AN OPTIMAL CLEANING ROBOT FOR ALL KINDS OF REFLECTION POOLS

Valério Ramos Batista, Francisco de Assis Zampirolli, José Artur Quilici-Gonzalez
Federal University of ABC
Av. dos Estados 5001, 09210-580 St André-SP, Brazil
valerio.batista@ufabc.edu.br, fzampirolli@ufabc.edu.br, jose.gonzalez@ufabc.edu.br

ABSTRACT
In this paper we introduce the first part of a long-term project devoted to building an autonomous robot that is optimal at cleaning any kind of reflection pool. In our context, optimization refers to both minimal energy consumption and shortest time spent on cleaning the whole pool effectively. This means, the robot is programmed not to leave a single spot need either brushing or dirt suction. Our robot differs from other automata especially because it resorts to a unique combination of mathematical, computational and automative strategies that involve Plane Geometry, Complex Analysis and Genetic Algorithms.

KEY WORDS
Cleaning robot, Reflection pools, Optimization.

1. Introduction
Robotics has already become part of our daily life in numerous and various activities. Agriculture, medicine, automakers and housework are just four examples in which robots perform their specific tasks. From cherry-harvesting [1] to coronary artery bypass surgery [2], from driverless cars [3] to vacuum cleaning [4], we could cite hundreds of works on robotics. Many popular robotic vacuum cleaners are not endowed with location capabilities to create or use any mapping [5], following instead a random spiral-like path around the room [6].

As for our present paper it is strongly based on autonomous navigation, which involves enabling the robot to find a new route to avoid obstacles that can suddenly appear on its way. This is a challenging problem in dynamic environments, as the robots have to quickly learn how to deal with moving obstacles, irregular terrains, low luminosity and shadows, etc. In a static environment most of the hurdles can be previously mapped and then avoided by the robot. Just a simple computer vision system is not enough to tackle unpredictable events quite common in a dynamic environment. There one needs a perception capacity to take the right action, and research on robot perception is still at a relatively slow development rate. In spite of that, there are intelligent robots that achieve fairly good results in some simplified dynamic environments if certain constraints are fixed beforehand.

A successful and popular example used in dynamic indoor environments is the Roomba robot [5, 6]. It employs reactive navigation [7] to vacuum any room with a flat floor. Basically, this type of robot operates on a random path and reacts to any obstacle by going round it or just turning back. More complex tasks require map-based navigation, which enables the robot to choose an optimal path. But in general this technique needs a control system that is much more sophisticated than the reactive navigation.

In this paper we present an application of map-based navigation for cleaning any kind of reflection pool, which will be considered as a simplified 2D outdoor environment that may contain obstacles. These are not known beforehand. Our contribution to this growing research area brings a unique combination of strategies to minimize time and cost. Moreover, these strategies rely on classical results from Plane Geometry, Complex Analysis and Genetic Algorithms. Therefore, they are accessible even to undergraduate students in Exact Sciences, in such a way that practical Engineering and Academia are still kept tight together. Hence, although robotic pool cleaning has already been achieved in previous works, like [8] and [9], we intend to improve it even more by adding the optimization through some accessible theory studied in undergraduate courses.

For the time being, just with [10] and [11] as main references we are able to project the way our robot will clean reflection pools. Details are given in the next sections, but at this point we must explain that swimming pools, tanks, aquariums and the like are not included in our present research. This is because we use some particularities to solve the problem as plainly as possible. For instance, reflection pools normally have a constant depth, and Section 2 shows that in this case tracing a map of the pool is something the robot can easily do by itself. Whenever possible the 2D-reduction makes the problem much easier to handle.

Saving energy will not only make the robot's work less expensive. Reflection pools can be large and they normally have obstacles: statues, fountains, etc. Hence the robot should not work connected to the mains but carry a battery instead. The battery will last on minimizing energy consumption, which increases with the pool size, the complexity of its shape and the number of obstacles. That is why the robot will have to maneuver as little as
possible and at a record time. Battery consumption is also influenced by the robot's weight.

But this is a quantity that we cannot estimate for now, because the robot must be small enough to pass through close obstacles, and at the same time carry a dirt container. Our purpose is not to build a robot for heavy cleaning. It will serve to keep the bottom clean, so it should work more frequently in the pool than the cleaning personnel. In this case we can project a robot with a light and small dirt container.

