of the head orientation. This can be done by computing the point C relative to the position of A (C.point-A.point), rotating that by the orientation of the head (A.rotation), and adding it to the point A:

```
C := A.point + rotate(C.point-A.point, A.rotation)
```

Then, the base angle at F, which is the servo angle, can be computed by:

Bezier support points diagram


Computing support points of a cubic bezier curve
A'
B'
C'
B
C
A
D
A' := A mirrored at B
B' := angle bisector of BA' and BC with length |BC|/3

```
F.angle := atan2(A.point.z, A.point.x)
```

The angles at E and D are computed by considering the triangle EDC and computing its angles with the cosine law:

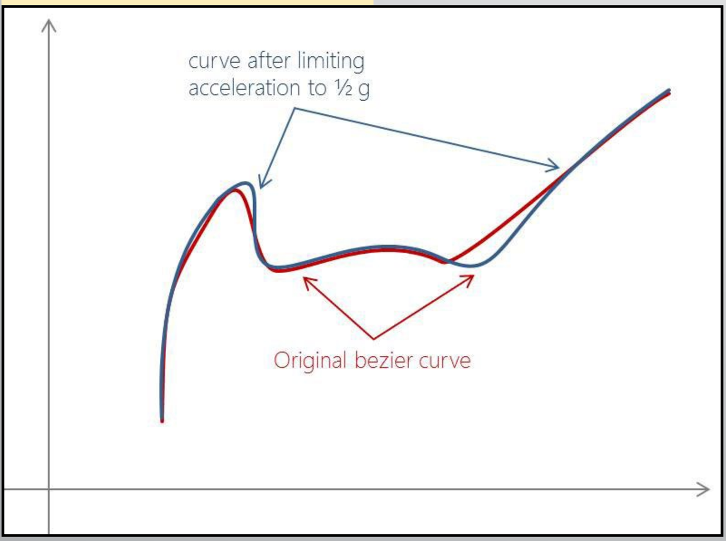
```
E.angle := 90° + acos( distance(E,D)2 +
  distance(E,C) 2 - distance(D,C)) /
  (2*distance(E,D) * distance(E,C) )
```

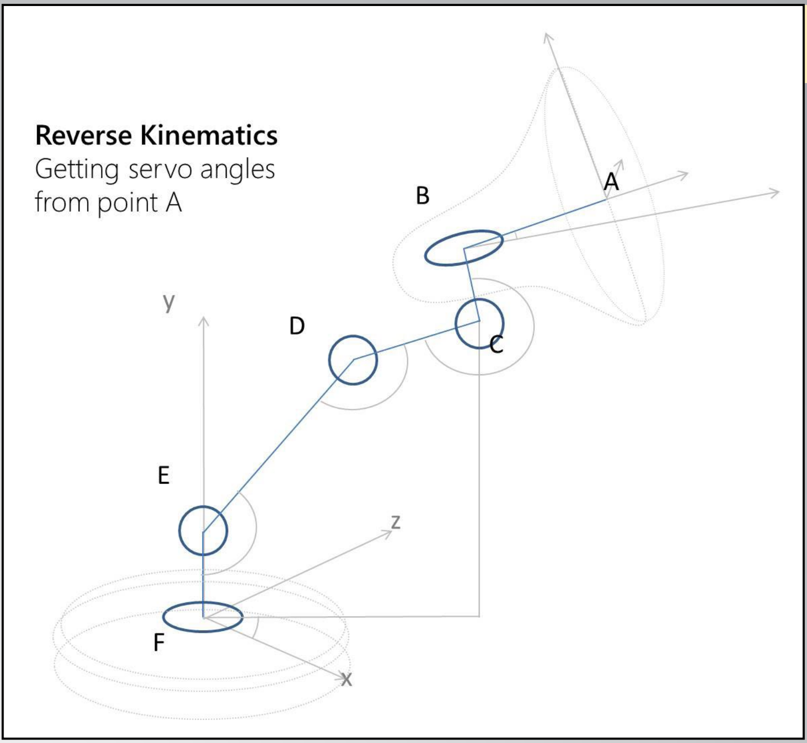
The angle at D is computed in the same manner

```
D.angle := acos( distance(E,D)2 +
  distance(D,C) 2 - distance(E,C)) /
  (2*distance(E,D) * distance(D,C) )
```

The ,ast servo is C, which angle is the orientation of the head around the z-axis minus the angle of CD to the horizon:

Acceleration limit diagram





Kinematics diagram

```
C.angle := A.rotation.z  + 270° -
  acos( C.point.y-E.point.y  /  C.distance(E) )
```

These angles are passed via I2C to the ATmega, where the Arduino library generates a 60Hz PWM signal for the servos. In the beginning, I was concerned about the high CPU use of 3D kinematics, and tried to implement it with fixed point integers and interpolated trigonometry, since I was used to a 24MHz ATmega. What a surprise it was when I recognized that using floats and sin/cos with no caching or table lookup had no noticeable performance impact.

Facial recognition

The facial recognition module uses OpenCV 3.0 with Haar cascade classifiers. Although the newer LBP cascades are significantly faster, they had many more false positives, so I thought 10 fps with Haar cascades is sufficient. From the 2D position of the detected face, the 3D position is estimated assuming a standard face size, which worked surprisingly well. Later on, Luci's trajectory planning module moves towards the face if it is very close, in order to simulate real interest, then moves away if it violates the European intimacy distance. Tracking a face in real time was a bit tricky, since grabbing images from the video stream and facial recognition has a latency of 250ms . So, the computation of the face's 3D position needs to be done relatively to the webcam's position 250ms ago. Consequently, when Luci moves quickly, this does not work well when the image becomes blurry, so the orientation of the head is directed towards the last stable face position until Luci moves slower and following the face in real time becomes possible again.

The trajectory planning module computes the next two points in advance for calculating the Bézier curve. Therefore, the detected face position is not valid anymore when the Kinematics module is sending a position to the servos a couple of seconds afterwards. The solution is to permanently compute the current 3D position and send that to the Kinematics module, in order to change the head orientation towards the face in real-time.

Conclusions

Similar to my experience with my work at my job, everything just took longer than expected. Surprisingly, the software and hardware worked out quickly, but getting the construction and the mechanics in a shape that worked without being too heavy, with properly mounted servos and springs, took me a couple of weekends in the basement. The mathematics were definitely challenging. Getting facial recognition done was the simplest part, but gets the most ahhs and ohhs. The guys from OpenCV did a pretty good job at making this really easy. The most fun part was the trajectory planning, such as how Luci should move when the webcam recognizes a face moving.

I have been thinking of enhancing Luci with a microphone and a beat detection module allowing her doing cool moves according to the music. My first thought was that it wouldn't be difficult, but rhythm detection seems to be extremely tricky.

Parts List

ODROID-U3 running Ubuntu 14.04.02 with the latest g++ compiler

Software uses OpenCV 3.0 and Boost 1.57 as base libraries

ATmega 328 running C++ firmware based on Arduino Library for Servos and I2C

Servos from Hitec: 77B (for turning base & nicking the head), 7954SH (lower strut, strong & expensive), 7775MG (upper strut, also expensive), 5065MG (turn head inside the lampshade)

3D print of a TurboCAD model made of ABS

Springs from my local dealer, 20 ball bearings, 0.5m2 multi-layered birch, and several brass axis

Source code on http://bit.ly/1iq9amT

Other projects

Two years ago, I was bored by using Excel too much at the office. Since I'm a professional software architect, I started learning electronics and embedded technology, and made my first robot, which can be viewed at http://bit.ly/1VHg7OX. His name is Paul, and his main purpose is to balance on a ball.