Non-contact waist circumference measurement

Dr. Dusadee Treemnuk
NECTEC, Thailand

Vincent Maire
INSA Toulouse, France

July 2018

Introduction

In the context of a research of innovative medical processes, it has been found that, despite the waist circumference is a very important anthropomorphic information which can prevent NCDs, there is a lack of systems able to measure it. Some are available on the market but are very expensive and not user-friendly. Thus, this project aims to find a new way and implement a system in order to measure the waist circumference without contact with the user.

This internal report cover the technical aspects that have been implemented during this project. Moreover, as it takes part of an internship, you will find in Vincent Maire's report [1] the details of the solutions considered but not kept.

First, this report will review the specifications of the desired system, then it will discuss about the hardware and the algorithm used to measure the waist circumference.

Note that this report is subject to evolve through time and is available in the git repository available at: `https://github.com/Vincema/lidar_waist_scan`.

The source code of the softwares can be found at: `https://github.com/Vincema/lidar_waist_scan` and `https://github.com/Vincema/navel_height_prediction`.

Contents

1 General design and specifications

The goal of the project is to develop a non-contact waist circumference measurement device.

1.1 General design

The solution chosen here is a set of 3 LIDARs mounted on a pole each and surrounding the user to gather a point cloud. Then, a curve is reconstructed from the points and the circumference. A schematic of the system and a picture of the prototype of a single pole are shown in the figure 1 and 2 respectively.

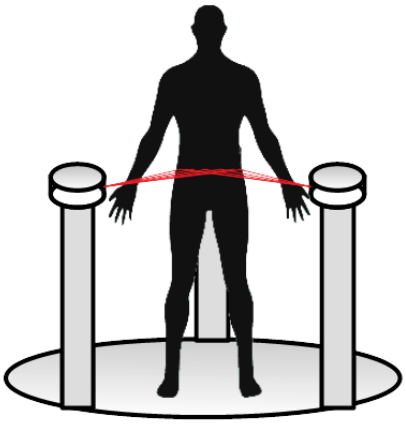


Figure 1: Simplified schematic of the system
The user is surrounded by 3 poles and 3 LIDARs. Each LIDAR is mounted on a servomotor.

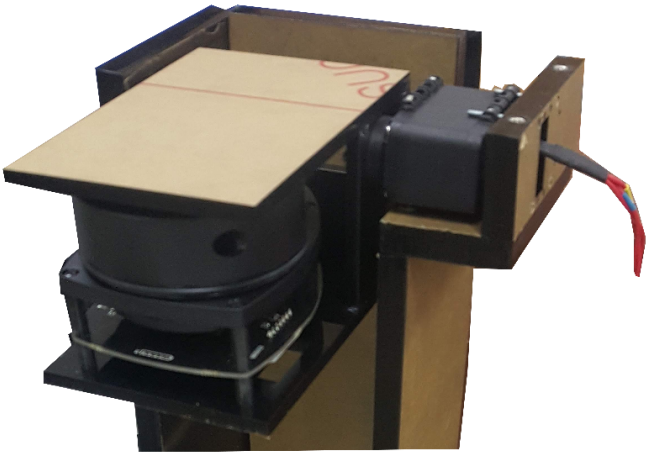


Figure 2: Prototype of a pole
We can see the servomotor which tilts the LIDAR.

To be able to aim at the user's navel (the level at which the measurement should be made), the LIDARs are mounted on a servomotor each which are on the top of a pole.

Then, a micro-controller controls the actuators and sensors, compute the circumference and send the result to an interface.

1.2 Specifications

The system must be:

- Automatic: no need any user operation to perform the measurement
- Low cost: not much more hardware than described in the section 1.1
- Fast: a maximum of a few seconds of measurement and computation

Considering the mechanic part built-up and functional, the list above explains the tasks that should be done to make the system able to work properly:

- Controlling 3 servomotors and 3 LIDARs with a raspberry Pi, making them working together
- Study how to find the user navel's height
- Implementing the algorithm for user circumference measurement from a point cloud
- Integrating the work in a program to control the machine and send the result to a computer via a serial communication

2 Hardware implementation

This project uses a Raspberry Pi v3, 3 RpLidar A1, and 3 AX12 servomotors as described on the figure 3.

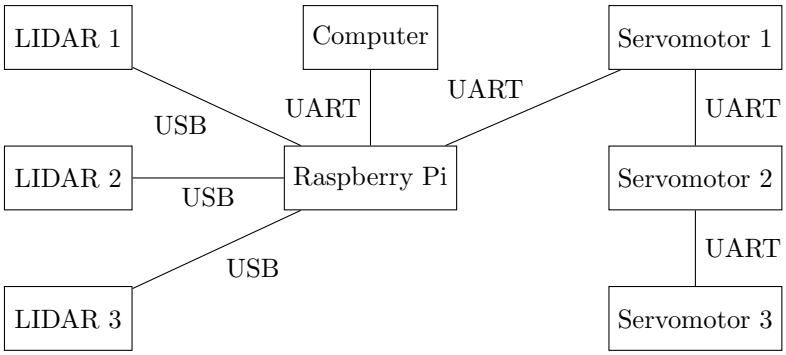


Figure 3: Basic architecture of the developed system

2.1 Raspberry Pi

The Raspi's GUI is reached through a VNC server, connected to Eduroam. To find its IP address, use *IP Finder*. A monitor can also be plugged to the HDMI port.

The program is integrally written in Python 3.6.

2.2 LIDARs

The open source library: `https://github.com/Roboticia/RPLidar` is used to control the LIDARs. While scanning, the 3 LIDARs gather the data alternately. They just need to be plugged to the Raspberry via USB.

2.3 Servomotors

The project uses the following open source library to control the servomotors: `https://pypi.org/project/pyax12/`.

The AX12 servomotors can be used in a daisy chain configuration. They have only 3 pins (2 for POWER and 1 for UART), which means that the RX/TX signals have to be multiplexed/demultiplexed in order to establish the communication. Thus, a simple circuit was designed as shown on the figure 4.

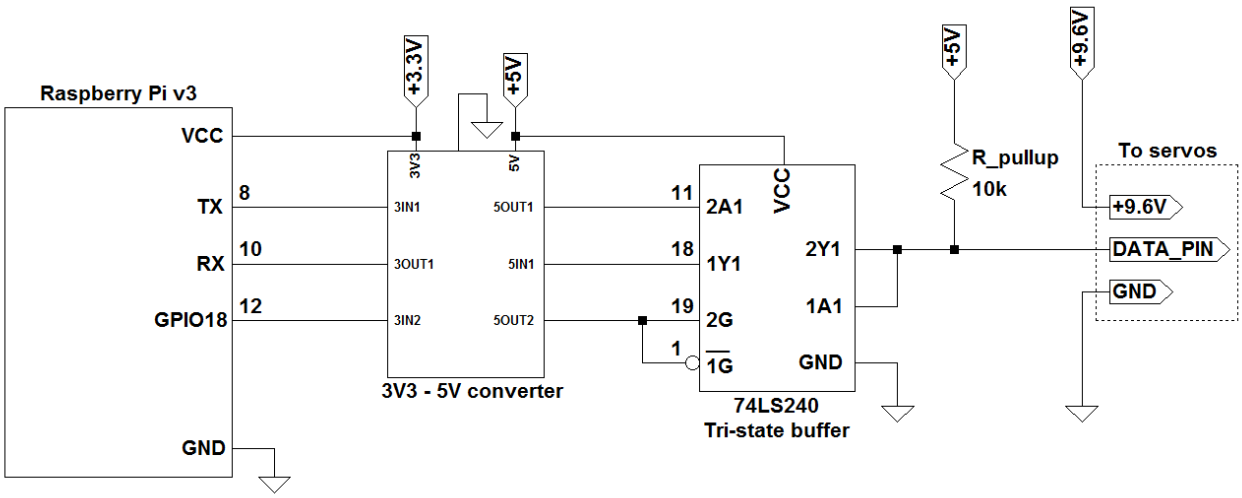


Figure 4: Electronic circuit diagram for the servomotors' control

Here, a 74LS240 tri-state buffer is used. It allows to control the direction of the communication with a single GPIO from the Raspi. When it is high, the Raspi can send data, when it is low, it receives the data.

To control the servomotors, few details need to be known:

- The ideal baudrate is 57600
- Each servo has an ID and they must all be different
- When they all run at the same time, an high current drain can occur. Thus, without a powerful generator and high value capacitance, the speed should be set slow.

To configure the servomotors, a script is available in the source. It allows to test a servomotor by changing its position, to display all the servos' information, to set the max angles values, to change the rotation speed, to set the IDs and to configure the baudrate to 57600 with a brute-force algorithm. The servomotors should be configured with this program before using it in the project.

3 Body scan

The following section describes how to find the navel's height of the user and the procedure to scan a specific level of the body.

3.1 Notations and physical quantities

Since now and until the end of this section, the norm an names described below will be used.

First, the units used are *millimeters* and *degrees*.

The figure 5 is a top view of the system. The origin of the system frame in blue is the middle of the base, represented by the circle. Since the lidar lie in a circle with D as radius, the angle $\{\beta_1,\ \beta_2,\ \beta_3\}$ are the angles of the LIDARs from the origin vector X. In our case:

$$\begin{bmatrix}\beta_1 & \beta_2 & \beta_3\end{bmatrix} = \begin{bmatrix}\frac{-\pi}{6} & \frac{-5\pi}{6} & \frac{-9\pi}{6}\end{bmatrix}$$

Each lidar has its own local frame represented in gold, and α is the angle between the Y vector of the local frame (oriented to the middle of the circle) and the measured laser beam. m is the distance between the laser beam source and the impact.