This is a long-term research, of which the first steps are presented and explained in the next sections. In this paper we show the whole programming strategy of our robot that will clean any kind of reflection pool in an optimal way.

2. Identifying the Pool

Usually the robot will clean the same pool again, say after a couple of days. But for the first time you should switch on the mode called newpool and the robot will map it with a method explained right next. However, you should always place the robot on a corner of the pool and in contact with the wall, because the same method will help find any pool that already exists in a database.

As we have already mentioned, the robot optimizes both time and energy consumption at cleaning the whole pool. This optimization does not include previous tasks like mapping or recognizing the pool beforehand. However, the user can skip these previous tasks by storing a name for the pool right after having it cleaned by the robot for the first time, and also place the robot always on the same corner. Whenever a name is chosen a display will show the map of the corresponding pool with a glyph indicating that specific corner. You should erase this glyph in case another corner is chosen for the robot to start, and it will identify its own location after having moved for a certain time.

Figures 1 and 2 illustrate the mapping method. Inspired in the “wall follower”, a maze solving algorithm, the robot will move at a constant speed while keeping its right-hand side in contact with the wall. At each change of direction the robot adds the displacement vector to a total sum that will be zero exactly when it completes a circuit. While moving along the outer contour the robot captures signs perpendicularly to its trajectory. Right after completing the first circuit these signs are compared with the ones that would be obtained in the case of a single contour component. Any mismatch will indicate the existence of obstacles inside the pool, but its exact number is uncertain. For instance, Figure 1 shows two obstacles embracing each other, and these could be interpreted as a single one if we relied just on sign mismatch.

Therefore, each mismatch will be investigated by the robot, which will approach the potential obstacles and also move along their contours according to the right-hand rule while capturing signs that are orthogonal to its trajectory. For each completed circuit the robot recalculates the mapping together with the new sign information. These data will eventually match, and this concludes the identification of all contour components of the pool.

![Figure 1. Applying the right-hand rule to the outer contour.](image1)

![Figure 2. Applying it again for each remaining component.](image2)

For the time being the chosen signs are images recorded by two video cameras that are attached to the robot. Of course, these images will not identify obstacles in the case of a completely dark pool, for example. Hence our present version depends on zebra tapes stuck to the whole extent of each contour component. The zebra pattern also helps in the case the robot cannot move at a constant speed due to wind blows, water waves, floors that are either too rough or too slippery, etc. Thus it is more reliable to count the zebra stripes instead of just timing the displacement of the robot. We shall also test other kinds of sign like ultrasound and laser, as discussed later in Section 6.
Moreover, our robot's dimensions are relatively small: 0.5, 1 and 1 meters of width, length and height, respectively. Therefore, we assume that it will always pass through narrow stretches and between obstacles, as depicted in Figures 1 and 2. After having obtained the map it is forwarded to the user, who will have to quadrangulate it for the robot to go over each single square according to an optimal sequence. As a user's task, quadrangulation implies that our algorithm is for now just semi-automatic. In future we shall make it automatic, and also apply a genetic algorithm combined with a reward-punishment strategy to find the aforementioned optimal sequence of squares.

We are also working on other methods for the robot to map the pool, this time in 3D. Details will be given in Section 6, but for now we must explain that different mapping methods are mandatory whenever one seeks optimization. By comparing their overall performance, we shall be able to project the most adequate robot's physical structure. Eventually one method can even turn out to be a complement of the other.

In the next section we explain how to choose an adequate quadrangulation. Regarding the reward-punishment strategy, it is discussed in Section 4.

3. Quadrangulation

The easiest approach is to subdivide the internal part of the pool with rectangles that fit the robot's dimensions, or even squares of side length 0.5m, so that the robot will always displace 0.5m horizontally and/or vertically. This fixed value is inconvenient because a considerable area will be dismissed, as exemplified in Figure 3. Since the robot will have a map then it can compute how much to displace in each direction and so even avoid collision with the wall. However, this still leaves a contour of dirt as depicted in Figure 4. Hence, moving only in two directions will lead to a lax work. But it is difficult to implement a robotic movement with many degrees of freedom.

![Figure 3. A standard quadrangulation.](image1)

Figure 3. A standard quadrangulation.

![Figure 4. Some extension of the horizontal/vertical movements.](image2)

Figure 4. Some extension of the horizontal/vertical movements.

