# Drawing robot

Ferran Alet and Maria Bauza
Tutor: Vicente Jimenez
*Projectes d'Enginyeria Fisica 2*
(Dated: May 28, 2015)

Teachers spend a great part of the class writing and drawing on the chalkboard: a time-consuming and error-prone task. We construct a system that takes a picture (for instance, the teacher's notes) and writes it on the board. Being cheaper than a projector, it is also portable and adaptable to any blackboard. Moreover, the teacher can later draw on top of the drawing, still having all the advantages that the chalkboard has over the projector.

## I. PHYSICAL SYSTEM

We considered two possibilities for the implementation:

1. Two bars, one horizontal and one vertical. A motor moves the vertical bar along the horizontal one and then another moves the pen along the vertical bar.
2. Two motors that move two chords attached to a gondola with the pen. The length of the chords determine the exact point in the plane (the intersection of two circles is two points, but gravity guarantees that we are drawing below both motors).

We decided that the second option was simpler, more elegant and probably cheaper. Moreover, it is also smaller and can be transported easily.

When we went online to see if someone had already thought of a similar idea we found that indeed the *Polargraph* project sold a whole system or some of its parts. Being a complicated project, having no prior experience and few weeks to do it, we decided that it was too risky to try to do everything from scratch; since, in case something didn't work, we would have to buy another option, loosing a considerable amount of time. Therefore, we decided to buy directly some components from the Polargraph project.

Our system consists on:

- **Beaded chord**, beads provide more resistance so that chord does not slip.
- **A gondola**, bought in pieces at Polargraph, it supports the pen and has connections for the chords.
- **2 bipolar stepper motors** with a pair of $26Ncm$, close to what we estimated we needed.
- **2 aluminum supports for our motors**, molded by our tutor Vicente Jimenez.
- **2 counterweights**, home-made; we set their weights through trial and error.
- **2 sprockets**, 3-D printed by polargraph, to connect the motors with the chords.
- **wires and solding instruments**
- **Servo**, which moves a lever that, when put against the blackboard, separates the gondola and avoids drawing.
- **Arduino UNO** to send orders to the motors.
- **Motor driver** done by our tutor to convert the 5V Arduino signal to the 9-12V needed by the motors.
- **Matlab** to analyze the images.

Notice that our system does not include a blackboard, since it is designed to work in (almost) all of them. The user can set any possible size up to 5 meters in length; only limited by the blackboard's height. Since the length gondola-counterweight is constant, the distance from the motor to the floor has to be at least as large as the diagonal of the drawing (the range in length for the gondola has to be the same for counterweight).

Different motor supports were considered. Our first trials were done with velcro both on the motors and on top of the blackboard; which worked well, but we finally set for aluminum frames that can be set to two different positions depending on the thickness of the blackboard.

## II. DESIGN AND CALCULATIONS

To do all our calculations we used the values we employed in most of our drawings, though we also tried other values.

- Distance between motors $d = 750mm$.
- Height (range for y): between 100 and $750mm$.
- Mass of the gondola $m_g = 160g$.
- Mass of each counterweight $m_a = m_b = 110g$.

In practice, we can easily adjust every parameter except for the mass of the gondola, which, as we will see, establishes a small range of reasonable values for $m_a, m_b$, but since the dimensions of the drawings always appear in ratios in equations, our design is easily scalable.

### A. Forces

First of all, note that the main specification for the motors is pair, not force; however, as sprockets radius are fixed at 5cm, it is equivalent to about forces. We can make a sketch of the static problem as follows:

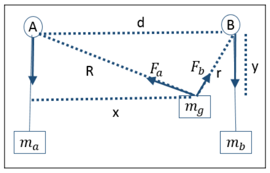


FIG. 1: Diagram of forces

From it, imposing that we have equilibrium and thus the total forces must be 0 we have:

$$F_x = F_b\frac{d-x}{r} - F_a\frac{x}{R} = 0$$

$$F_y = m_g \cdot g - F_b\frac{y}{r} - F_a\frac{y}{R} = 0$$

From the first equation we have: $F_a = F_b\frac{d-x}{x}\frac{R}{r}$. Substituting in the second equation we get:

$$F_b = \frac{g \cdot m_g \cdot r}{y}\frac{x}{d}, \quad F_a = \frac{g \cdot m_g \cdot R}{y}\frac{d-x}{d}$$