The red zone inside the circle is a non-measurable area. Indeed, the lidars need a minimum distance to be able to perform a measurement. Moreover, the more this red zone is large, the wider the range of reachable scan heights. d is the radius of the non-measurable area.

Note that in this case, $\alpha < 0$ and $\beta < 0$.

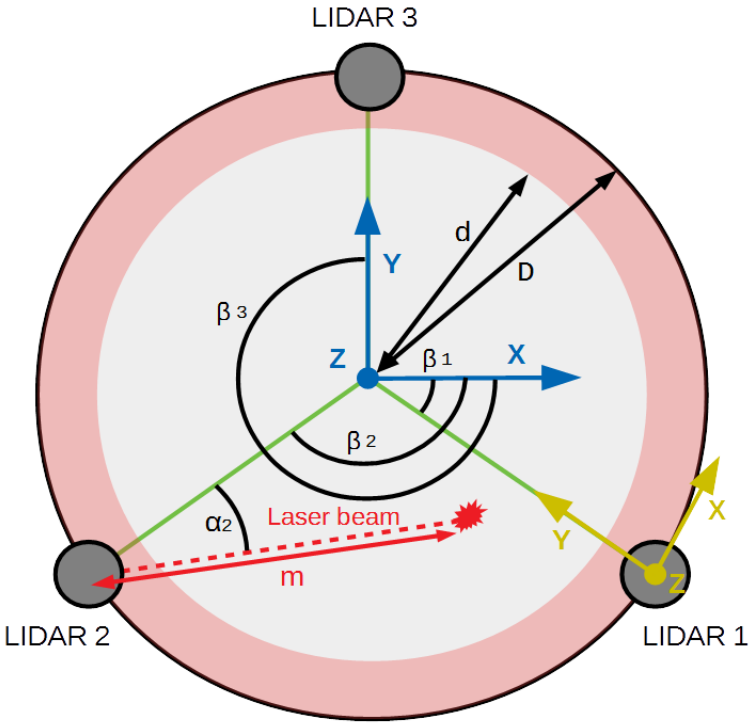


Figure 5: Top view of the system

The figure 6 shows a side view of the system. We can see that a servomotor is tilting the lidar and the angle between the zeroing (LIDAR in flat position) and the tilting is called θ. Note that again in this case, $\theta < 0$.

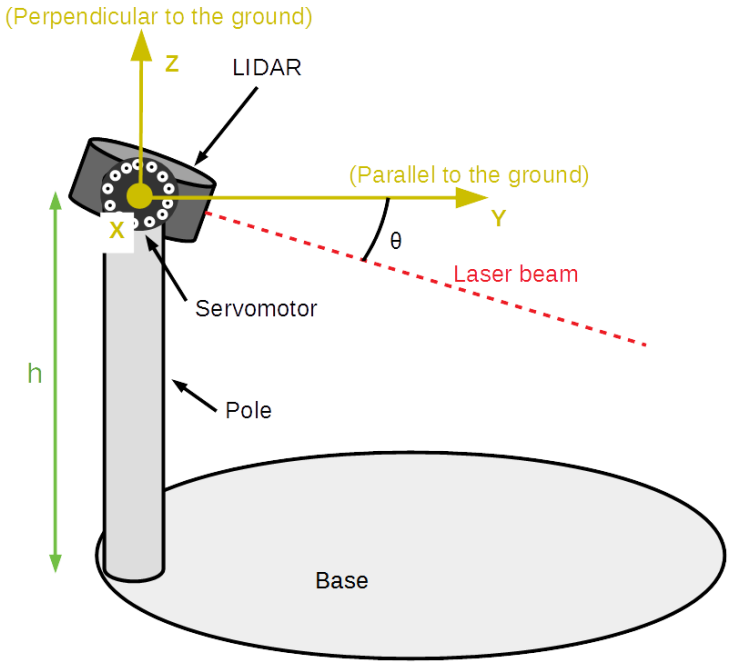


Figure 6: Side view of the system

Finally, h is the height of the lidar relative to the base. It is important because the origin of the altitude is at the base level.

3.2 Scanning procedure

Let's assume in this section that we want to perform a scan at an height $\tilde{h}$.

The figure 7 intuitively shows the main problem that occurs when the lidar is higher or lower than the desired part of the body to scan. If we assume the human body as a cylinder, we can see that the lidar will measure the body at different height instead of the one desired.

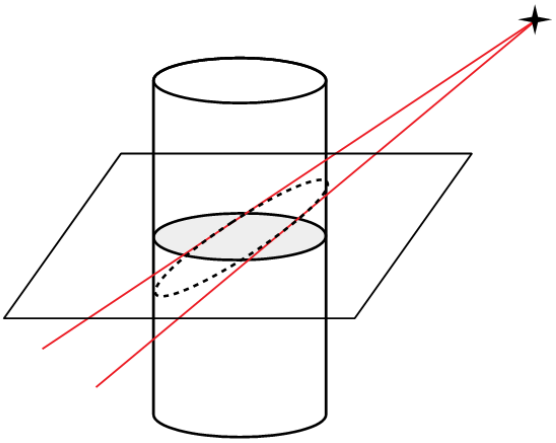


Figure 7: Cylinder cut from an higher source

To overpass this issue, a protocol has been implemented. It consists basically into tilting the lidar at many different angles, and to flatten the data measured close to the height $\tilde{h}$.

The figure 8 is a side view of the system, showing the lidar mounted on the pole on the left and the base at the bottom. The algorithm 1 describes the different steps of the scanning process.

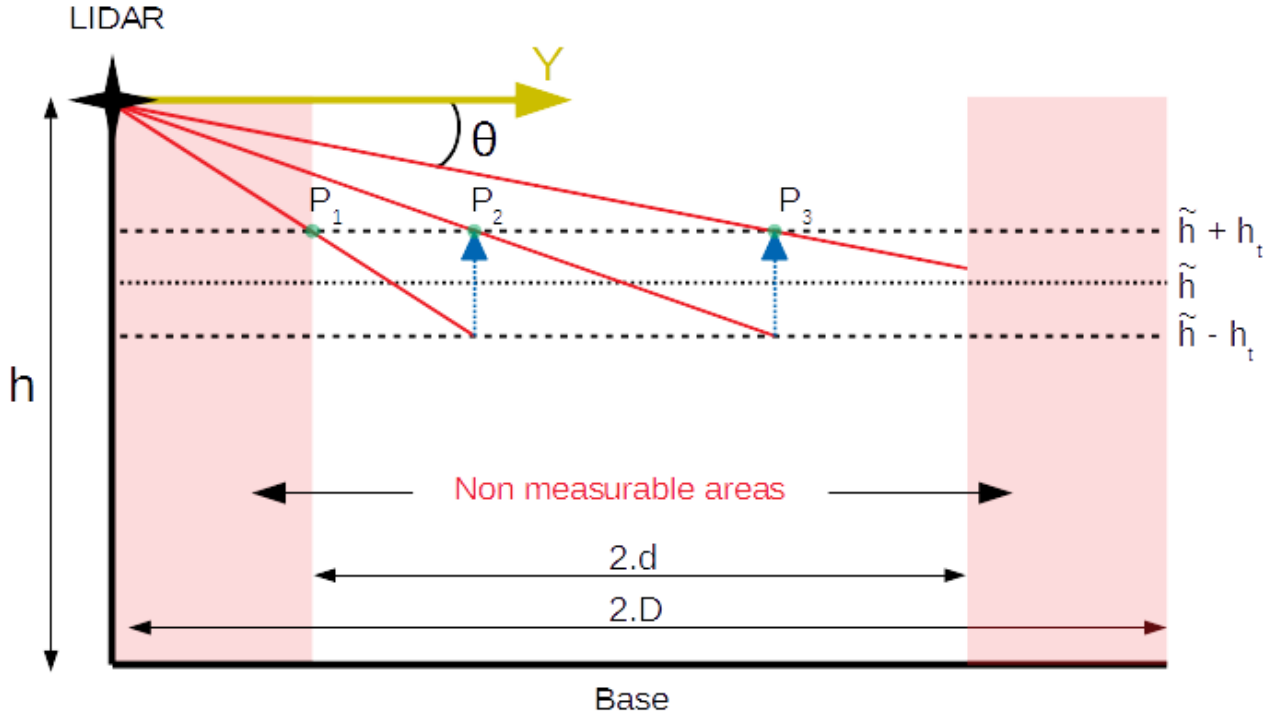


Figure 8: Side view of the system with scanning procedure explanation

The figure 8 shows a tolerance zone around the desired height measurement $\tilde{h}$ at levels $\tilde{h} + h_t$ and $\tilde{h} - h_t$. All this tolerance zone should be covered horizontally by the set of scans. By reading the figure 8, the process is as follows:

1. Compute P_1, the intersect of the tolerance surface $\tilde{h} + h_t$ vertically and the beginning of the measurable area horizontally.
2. Compute the intersection of a laser beam passing through P_1 which would intersect the inferior tolerance surface at level $\tilde{h} - h_t$.
3. Vertically project the point crossing the laser beam and the inferior tolerance surface on the superior tolerance surface to get P_2.
4. Repeat the process from point P_2 since the step 2 while $P_n < 2.D - (D - d) = D + d$.
5. Scan the body with the angle corresponding to make the laser beam passing through the points P_i found.