Of course, the robot could clean the border of the pool while it performs the route illustrated in Figures 1 and 2. This would remove the trace of dirt shown in Figure 4. However, such a strategy would penalize the optimization we are searching for, especially in the case of pools that have obstacles.

Our approach is to make use of a non-Euclidean quadrangulation. See Figure 5. As we have mentioned in Section 2, for the time being we work with a semi-automatic method in which the user chooses a quadrangulation with the help of the Schwarz–Christoffel Matlab Toolbox. See [11] for details.

The Schwarz-Christoffel mapping is conformal. Because of that, in Figure 5 one can see the curves of the non-Euclidian quadrangulation always crossing each other orthogonally. Although the squares are now curved the robot will move between them by taking just one of two directions of a cross, either forwards or backwards. This makes it much easier to program the robot's displacements. It also saves energy if changing direction is reduced to a minimum, especially in the case of reverse gear. This will count in the robot's programming strategy explained in the next section.

But maneuvers will happen with angles just close to 90° because the robot will follow a sequence of vertices that belong to curved squares. As a matter of fact rotation of any angle happens at the very beginning, as discussed in Section 2. Of course, the robot will already clean the contours of the pool while going round them. If a quadrangulation is provided in time, vertices on the contours will already been toggled as “clean” and the rest as “dirty”. Otherwise all vertices of the quadrangulation will start as “dirty”. Future developments will make use of image analysis in real time to decide between “dirty” and “clean”, but then the robot's brushes will have to rise and lower accordingly.
4. The Reward-Punishment Learning Strategy

An autonomous robot should have the capabilities to take decisions to find its way through the hurdles, but at the same time it has to obey some orders such as “stop your journey and return to the base”. Our robot will receive orders through a built-in laptop that captures them through a nearby Ethernet Access Point (802.11). Two video cameras commanded by the laptop will collect images of the environment, whose data will be used for Map Calculation. The Path Planning will be based on a Reward-Punishment Learning Strategy.

In Chapter 9 of [10] the author gives an introductory explanation about Genetic Algorithms followed by a nice practical example: Robby, the soda-can-collecting robot. It works inside a 10x10 array of cells, half of them empty and the other half containing one soda-can per cell. The cans are distributed at random and Robby can act only 200 times to pick them all. Each action must be one of these drill commands: one cell left / right / forward / backward, rest, pick, or a random choice of them. During the machine learning training process, picking in an empty cell or bumping into the wall makes Robby lose 1 and 5 points, respectively, but for each picked can the robot gets 10 points.

Of course, 500 points is the maximum score, which Robby would always achieve had it access to some extra information. It is not the case, but letting Robby always evolve in a Genetic-algorithm-based learning strategy, after 10 thousand tries it got an average of 483 points in the end.

Inspired in this example our cleaning robot will get a graph $G = (V, E)$ generated from a quadrangulation as illustrated in Figure 5 (center / down). Our vertices will play the same role as the cells in Robby’s case. Our robot’s base is 0.5m x 1m and it will have two brushes that rotate with opposite orientations, so that dirt will be swept into a suction pipe. The water gets into the dirt container and passes through a filter, which can be a stiff gauze, so that the cleaned water is expelled back to the pool.

All this must happen while the robot is going on. Due to inertia the start of a displacement consumes much more energy than a continuous move. Hence our robot will avoid stopping while cleaning. The set $V$ will have the least possible number of vertices such that, whenever $v_1, v_2$ in $V$ are connected, their distance is at least 35cm and at most 45cm. This is to ensure that no trace of dirt will be left between any pair of vertices, even in the diagonals of the quadrangulation.

Our first reward-punishment learning strategy will give 3 points whenever the robot cleans a dirty spot, but 1 point is debited every 10s. A forward or backward move represents +1 and -1 point, respectively, but going either left or right gives naught. We have not simulated this strategy yet, but our guess is that the robot will end up with trajectories that resemble spirals. Anyway, some other values of reward-punishment will be tested in order...
to compare the possibly different trajectory shapes that optimize cleaning.

5. Results

For the time being we have already programmed a user-friendly graphical interaction in Matlab to draw and save pool contours, as illustrated in Figure 5 (top). The user can choose strategic points of the contours to divide the pool in subregions, and each of them invokes the Schwarz–Christoffel Matlab Toolbox [11].