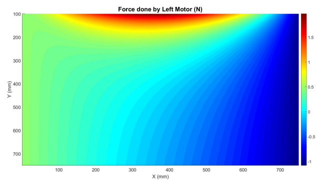
However, to obtain the real forces done by each motor we have to subtract their counterweights:

$$F_{MotorA} = F_a - g \cdot m_a = g(\frac{m_g \cdot R}{y}\frac{d-x}{d} - m_a)$$

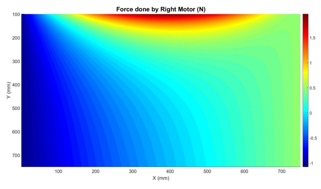
$$F_{MotorB} = F_b - g \cdot m_b = g(\frac{m_g \cdot r}{y}\frac{x}{d} - m_b)$$

Notice that the masses of the counterweights essentially shift the range of $F_a, F_b$. Thus, we have to chose them so that the ranges are approximately centered at 0 to minimize the maximum force the motor has to do. We settled for masses slightly less than that of the gondola.

Calculating the force for the 2 motors at all possible points (for $y > 10cm$, since it diverges for $y = 0$) results in figure 2. It can be seen how a motor does most effort when the gondola is on top (since the angle is very small) and does no work when the gondola is just below the other motor.



(a) Left Motor



(b) Right Motor

FIG. 2: Forces done by each motor

Adding the counterweights helps to reduce the work necessary on top, but can also cause problems if the motor did not need to do much force, since now it has to do it in the other direction. To measure that we calculated for every point the maximum effort of one of the 2 motors, regardless of the direction, obtaining figure 3.

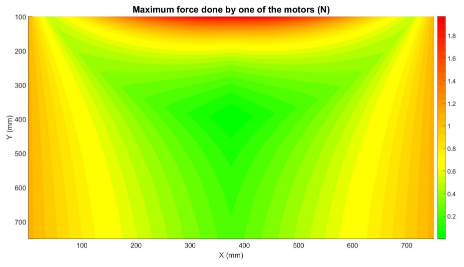


FIG. 3: Maximum tension

Notice that on top the tension is too big and at the sides the tension for one of the motors is too small; thus, we can only work at the central region. We confirmed that when we did the experiments. With a reasonable height ($> 10cm$) all tensions are at most $2N$. Given that we use sprockets of radius 5cm that translates to $10Ncm$, comfortably below the $26Ncm$ in specifications.

## B. Precision

Not all parts of the drawing are equally precise because taking a step will have more effects in some parts of the drawing than in others. To calculate this, we did the following for every possible point (x,y):

1. Easily calculate the length of the chords $R$, $r$
2. Take one step more and one less for $R$: $R+s$, $R-s$
3. Using the formula of the intersection of 2 circles from [1] we calculate $(x_+, y_+)$ for lengths $(R+s, r)$ and $(x_-, y_-)$ for $(R-s, r)$ and then take $dR = ||(x_+ - x_-, y_+ - y_-)||$: the change in position when moving a step up and down in motor A.
4. Similarly we calculate dr for motor B.
5. precision$(x,y) = \sqrt{dR(x,y) \cdot dr(x,y)}$ as a characteristic measure of precision; less is better.

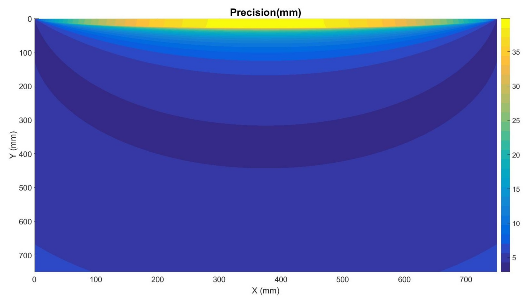


FIG. 4: Precision

We can see that we have a good precision roughly everywhere except for the top region. However, this region was already discouraged because we had too much tension on it.

## C. Velocity

We use the wave drive mode to control each motor and in our setup each step requires a minimum time of 10ms. Given that every step (following specifications) is $0.475mm$ we have a velocity of $4.75cm/s$. This only gives an order of magnitude because both motors can move at the same time (thus moving more distance).

This velocity allows a fine velocity, completing a drawing in the order of 2 minutes.

## III. CODING

### A. Drawing accurately the image

The first challenge in drawing is choosing what lines to color to achieve an accurate, or at least recognizable, drawing of the picture. We decided to do the following:

1. Convert the image to black and white.
2. Set a threshold for the gray level and convert it to an image of 1s and 0s.
3. Using this binary image, find its borders.