Algorithm 1 Pseudo code of the scanning procedure

Input: $h,\ \tilde{h},\ h_t \in \mathbb{R}^+$
Output: θ_{arr} *// Array that represents the pattern of angles to follow sequentially*

$\theta_{arr} \leftarrow [\ ]$
if $h \leq \tilde{h} + h_t$ *and* $h \geq \tilde{h} - h_t$ **then**
  return [0]
end if
if $h \geq \tilde{h} + h_t$ **then**
  $h_1 \leftarrow \tilde{h} + h_t - h$, $h_2 \leftarrow \tilde{h} - h_t - h$
else
  $h_1 \leftarrow \tilde{h} - h_t - h$, $h_2 \leftarrow \tilde{h} + h_t - h$
end if
$d_1,\ d_2 \leftarrow D - d$
while $d_2 < d + D$ **do**
  $new\_theta \leftarrow Atan2(h_1, d_1)$
  $\theta_{arr}.append(new\_theta)$
  $d_2 \leftarrow \dfrac{h_2}{Tan(new\_theta)}$
end while
return θ_{arr}

3.3 Finding the user navel's height

Two different approach could be used to find the height of the user's navel: scanning the whole or a part of his body and detecting it, or estimating it through a statistical analysis.

3.3.1 Detecting the navel's height

Before implementing the detection, we need to know the degree of accuracy of our sensors and actuators. First, the datasheets of the LIDARs and servomotors give:

- Servomotors' resolution: $\frac{330}{1024}^\circ = 0.322^\circ$
- LIDARs' distance resolution: $< 0.5\ mm$
- LIDARs' angular resolution: $\leq 1^\circ$

We will consider the uncertainty of measurement of the Servomotors' position and the LIDARs' distance as negligible. However, let's compute the uncertainty u of the LIDARs' angle measurement:

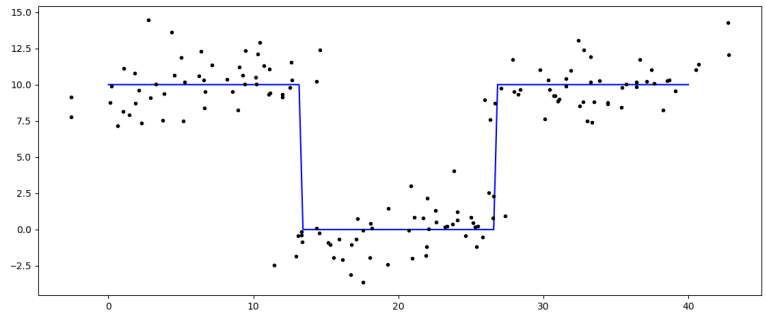
$$u = \frac{angular\_resolution/2}{\sqrt{3}}\ deg = \frac{0.5}{\sqrt{3}}\ deg \tag{1}$$

We choose to divide the half of the angular resolution by $\sqrt{3}$ as the measurement is likely made by a rotary encoder, giving measurements following an uniform probability distribution.

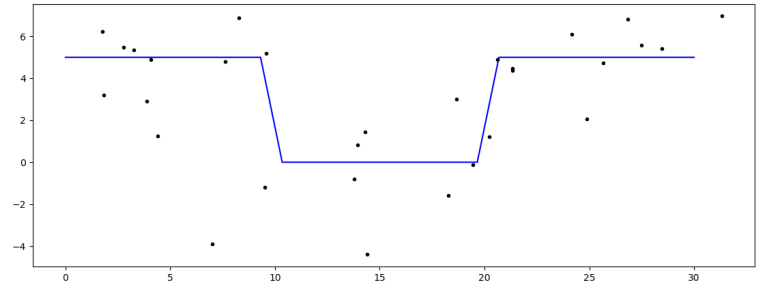
Let $v = [m, 0]$ be the measured vector in 2D space with m the measured distance by the LIDAR, the uncertainty of position u_p is given by:

$$u_p = \sqrt{\big[m - m.cos(u)\big]^2 + \big[-m.sin(u)\big]^2} \tag{2}$$

Then, the expanded uncertainty $U_p = k.u_p$ with k the coverage factor. $k = 1.96$ for a level of confidence of 95%. Assuming for example, that the user is 500mm away from the LIDAR ($m = 500$), then $U_p = 5.0mm$. Simulation have been made and are shown on the figure 9.



(a) Favourable conditions (width=10mm, depth=10mm, 150 data points)



(b) Bad conditions (width=5mm, depth=5mm, 30 data points)

Figure 9: Simulations of the user's navel detection

The main problems with detecting the users' navel are cited below:

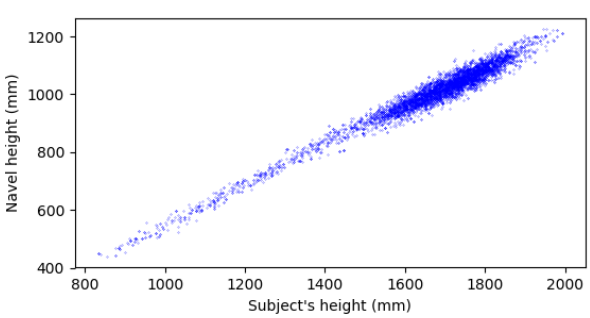
- The characteristics of each navel are highly different for each person
- If the navel is too small or not depth enough, the data become too noisy
- It required a lot of time to scan a part of the body with LIDAR
- If the patient is short or tall compared to the pole, then the navel is not aligned horizontally with the LIDAR
- It can cause some user inconvenience because he must not move at all and must not inflate his waist while breathing during the detection

For these reasons, the detection algorithm has not been implemented

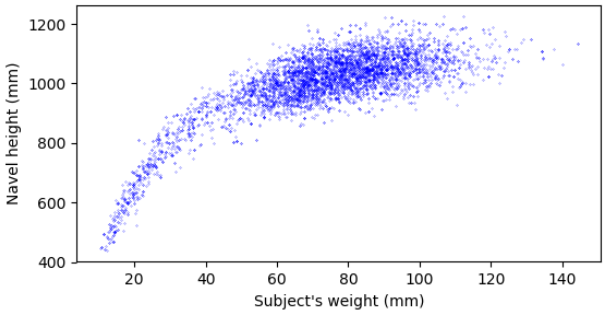
3.3.2 Estimating the navel's height

A estimation of the navel's height was tried, based on other anthropomorphic parameters such as stature, weight and age. Two open databases have been used: one for children from the *United States Consumer Product Safety Commission* [2] and another from the *United States Army* [3] for adults. They both gathered data from both genders.

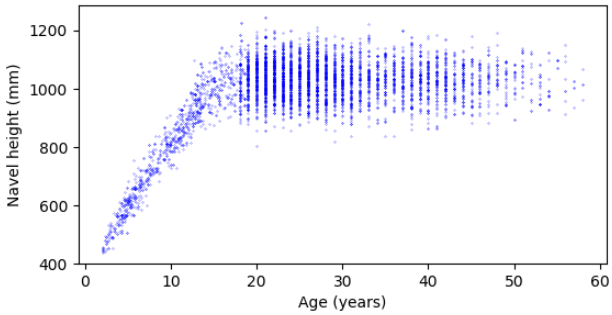
The figure 10 shows the thousands of measurement and the height of the navel depending of the patients' stature (height), their weight and age.



(a) Navel's height depending on patient's stature



(b) Navel's height depending on patient's weight



(c) Navel's height depending on patient's age

Figure 10: Patient navel's height depending on their stature, weight and age.

Intuitively, we can notice a correlation between the navel's height and each of the parameters. If we compute the correlation coefficient r described in the equation 3, we get the table 1.

$$r = \frac{Cov(X, Y)}{\sigma_X \sigma_Y} \tag{3}$$

X and Y are the variables and σ_X and σ_Y their standard deviation. Moreover $r \in [-1, 1]$.

Correlation (%)	Stature	Weight	Age
Navel's height	98%	79%	50%
Stature	/	84%	54%
Weight	/	/	61%

Table 1: Correlation of various anthropomorphic data among each others

As the 4 variables are all pretty well dependant to each other, a multi-layered perceptron has been tested (provided by the *sickit-learn* library) to estimate the navel's height. Moreover, as the correlation coefficients of the navel's height with the weight and stature is higher, a 2D linear regression has been implemented. Finally, a simple linear regression between the navel's height and stature has been tried.

By splitting the set of data into two groups, we can use a first part to train the regression while the other part is used to test the regression.

The error $\epsilon_{\%}$ of the estimated navel's height for each test data is computed as follow:

$$\epsilon_{\%} = \frac{|\text{estimated - true}|}{\text{stature}}.100 \tag{4}$$

The error is thus given as a portion of the patient's stature. The errors found for each 3 methods are given below:

- Multi-layer perceptron: 1.21%
- 2D linear regression stature/weight: 1.04%
- Linear regression on the stature: 0.94%

As the simple linear regression shows the best results, we use this method and the error terms of a set of test are shown in figure 11 in a distribution graph.

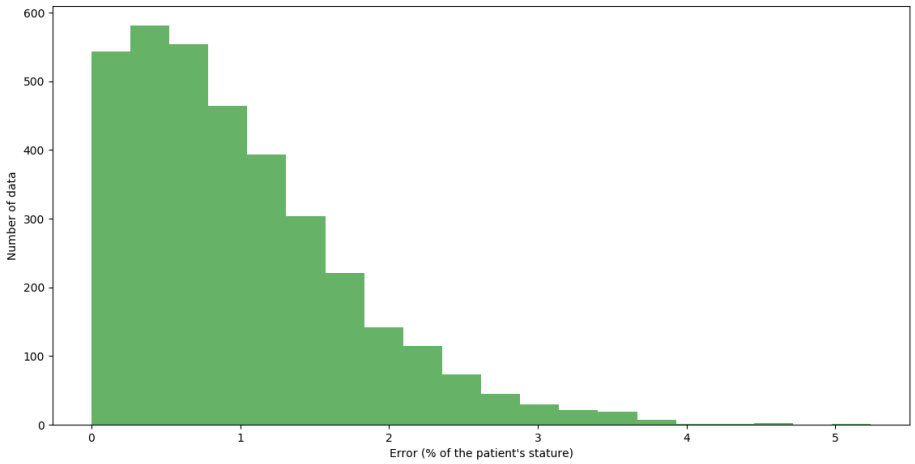


Figure 11: Distribution of the estimated navel height's error

All the results are presented in the table 2 and are interpolated for people 140cm, 160cm and 180cm tall:

Error:	Fraction of the stature	140cm tall patients	160cm tall patients	180cm tall patients
Mean	0.94%	13.16mm	15.04mm	16.92mm
Median	0.83%	11.62mm	13.28mm	14.94mm
75-percentile	1.39%	19.46mm	22.24mm	25.02mm
95-percentile	2.40%	38.40mm	38.40mm	43.20mm
Max error	5.24%	73.36mm	83.84mm	94.32mm

Table 2: Error of the estimated navel's height

Note that the databases used are not sampling a good representation of the population (soldiers and American children). Some other sets of data more adapted to global population may be found.

3.4 Running a scan

The 3 LIDARs have to scan in parallel. However, only one thread is running at the same time. It means that the program gathers the measurements one by one on each LIDAR. Moreover, the servomotors are all moving at the same time to avoid too much distortions if the users moves during the scan. Once the Raspi has yielded the measurements, it saves them into text files containing the angle of rotation of the lidar α, the angle of tilting of the servomotors θ and the distance from the measured object m, giving a file containing:

$$\begin{bmatrix} \alpha_j & \theta_j & m_j \end{bmatrix}_i$$

with i number of LIDARs and j number of measurements. Another file contains the scan's information (for now, only the height $\tilde{h}$).

3.5 Raw data to point cloud

Once the scan is over, 3 text files and 1 info file have been generated. The goal now is to locate every measurement in the 3D space.

Each measurement is a 3D vector, having the LIDAR as origin. Let's find the orientation of the vector for each measurement of the LIDAR i. m is the norm of the vector, and is the distance measured by the LIDAR. In the LIDAR's frame as shown in the figure 5 and figure 6, let's assume the vector $v_{0,j} = \begin{bmatrix} 0 & m_j & 0 \end{bmatrix}$, which mean a vector having the LIDAR as origin and pointing forward.

Now we want to rotate it with the rotation matrix given in the Slabaugh's article [4]. The elemental rotation matrices around the axis X and Z can be expressed as follow:

$$R_X(\theta_j) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_j) & -sin(\theta_j) \\ 0 & sin(\theta_j) & cos(\theta_j) \end{bmatrix} \tag{5}$$

$$R_Z(\alpha_j) = \begin{bmatrix} cos(\alpha_j) & -sin(\alpha_j) & 0 \\ sin(\alpha_j) & cos(\alpha_j) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Moreover, we need to get the final vector in the system's frame. To transform the LIDAR's frame into the system's frame, we will rotate the around Z-axis by $+90°$ (to match the X and Y axis), then rotate it around the X-axis by an angle Θ and rotate it again around the Z-axis by the value of its position angle β_i.

Thus, the final rotation matrix can be written by combining 5 and 6:

$$R_{final}(\beta_i, \theta_j, \alpha_j) = R_Z(\beta_i + 90°).R_X(\theta_j).R_Z(\alpha_j) = \begin{bmatrix} c_1c_3 - c_2s_1s_3 & -c_1s_3 - c_2c_3s_1 & s_1s_2 \\ c_3s_1 + c_1c_2s_3 & c_1c_2c_3 - s_1s_3 & -c_1s_2 \\ s_2s_3 & c_3s_2 & c_2 \end{bmatrix} \tag{7}$$

For readability purpose, c means cos and s means sin, while 1 correspond to $\beta_i + 90°$, 2 to θ_j and 3 to α_j. For instance, $c2$ means $cos(\theta_j)$.

We obtain:

$$v_j^T = R_{final}(\beta_i, \theta_j, \alpha_j).v_{0,j}^T = m_j. \begin{bmatrix} -c_1s_3 - c_2c_3s_1 \\ c_1c_2c_3 - s_1s_3 \\ c_3s_2 \end{bmatrix} \tag{8}$$

So, v_j is the vector in the system frame having the system origin as starting point. A simple shift can be apply to it depending on the position of the LIDAR. The point P_j is the 3D measurement j made by the LIDAR i:

$$P_j = v_{0,j}^T + \begin{bmatrix} D.cos(\beta_i) \\ D.sin(\beta_i) \\ h \end{bmatrix} \tag{9}$$

It could be enough in theory but practically, the system looks more like presented on figure 12. When the LIDAR is tilted, its position changes.

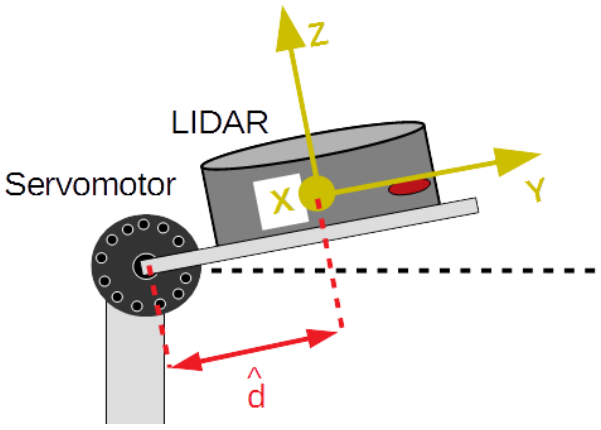


Figure 12: Mechanism of the LIDAR tilting of the prototype

Let's add to our equation 9 a position correction with $\hat{d}$ the distance between the center of rotation of the servomotor and the LIDAR.

$$P_j = v_{0,j}^T + \begin{bmatrix} D.cos(\beta_i) \\ D.sin(\beta_i) \\ h \end{bmatrix} + \begin{bmatrix} \hat{d}.cos(\beta_i).(1 - cos(\theta_j)) \\ \hat{d}.sin(\beta_i).(1 - cos(\theta_j)) \\ \hat{d}.sin(\theta_j) \end{bmatrix} \tag{10}$$

The figure 13 shows an example of scanned 3D cloud.

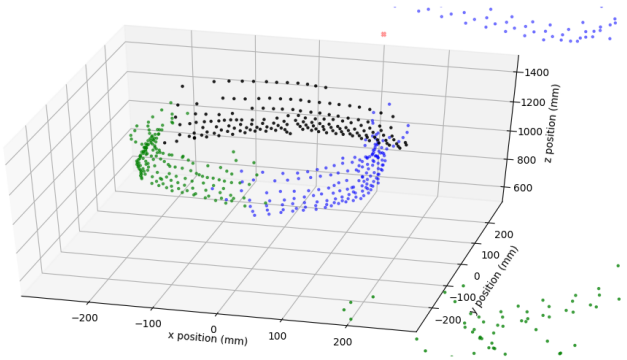


Figure 13: Example of a 3D point cloud computed. Each color corresponds to a different LIDAR.

4 Contour reconstruction

After flattening the points around $\tilde{h}$, we get a 2D point cloud. The goal now is to compute a smooth curve that fits the data points.

4.1 Presentation of the method

The method chosen here is inspired by the Wang et al's paper [5] which presents a method called *Squared Distance Minimization* for computing a closed planar B-Spline curve that reconstructs the shape of a point cloud of non-ordered data.

A B-spline curve is a specific combination of splines which are themselves, functions defined piecewise by polynomials. The optimization process is a non linear least square problem and will be performed by minimizing an objective function.

Contrary to the LOESS based reconstruction that was first implemented in this project and presented in the Vincent MAIRE's report [1], this method has been chosen mostly because it is well documented and many papers discuss about it. Thus, some improvements can be made more easily in the future.

4.2 Sum up on B-spline

A B-spline curve can be written:

$$P(t) = \sum_{i=1}^{m} B_{i,n}(t) P_i \tag{11}$$

with t the value of the knot, m the number of control points, $B_{i,n}(t)$ the basis element of order n at the knot t and $P_i(t)$ the position of the control point i.

A B-spline has an order, which defines how far a node will affect another from. In our case, we will use an order of 3 as it is the most common used and the author of the paper does not provide any further details about this.

The B-spline is also set by a knot vector, which gives a value to each knot at each control point. A control point is the coefficient that, to sum up the equation 11, represents the position of the B-spline. If we consider a zero order B-spline, then the curve at the knot value t_i will exactly pass through the value of P_i.

A B-spline basis element is a recursive function defined by the equation:

$$\begin{aligned} B_{i,0}(t) &= \begin{cases} 1 & if\ t_i \leq t < t_{i+1} \\ 0 & otherwise \end{cases} \\ B_{i,n}(t) &= \frac{t - t_i}{t_{i+n} - t_i} B_{i,n-1}(t) + \frac{t_{i+n+1} - t}{t_{i+n+1} - t_{i+1}} B_{i+1,n-1}(t) \end{aligned} \tag{12}$$

4.3 Method's steps

We assume that the number of control points and the order of the spline will stay the same. This method consists in translating the control points at every step.