Most simple images we wanted to draw, mostly text, have few colors and lose little information in the first 2 steps, converting each pixel either to black or white. Notice that the gray threshold has to be adjusted for every image, since some images are darker than others.

Then, we also found that a common and useful way to represent an image is using its borders (lines that separate connected regions of the same type of pixels). After obtaining them using Matlab's tools, we process them. First we eliminate the very short ones since they are generally small mistakes and unimportant. Then we try to solve a big problem derived from using the borders: when we analyze the image of a text every letter/digit has a border that goes around it, thus essentially duplicating the letter, which we don't want. We therefore implemented an algorithm that carefully tries to erase the duplicates by trying not to draw near where it has already drawn. This is not trivial because it is not easy to separate those points that are close to the place you want to draw but belong to a different border and so you do not want modify them.

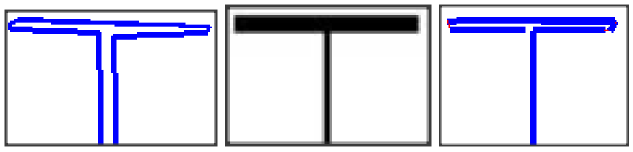

(a) Double(bad) (b) Original (c) Single(good)

FIG. 5: Correction of double drawing

Finally, another thing that has to be taken into account is that lines in a picture do not correspond to lines in the 'motor space': for example, if position A corresponds to lengths 10, 20 and position B corresponds to positions 20, 40 then the point corresponding to lengths 15, 30 is no on the line AB. Thus, we had to change our algorithm to work with curves instead of segments.

### B. Correcting some problems

When one combines programming with a physical system many issues can arise; we will comment the two most significant examples.

An important fact we needed to take into account is that we cannot approximate anything. For example, we

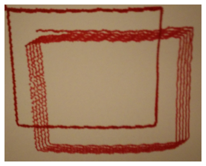


FIG. 6: Accumulation of small errors

noticed that several drawings started to look worse at the end. We first thought that motors were losing steps, but the reality is that microerrors done by rounding to integer steps, started to accumulate resulting in noticeable errors at the end. Figure 6 is the comparison between 5 rectangles with the microerror, successively accumulated, and 5 rectangles one on top of the other in the case with rounding errors carefully solved.

Another important issue that must be considered is the small computing capacity of Arduino. Because of that, most computations had to be done in Matlab. We just passed to Arduino movement instruction (moving a motor or the servo) and the time to wait until the next instruction. We calculated that the communication would be much faster than drawing and thus, we could send all the instructions one by one without worrying about it taking too long.

However, after several experiments we observed that the contrary was happening: because Arduino was busy doing the picture it couldn't read all the instructions as they came and some were lost because the channel was saturated. Therefore, we instructed Matlab to also handle the time and just send an instruction to Arduino every time one needed to be done.

### C. Traveling Salesman Problem

Once we had found the lines we wanted to color we had to figure out in what order we had to draw them, as different orders would lead to very different trajectories and therefore, very different drawing times. We want to minimize the overall drawing time as much as possible, consequently, our measure of time is the euclidean distance between points traveled plus an extra penalty if we had to put the chalk up between the points (when the distance between endpoints is very small we don't put the servo up since there is no gain in resolution).

Finding such order is a famous NP-hard ([2]) problem, for which there is no polynomial (and hence fast) exact algorithm. Thus, one must restrict to approximation algorithms that can also find good results. Our problem of joining curves with the fewest possible time is fundamentally different from joining points; thus, we could not use prewritten algorithms and had to code a tailored adaptation of one of them: 2-opt.

2-opt is an algorithm that finds local optimum paths in the TSP between points. We first describe the original 2-opt and then the changes we did.

1. Start with a random order
2. While we haven't arrived to a local optimum, randomly choose a subpath $p_i...p_j$ and try to reverse it, thus changing the order $p_1...p_{i-1}p_ip_{i+1}...p_{j-1}p_jp_{j+1}...p_n$ to $p_1...p_jp_{j-1}...p_{i+1}p_i...p_n$.
3. Regardless of its length , for every subpath changed we are only changing two edges: $p_{i-1} - p_i$ to $p_{i-1}p_j$ and $p_j - p_{j+1}$ to $p_i - p_{j+1}$. Thus, we can easily evaluate if reversing the path improves it. If it does; keep it reversed and restart the search for other subpaths.
4. If we have tried all possible subpaths and none improves the current path, the algorithm terminates.