The first step requires to find a good initial B-spline curve which fits with the data, since this method is highly dependent to the initial conditions.

The fitting B-spline can be written from the equation 11: $P_c(t) = \sum_{i=1}^{m} B_i(t) P_{c,i}$. Let P_+ be the updated B-spline after moving the control points as $P_+ = P_c + D$. Then, we will define as a *footpoint* the closest point $P_c(t_k)$ of the curve to the data point X_k. T_k, N_k and ρ are respectively the tangent unit vector, the normal unit vector and the curvature radius of the curve $P_c(t)$ at the point $P_c(t_k)$. d is the signed distance as $|d| = \left\| P_c(t_k) - X_k \right\|$ and its sign is chosen following this rule: $d \geq 0$ if X_k is on the same side of the curve curvature center at the point $P_c(t_k)$, $d < 0$ otherwise.

For each iteration of the algorithm, every of the described above characteristics must be computed. Now, let's define the error term named *Squared Distance* as:

$$e_{SD,k}(D) = \begin{cases} \dfrac{d}{d-\rho}[(P_+(t_k) - X_k)^T T_k]^2 + [(P_+(t_k) - X_k)^T N_k]^2 & if\ d < 0 \\ \left[(P_+(t_k) - X_k)^T N_k\right]^2 & if\ 0 \leq d < \rho \end{cases} \tag{13}$$

By computing at each iteration the Squared Distance error term for every data point, we obtain the f_{SD} term that we will minimize as:

$$f_{SD} = \frac{1}{2} \sum_k e_{SD,k} + \lambda f_s \tag{14}$$

Here, f_s is a regularization term to improve the stability and λ is a pondering coefficient. The f_{SD} is quadratic and thus, can be minimized by solving a linear system of equations. Then an incremental change of D to the initial control points is applied.

The process can be iterated until a user custom error term is below a threshold or if the incremental change D becomes small enough.

4.4 Implementation

This section describes how this method has been implemented for our system.

4.4.1 B-spline computing

Actually, the program is written in Python, which provides libraries for B-spline computing (through *Scipy*). Two are particularly interesting and their characteristics are presented in the table 5. In this project, these two functions have been used.

scipy.interpolate.BSpline	• Create a 1D open B-spline function object from the knots, control points and order • Evaluate the value, the derivative and integral of the B-spline at t • Return the basis element of the B-spline at t
scipy.interpolate.splprep	• Create a 2D open or closed interpolation B-spline object from an ordered list of points

Table 3: Two interesting B-spline computing libraries provided by Scipy

To compute a B-spline, **scipy.interpolate.BSpline** is called twice, in order to generate a B-spline on both x and y axis. Then the values on both axis are evaluated independently.

Then, to make the B-spline periodic as we want to reconstruct a closed shape, it is necessary to append the n first knot values and coefficients to the B-spline knot values and coefficient. That way, the B-spline will repeat its last values as if it was closed.

4.4.2 Finding the initial spline

As said in the section 4.3, an initial B-spline must be found before running the algorithm. The chosen implementation here is a circle fitting, since the human shape is quite circular and symmetric about the two axis.

The least square fitting method as described in the Dumbach et al 's paper [6] is used, computing the radius and position with the equations 15.

$$
\begin{aligned}
A &= n\sum_{j=1}^{n} x_j^2 - \left(\sum_{j=1}^{n} x_j\right)^2 \\
B &= n\sum_{j=1}^{n} x_j y_j - \left(\sum_{j=1}^{n} x_j\right)\left(\sum_{j=1}^{n} y_j\right) \\
C &= n\sum_{j=1}^{n} y_j^2 - \left(\sum_{j=1}^{n} y_j\right)^2 \\
D &= 0.5\left\{n\sum_{j=1}^{n} x_j y_j^2\left(\sum_{j=1}^{n} x_j\right)\left(\sum_{j=1}^{n} y_j^2\right) + n\sum_{j=1}^{n} x_j^3 - \left(\sum_{j=1}^{n} x_j\right)\left(\sum_{j=1}^{n} x_j^2\right)\right\} \\
E &= 0.5\left\{n\sum_{j=1}^{n} y_j x_j^2\left(\sum_{j=1}^{n} y_j\right)\left(\sum_{j=1}^{n} x_j^2\right) + n\sum_{j=1}^{n} y_j^3 - \left(\sum_{j=1}^{n} y_j\right)\left(\sum_{j=1}^{n} y_j^2\right)\right\}
\end{aligned}
\tag{15}
$$

$$
\begin{aligned}
a_M &= \frac{DC - BE}{AC - B^2} \\
b_M &= \frac{AE - BD}{AC - B^2} \\
r_M &= \frac{1}{n}\sum_{j=1}^{n} \sqrt{(x_j - a_M)^2 + (y_j - b_M)^2}
\end{aligned}
$$

With x_j and y_j the coordinates of the j-th point, a_M and b_M the position of the fitted circle's center and r_M its radius.

Now that we have a circle that barely fits with the point cloud, we split it into n areas, the number of control points of the B-spline, equal sections as shown in the figure 14. Let's assume moreover that it creates areas with infinite size as they grow even outside the circle.

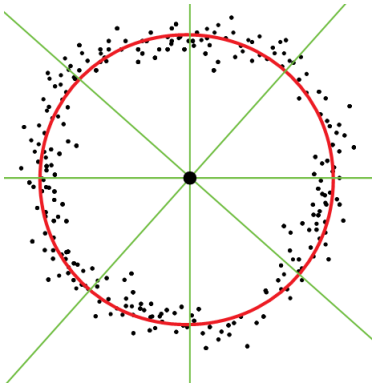
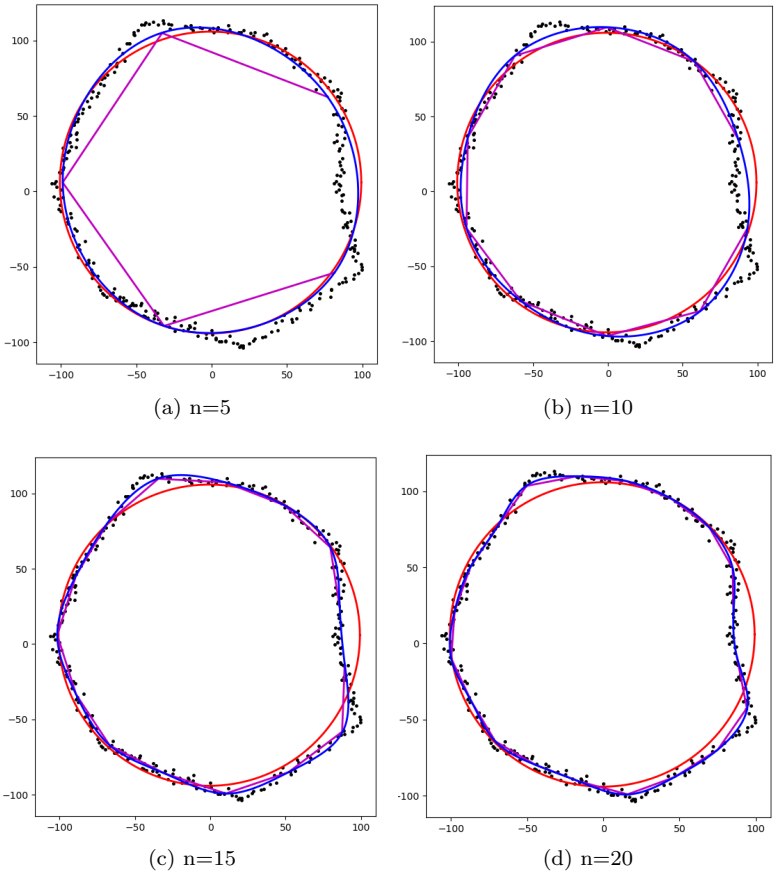


Figure 14: How to split the fitting circle. Example with 8 control points.

Then, the median of the distance of the points to the center is computed for each area, taking into account the points that lie in the concerned area. We use the median as it is more robust to outliers than averaging. Finally, since we need the value of the first control points, the function *scipy.interpolate.splprep* is called to create an interpolation B-spline that fits the computed points. The control points of the initial B-spline are then extracted from this previous B-spline.

In the case an area does not include any points, then we compute the average of the median of the two surrounding zones. If the surrounding zones do not include any points too, a linear interpolation is made between the two closest zone that contain points.

This method brought good results and becomes even more accurate when the number of control points increases. Some examples of the results for several number of control points n are shown in figure 15.



(a) n=5 (b) n=10

(c) n=15 (d) n=20

Figure 15: Initial B-spline results
In red, the fitted circle, in magenta the distance median for each area and in blue the initial B-spline found.

4.4.3 Point's parameters computing

Many different parameters should be computed for each point of each iteration as presented in the section 4.3.

First, the footpoint which is the closest point $P_c(t_k)$ of the curve to the data point X_k must be computed. Many different methods can be implemented. In our case, the following process is applied:

1. Evaluating the B-spline with a large number of point (here 50 points per control node)
2. Computing the distance of each point to each point of the evaluated curve
3. Finding the minimum distance and returning t_k the knot value of the footpoint $P_c(t_k)$

Then the tangent and normal vector can be recovered by derivating respectively once and twice the B-spline and dividing each x and y parameters by the vector's norm to make it unit.

The radius ρ of the curve at t_k is the norm of the normal vector N_k.

To find the signed distance d, we compute $k = (X_k - P_c(t_k))^T N_k$. If the parameter of $k = [k_x, k_y]$ with the greatest value is negative, then $d < 0$, else $d \geq 0$.

4.4.4 Minimizing the objective function

The function f_{SD} described in the equation 14 must be minimized in order to find the optimal incremental update D. The implementation of the algorithm is inspired from the work of Pekelny [7] and Mhala [8].

As shown in Wang et al 's paper [5], f_{SD} is positive quadratic in D_i, which means that it can be minimized solving $\frac{\partial f_{SD}}{\partial D} = 0$. It can be written in the form $Ax = b$ as follow:

$$\begin{bmatrix} \left(\frac{\partial f_{SD}}{\partial D_{1,x}}\right)_{1,x} & \cdots & \left(\frac{\partial f_{SD}}{\partial D_{1,x}}\right)_{n,x} & \left(\frac{\partial f_{SD}}{\partial D_{1,x}}\right)_{1,y} & \cdots & \left(\frac{\partial f_{SD}}{\partial D_{1,x}}\right)_{n,y} \\ & & \vdots & & & \\ \left(\frac{\partial f_{SD}}{\partial D_{n,x}}\right)_{1,x} & \cdots & \left(\frac{\partial f_{SD}}{\partial D_{n,x}}\right)_{n,x} & \left(\frac{\partial f_{SD}}{\partial D_{n,x}}\right)_{1,y} & \cdots & \left(\frac{\partial f_{SD}}{\partial D_{n,x}}\right)_{n,y} \\ \left(\frac{\partial f_{SD}}{\partial D_{1,y}}\right)_{1,x} & \cdots & \left(\frac{\partial f_{SD}}{\partial D_{1,y}}\right)_{n,x} & \left(\frac{\partial f_{SD}}{\partial D_{1,y}}\right)_{1,y} & \cdots & \left(\frac{\partial f_{SD}}{\partial D_{1,y}}\right)_{n,y} \\ & & \vdots & & & \\ \left(\frac{\partial f_{SD}}{\partial D_{n,y}}\right)_{1,x} & \cdots & \left(\frac{\partial f_{SD}}{\partial D_{n,y}}\right)_{n,x} & \left(\frac{\partial f_{SD}}{\partial D_{n,y}}\right)_{1,y} & \cdots & \left(\frac{\partial f_{SD}}{\partial D_{n,y}}\right)_{n,y} \end{bmatrix}_{2n,2n} \begin{bmatrix} D_{1,x} \\ \vdots \\ D_{n,x} \\ D_{1,y} \\ \vdots \\ D_{n,y} \end{bmatrix}_{1,2n} = \begin{bmatrix} b_{1,x} \\ \vdots \\ b_{n,x} \\ b_{1,y} \\ \vdots \\ b_{n,y} \end{bmatrix}_{1,2n} \tag{16}$$

The left matrix is A, the middle is x and the right is b, with n the number of control points, D the incremental update and b the constant term of the derivative. Each term $\left(\frac{\partial f_{SD}}{\partial D_{i,x/y}}\right)_{j,x/y}$ must be computed, which is the coefficient of $D_{j,x/y}$ when f_{SD} is derivated by $D_{i,x/y}$.

With the equation 14, we see that f_{SD} is a combination of the error term e_{SD} and the regularization term f_s. First, we do not consider the regularization term, thus:

$$\frac{\partial f_{SD}}{\partial D_{i,x/y}} = \frac{1}{2}\sum_k \frac{\partial e_{SD,k}}{\partial D_{i,x/y}} \tag{17}$$

When $0 \leq d < \rho$, with the formula of e_{SD} defined in the equation 13:

$$\begin{aligned} e_{SD,k} &= [(P_+(t_k)) - X_k)^T N_k]^2 \\ \frac{\partial e_{SD,k}}{\partial D_{i,x}} &= \frac{\partial [(P_+(t_k)) - X_k)^T N_k]^2}{\partial D_{i,x}} \\ &= 2\beta_i(t_k) N_{k,x}[(P_+(t_k) - X_k)^T N_k] \\ &= 2\beta_i(t_k) N_{k,x}\left[\left(\sum_{j=1}^{n} \beta_j(t_k) D_j\right)^T N_k\right] + 2\beta_i(t_k) N_{k,x}[(P(t_k) - X_k)^T N_k] \\ &= 2\beta_i(t_k) N_{k,x}^2 \sum_{j=1}^{n} \beta_j(t_k) D_{j,x} + 2\beta_i(t_k) N_{k,x} N_{k,y} \sum_{j=1}^{n} \beta_j(t_k) D_{j,y} + 2\beta_i(t_k) N_{k,x}[(P(t_k) - X_k)^T N_k] \end{aligned} \tag{18}$$

In the same way as the equation 18, we can find $\frac{\partial e_{SD,k}}{\partial D_{i,y}}$ when $0 \leq d < \rho$:

$$\frac{\partial e_{SD,k}}{\partial D_{i,y}} = 2\beta_i(t_k) N_{k,y}^2 \sum_{j=1}^{n} \beta_j(t_k) D_{j,y} + 2\beta_i(t_k) N_{k,x} N_{k,y} \sum_{j=1}^{n} \beta_j(t_k) D_{j,x} + 2\beta_i(t_k) N_{k,y}[(P(t_k) - X_k)^T N_k] \tag{19}$$

Now, when $d < 0$, the error term is very similar as it is the sum of the error term when $0 \leq d$ and itself when N is replaced by T and multiply $\frac{d}{d-\rho}$. Thus, when $d < 0$, we have:

$$
\begin{aligned}
\frac{\partial e_{SD,k}}{\partial D_{i,x}} &= 2\beta_i(t_k)N_{k,x}^2\sum_{j=1}^{n}\beta_j(t_k)D_{j,x} + 2\beta_i(t_k)N_{k,x}N_{k,y}\sum_{j=1}^{n}\beta_j(t_k)D_{j,y} \\
&+ 2\beta_i(t_k)N_{k,x}[(P(t_k)-X_k)^T N_k] \\
&+ 2\beta_i(t_k)\frac{d}{d-\rho}T_{k,x}^2\sum_{j=1}^{n}\beta_j(t_k)D_{j,x} + 2\beta_i(t_k)\frac{d}{d-\rho}T_{k,x}T_{k,y}\sum_{j=1}^{n}\beta_j(t_k)D_{j,y} \\
&+ 2\beta_i(t_k)\frac{d}{d-\rho}T_{k,x}[(P(t_k)-X_k)^T T_k] \\
\frac{\partial e_{SD,k}}{\partial D_{i,y}} &= 2\beta_i(t_k)N_{k,y}^2\sum_{j=1}^{n}\beta_j(t_k)D_{j,y} + 2\beta_i(t_k)N_{k,x}N_{k,y}\sum_{j=1}^{n}\beta_j(t_k)D_{j,x} \\
&+ 2\beta_i(t_k)N_{k,y}[(P(t_k)-X_k)^T N_k] \\
&+ 2\beta_i(t_k)\frac{d}{d-\rho}T_{k,y}^2\sum_{j=1}^{n}\beta_j(t_k)D_{j,y} + 2\beta_i(t_k)\frac{d}{d-\rho}T_{k,x}T_{k,y}\sum_{j=1}^{n}\beta_j(t_k)D_{j,x} \\
&+ 2\beta_i(t_k)\frac{d}{d-\rho}T_{k,y}[(P(t_k)-X_k)^T T_k]
\end{aligned}
\tag{20}
$$

From the equations 18, 19 and 20, we can extract the coefficients as follow:

$0 \le d < \rho$	**Derivative by** $D_{i,x}$	**Derivative by** $D_{i,y}$
Coefficient $D_{j,x}$	$2\beta_i(t_k)\beta_j(t_k)N_{k,x}^2$	$2\beta_i(t_k)\beta_j(t_k)N_{k,x}N_{k,y}$
Coefficient $D_{j,y}$	$2\beta_i(t_k)\beta_j(t_k)N_{k,x}N_{k,y}$	$2\beta_i(t_k)\beta_j(t_k)N_{k,y}^2$
Constant b_j	$2\beta_i(t_k)N_{k,x}[(P(t_k)-X_k)^T N_k]$	$2\beta_i(t_k)N_{k,y}[(P(t_k)-X_k)^T N_k]$

Table 4: Coefficients $D_{j,x/y}$ of derivative by $D_{i,x/y}$ when $0 \le d < \rho$

$d < 0$	**Derivative by** $D_{i,x}$	**Derivative by** $D_{i,y}$
Coefficient $D_{j,x}$	$2\beta_i(t_k)\beta_j(t_k)N_{k,x}^2$ $+2\frac{d}{d-\rho}\beta_i(t_k)\beta_j(t_k)T_{k,x}^2$	$2\beta_i(t_k)\beta_j(t_k)N_{k,x}N_{k,y}$ $+2\frac{d}{d-\rho}\beta_i(t_k)\beta_j(t_k)T_{k,x}T_{k,y}$
Coefficient $D_{j,y}$	$2\beta_i(t_k)\beta_j(t_k)N_{k,x}N_{k,y}$ $+2\frac{d}{d-\rho}\beta_i(t_k)\beta_j(t_k)T_{k,x}T_{k,y}$	$2\beta_i(t_k)\beta_j(t_k)N_{k,y}^2$ $+2\frac{d}{d-\rho}\beta_i(t_k)\beta_j(t_k)T_{k,y}^2$
Constant b_j	$2\beta_i(t_k)N_{k,x}[(P(t_k)-X_k)^T N_k]$ $+2\frac{d}{d-\rho}\beta_i(t_k)T_{k,x}[(P(t_k)-X_k)^T T_k]$	$2\beta_i(t_k)N_{k,y}[(P(t_k)-X_k)^T N_k]$ $+2\frac{d}{d-\rho}\beta_i(t_k)T_{k,y}[(P(t_k)-X_k)^T T_k]$

Table 5: Coefficients $D_{j,x/y}$ of derivative by $D_{i,x/y}$ when $d < 0$

It is now possible to fill a matrix for each point with those coefficient, then summing them all.