Since it is a randomized algorithm, we may get different results in every simulation; thus, it is advisable to repeat it several times trying to find good local optima.

We tailored the algorithm to handle curves instead of points. From our curves we are only interested in their endpoints. The algorithm handles 2 things: the order between the curves and the sense in which we travel every curve. Let us enumerate the curves from 1 to n, let their order be $o_1, ..., o_n$ and the endpoints from curve $o_i$ be $p_{i,1}$ and $p_{i,2}$. Finally, let the sense of curve $o_i$ be $s_i$, describing the first point of the curve. Then $p_{o_i,s_i}$ will be the starting point of curve $o_i$ and $p_{o_i,2-s_i}$ will be its final point.

Since the sense in which we travel a path internally doesn't change the time to walk through it, we may only care about the time spent going between curves, but not along the curves. Therefore the total distance is:

$$\sum_{i=1}^{n-1} dist(p_{o_i,2-s_i}, p_{o_{i+1},s_{i+1}})$$

Then, we run the algorithm with some adaptations:

- When we reverse a subpath of curves we are also changing the sense in which we walk all the curves, thus we set $s_{i...j}$ to $2 - s_{i...j}$.
- The edges changed are still 2, but they are now between endpoint of curves.
- In the original algorithm there was no point in choosing to reverse a subpath of length 1 since it would not change anything. On the contrary, now choosing a subpath of length 1 is reversing the sense in which we traverse the curve.

As figure 7 shows, this algorithm greatly reduces drawing time. Red thin lines represent movements without drawing; one can observe that after this improvement only a few short red lines are done.

## IV. RESULTS AND CONCLUSIONS

We were able to construct an end-to-end system that given an image can draw a good representation of it on a blackboard with great level of precision. The order of magnitude in time is under 5 seconds in computation and 1-5 minutes of drawing, depending on the size.

The system can paint very complex pictures such as the Mona Lisa (figure 9) with reasonable detail, being clearly recognizable. In the domain we wanted to specialize (text that could be written in a class) the system performed almost perfectly; as can be seen in figure 8.

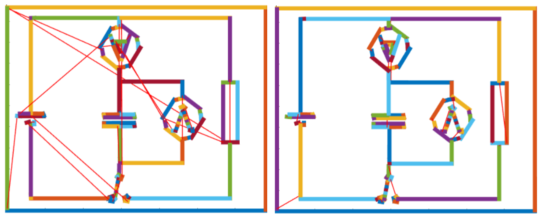

(a) Wihout TSP (b) With TSP

FIG. 7: Improvement using TSP algorithm

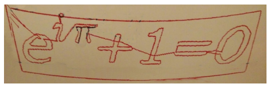


FIG. 8: Euler equation (without lifting the pen)

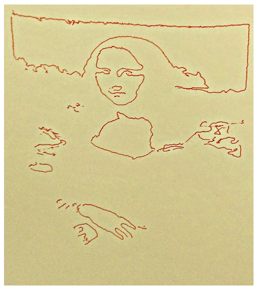


FIG. 9: Drawing of da Vinci's Mona Lisa

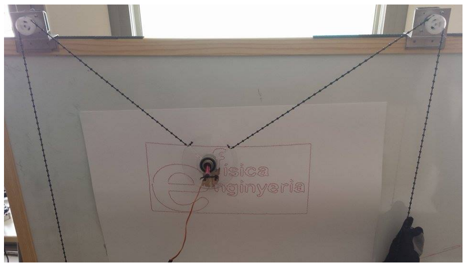


FIG. 10: Robot drawing Enginyeria Fisica logo

To improve this system we suggest increasing its velocity (mainly limited by the power of the motors, and therefore their cost). We would also like to explore possible applications that try to help a teacher in giving class, such as being able to talk directions to the robot or being able to write functions to Matlab and have them plotted on the blackboard.

[1] Mathworld Wolfram on Circle-Circle Intersection. `http://mathworld.wolfram.com/Circle-CircleIntersection.html`

[2] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. (2001), "35.2: The traveling-salesman problem", *Introduction to Algorithms* (2nd ed.), MIT Press and McGraw-Hill, pp. 10271033, ISBN 0-262-03293-7.

[3] G. A. CROES (1958). A method for solving traveling salesman problems. Operations Res. 6 (1958) , pp., 791-812.

[4] S. Lin and B.W. Kernighan *An Effective Euristic Algorithm for the Traveling-Salesman Problem* , Operations Research Vol. 21, No. 2 (Mar-Apr. 1973) 498-516