We can now consider the regularization term. It is used to smooth the curve. The Wang et al 's paper [5] gives two regularization functions that can be used:

$$\begin{aligned} F_1 &= \int \left\| P^{'}(t) \right\|^2 dt \\ F_2 &= \int \left\| P^{''}(t) \right\|^2 dt \end{aligned} \tag{21}$$

However, these functions are computationally very expensive. A simpler regularization term could be to align each control point with its closest neighbors. This equals to minimize the distance between each control points and the middle m_i of the segment which links its two closest neighbors.

$$f_s = \sum_{i=0}^{n} \|m_i - P_+\|^2 \tag{22}$$

The norm is squared in f_s in order to make it quadratic and to minimize it easily. We need to compute $min\ f_s$, which can be performed by solving $\dfrac{\partial f_s}{\partial D} = 0$. Moreover, the middle of the segment which links P_{i-1} and P_{i+1} is computed as: $m_i = \dfrac{P_{i+1} + P_{i-1}}{2}$.

$$\begin{aligned} f_s &= \sum_{i=0}^{n} \left\| m_i - (P_i + D_i) \right\|^2 \\ &= \sum_{i=0}^{n} \left(\left[(m_{i,x} - (P_{i,x} + D_{i,x}) \right]^2 + \left[(m_{i,y} - (P_{i,y} + D_{i,y}) \right]^2 \right) \\ \frac{\partial f_s}{\partial D_{i_x}} &= -2m_{i,x} + 2P_{i,x} + 2D_{i,x} \\ \frac{\partial f_s}{\partial D_{i_y}} &= -2m_{i,y} + 2P_{i,y} + 2D_{i,y} \end{aligned} \tag{23}$$

We can now add the coefficients found in the equation 23 to the matrices A and b as follows:

$$\begin{aligned} A_{reg}[i][i] = 2\lambda,\ A_{reg}[i+n][i+n] = 2\lambda \\ b_{reg}[i] = 2\lambda(P_{i,x} - m_{i,x}),\ b_{reg}[i+n] = 2\lambda(P_{i,y} - m_{i,y}) \end{aligned}$$

The influence of the λ value will be discussed later in the section 4.6.3.

It is now possible to solve the system $(A + A_{reg}).x = (b + b_{reg})$ to get D.

4.4.5 Outliers detection

A single false data point can produce an high change on the curve. Thus, at each iteration, we compute the standard deviation σ_d of the distances d between the points and their footpoint on the curve. Then, if $d_i > 6\sigma_d$, the data point i is not considered anymore. The result are discussed in the section 4.6.5.

4.4.6 Instability due to the lack of points

In some cases, no or too few data points have their footpoint on the curve between two control points as $\exists\ i\ as\ \neg\exists\ t_k \in [t_i, t_{i+1}[$. The matrix A can thus become very small in some lines and bring very high values for D.

To counter this problem, we sum each line of the matrix A and compare it to a threshold. If the sum is lower than the threshold, the control control point is moved manually, half way to the middle of the segment connecting its valid neighbours. The method is shown in figure 16 from the Pekelny's report [7].

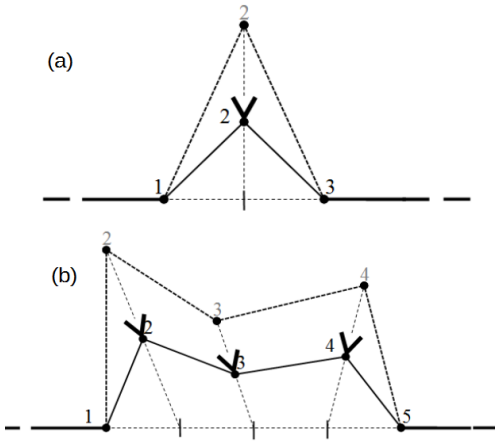


Figure 16: Method for moving manually the control points for a single manual move (a) and multiple moves (b)

Then considering the j^{th} line of the A matrix which was too small, we now set the coefficients as follows:

$$A[j][j] = 1,\ A[j][\neq j] = 0,\ A[\neq j][j] = 0$$

$$b[j] = 0$$

4.4.7 Stop conditions

For each iteration (or loop) l of the algorithm, an error term ϵ_l is computed as follows:

$$\epsilon_l = \frac{1}{nb\_of\_points} \sum_k d_k^2 \tag{24}$$

Then, this squared average is compared to a threshold value and to the previous value of ϵ_l with $\epsilon_0 = +\infty$. If ϵ_l is lower than the threshold, the current B-spline is optimized and the algorithm stops. If $\epsilon_l > \epsilon_{l-1}$, the B-spline at the iteration $l-1$ is kept and the algorithm stops. Finally, if $l = l_{max}$, the algorithm stops and the last B-spline is saved.

For a greater processing speed, the gradient of ϵ_l can be computed in order to stop the algorithm if the variation becomes too small.

4.5 Point cloud generator

In order to test the algorithm, a generator of point cloud has been implemented. It consists into drawing a circle with a few points, and randomizing their position around a perfect circle. Then points are generated all around the curve following a normal law.

The figure 17 shows different shapes generated for testing purpose. Various parameters like the standard deviation errors of the data points and shape points and the number of edges are tested in those examples.

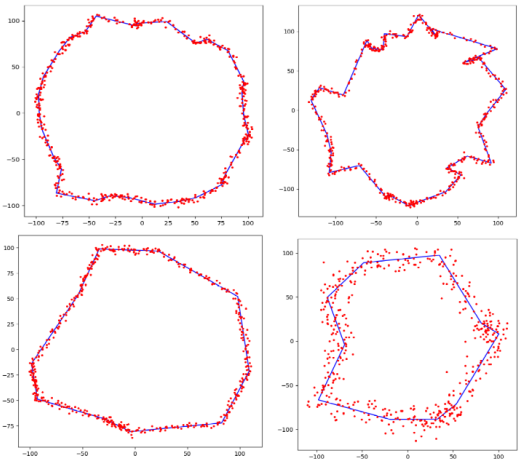


Figure 17: Examples of generated testing shapes with the raw shape in blue and the data points in red

4.6 Results

This section will describe the results of this method and some improvements that can be done.

4.6.1 Testing

All along this section, we will try to fit a point cloud and varying:

- The ration $\frac{n}{p}$ with n the number of control points and p the number of point in the cloud
- The pondering factor λ of the regularization term
- The sharpness of the point cloud

First the figure 18 shows what the result looks like. It has reached its minimum error in 1 iteration.

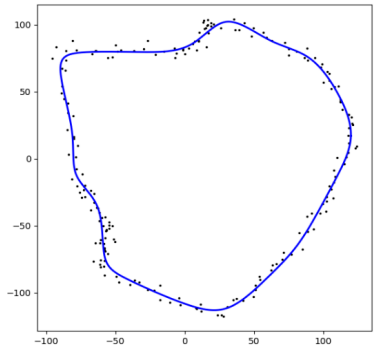
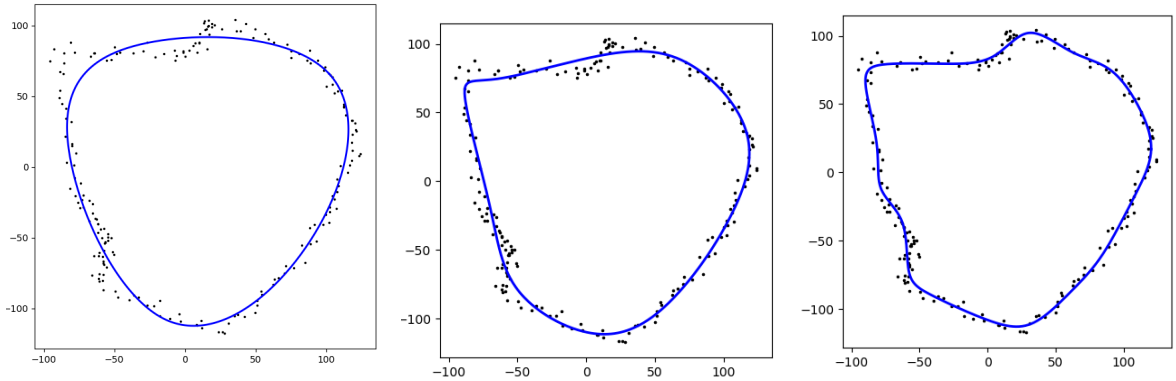


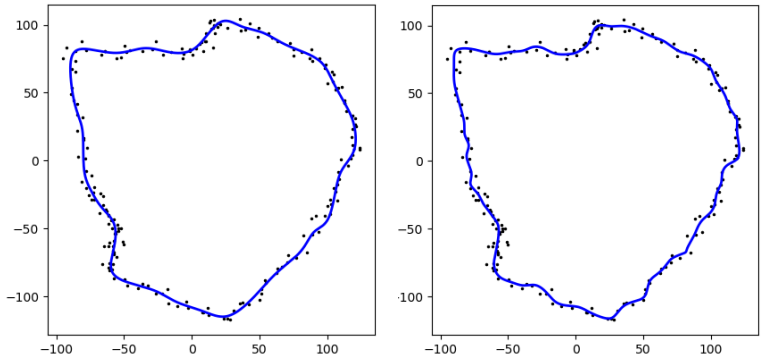
Figure 18: B-spline fitting in a basic point cloud (n=20, p=200, λ=0.1)

4.6.2 Influence of the number of control points

The figure 19 displays several curve reconstruction processes for various number of control points.



(a) 5 control points (b) 10 control points (c) 20 control points



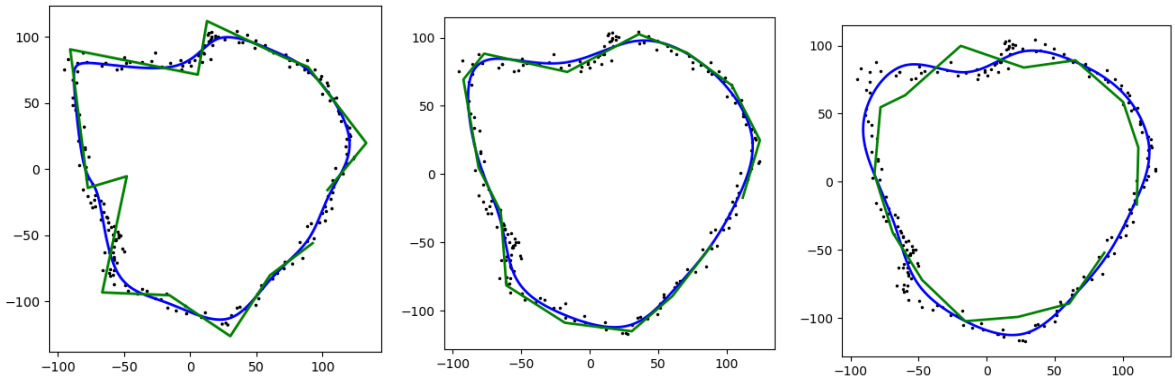
(d) 40 control points (e) 80 control points

Figure 19: Influence of the number of control points on the curve (p=200, λ=0.1)

The number of control points describes the roughness of the shape. In our case, we want a high degree of precision on the body but if the data points become too few or too noisy, we will face an over-fitting problem as on the curve 19e. However, choosing too few control points won't be enough to fit the contour of the human waist, as on the curve 19a that is barely a round shape. 15 control points seem to be great for our purpose.

4.6.3 Influence of the regularization coefficient's value

Tests with several regularization coefficient's value are shown in figure 20.



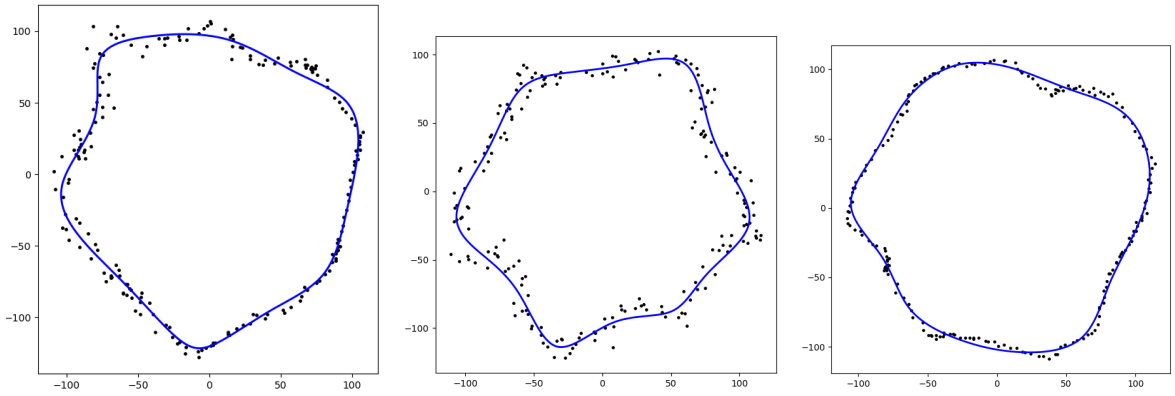
(a) λ=0.01 (b) λ=0.1 (c) λ=10

Figure 20: Influence of regularization coefficient's value on the curve (n=15, p=200). The control points are shown in green while the curve is in blue

As we expected, the higher the λ's value, the more the curve will tend to minimize its circumference, i.e. becoming a circle.

4.6.4 Influence of the sharpness of the point cloud

The figure 21 shows the curve fitting for low, high and both point density:



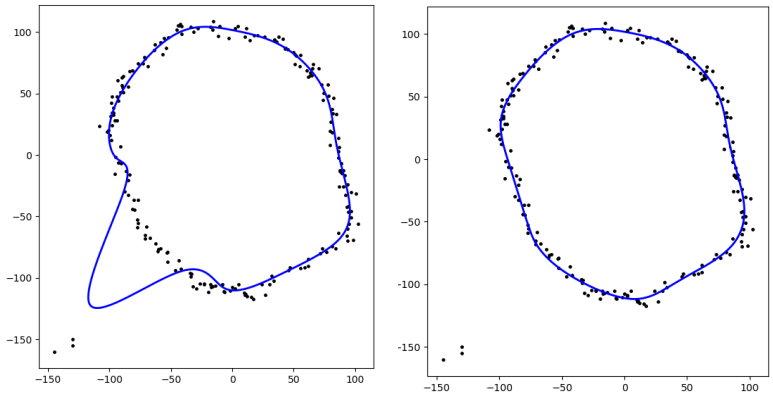
(a) Low and high density point cloud (b) Low density point cloud (c) High density point cloud

Figure 21: Influence of point cloud's sharpness on the curve (n=15, p=200, λ=0.1)

The results are quite satisfying as the curve always fit with the data. When the density is quite low as on the figure 21b, a few number of control points is required, contrary to the figure 21c where we notice that the curve is lacking of control nodes to fit perfectly with the data.

4.6.5 Outliers detection

The figure 22 shows clearly that the outliers detection works great when they are few enough. However, it won't work if the initial B-spline is fitted close to them.



(a) Without outliers detection (b) With outliers detection

Figure 22: Test of the outliers detection (n=15, p=200, λ=0.1)

4.6.6 Possible improvements

Several issues occurred while testing this solution.

First, if the set of data is becoming large or if the desired number of control points is too high, the algorithm can take a lot of time to run. A solution could be to run the program on a faster microprocessor or too run the algorithm with multiple threads.

It appears also that sometime, the number of control points is not well adapted to the shape (as seen in the section 4.6.2). A paper from Yang et al [9] presents a method to increase or decrease the number of control points at each iteration until the desired error becomes lower than the objective.

Moreover, it could be a great idea to implement an improved outliers detection as the currently implemented one is not robust to an high outlying points.

Finally, a better regularization function should be integrated, bringing probably better results. Indeed, the current one only take in account the control points' position instead of the B-spline curve.

5 Running the program from a computer

It is possible to run the whole program without accessing to the Raspi GUI's remotely.

5.1 User's manual

A serial bus is connected to the Raspi, and a USB-TTL converter allows the user to plug it to a computer. The program is run automatically at the Raspi's start-up and any terminal can be used to interact with it. Notice that the baudrate should be configured to 19200 and a constant should be set to *True* to tell the Raspi to print and read on the serial. Finally, an other constant must be set to *False* to avoid the Raspi showing the plot when using the serial.

5.2 Implementation

The printing and reading functions have been overrode in order to choose where to read or print (serial or console).

Moreover, as the RX/TX ports are already taken by the servomotors serial, simple GPIO's are used and the serial is emulated in the software by a bitbang library. Thus, the baudrate cannot be set higher as the speed is limited by the CPU's speed.

List of Figures

List of Tables

References

[1] V. Maire, "Internship report: Conception and implementation of a non contact waist circumference measurement system," July 2018.

[2] United States Consumer Product Safety Commission, "Database: Anthropometry of infants, children and youts to age 18 for product safety design." `www.humanics-es.com/recc-children.htm`, 1977.

[3] US Army, "Database: Anthropometric survey of us army personnel (ANSUR II)." `www.openlab.psu.edu/ansur2/`, 2012.

[4] G. G. Slabaugh, "Computing euler angles from a rotation matrix," Jan. 1999.

[5] W. Wang, H. Pottmann, and Y. Liu, "Fitting B-spline Curves to Point Clouds by Curvature-based Squared Distance Minimization," *ACM Trans. Graph.*, vol. 25, pp. 214–238, Apr. 2006.

[6] D. Umbach and K. N. Jones, "A few methods for fitting circles to data," *IEEE Transactions on Instrumentation and Measurement*, vol. 52, pp. 1881–1885, Dec. 2003.

[7] Y. Pekelny, "Implementation of an algorithm of fitting b-spline curve to cloud points." `www.cs.technion.ac.il/~cs234326/projects/BspFitting_website/index.html`, Oct. 2005.

[8] V. V. Mahla, "Fitting b-splines curves to coarse data cloud using modified squared distance minimization method." `uta-ir.tdl.org/uta-ir/bitstream/handle/10106/26161/MHALA-THESIS-2016.pdf?sequence=1`, Aug. 2016.

[9] H. Yang, W. Wang, and J. Sun, "Control point adjustment for b-spline curve approximation," *Computer-Aided Design*, vol. 36, pp. 639 – 652, June 2004.