More specifically, we use the rectangular mapping of the toolbox, which has just one limitation: subregions ought to be quite regular polygons with a few number of edges. Otherwise the toolbox can crash, and in this case the user must start again from the beginning.

However, this does not mean that our program cannot handle pools with curved contours. From Section 2, the robot will identify curved stretches whenever the tires are not parallel to its main axis during displacement. When the user gets a map like in Figure 5 top, it will show a sample of points on the curved stretch. We may add a dotted indication of the real shape, but the user will access the sample points. After having obtained the quadrangulation, curved squares along the approximating polygonal line will have their sides extended to the actual boundary.

Another limitation is the robot's shape: it cannot reach pointy corners of the pool. However, any shape is somehow fated to this problem, and at the Introduction we mentioned that our purpose is to keep the pool clean rather than build a robot for heavy cleaning. Taking this into account, the user can choose a quadrangulation that profits from wide corners and treat the pointy ones separately, as suggested in Figures 6 and 7.

For the isolated triangle, take the edge opposite to the greatest angle, which you should have made as close as possible to a right angle. In Figure 7 this edge is vertical, and we shall consider it as the base of the triangle. Now choose a point of the base that is closer to the greater between its two adjacent angles. The more these angles are similar in value, the more you approach the middle point of the base for your choice. Hence, when the graph \( G = (V, E) \) is generated it will include vertices sufficiently close to the pointy corner of the pool.

As mentioned before, here we present the first steps of a long-term research. Many simulations and comparisons between different methods will influence the way we shall project the robot physically. The next section deals with just one of the several items of our planning.
6. Future Developments

As mentioned in Section 2 we also have been working on a 3D method to map the pool. There are several techniques of 3D reconstruction with different purposes. For instance, in medicine we cite techniques described in [12] and [13], and specifically in Computer Tomography (CT) we have [14] and [15]. These techniques give a detailed 3D image of the patient's organ, so that its appearance and the occasional presence of a tumor can be easily checked.

In our case the 3D method can follow the scheme presented in Figure 8. It requires two video cameras attached to the robot. In Figure 8, points P1 and P2 represent the images of the same pixel P that were captured by the cameras. P has coordinates (Xp, Yp, Zp) with respect to a fixed referential, and these coordinates are computed via the data depicted in the figure.

Of course, P represents one of several marker signs in the pool, but they can be unequally spaced. So, instead of zebra tapes one can use just a few number of marker signs. If they are equally spaced, then just one camera is enough to map the pool. Figure 9 shows a simple test in which the distance between the camera and the wall is computed for different positions of the camera, in our case a SAMSUNG Galaxy Note 2 smartphone at 8Mp. Our marker signs are black rectangles with dimensions 19.2cm x 8.6cm, but these number are not necessary in our computations.

We begin with Figure 9 top, where the camera was at a distance D ≈ 52cm from the wall, and the alignment between its focal lens and the center of the marker was perpendicular to the wall. Notice that this distance is printed on the top left corner of each marker image in Figure 9. In this case, the image of our marker has a width of x = 1190 pixels. The centroid of the marker image is computed via pixels, and it appears as a red asterisk in

Figure 8. Taken from https://commons.wikimedia.org/wiki/File:Fig_1_3D_reconstruction.png

Figure 9. This was our calibration starting point. If the alignment between the focal lens and the marker center is kept perpendicular to the wall, then any rectangle image with y pixels of width will imply that D' = 52*x/y is the new actual distance.

As a software package for image-based 3D reconstruction we can take the Matlab Computer Vision System Toolbox. See http://www.mathworks.com/products/computer-vision for details and examples.
But measuring distances through image-based reconstruction can show an inferior performance compared with ultrasound and laser rays. They discard marker signs, and if their implementation in the robot proves to be relatively simple, then we shall replace the cameras with suitable emitting devices.

Finally, in order to manufacture the robot we do not have to start from zero. Many commercially available robots already come with a basic structure to which we shall add components that specifically comply with our purposes. The Robotnik Summit XL OMNI is such an example. See https://youtu.be/8sH1a511_q4 for a short demonstration.

7. Conclusions

In this work we have presented the first strategies to build an optimal cleaning robot for reflection pools. Although pool cleaning is an already old issue in robotics, see [8] and [9], our method aims at consuming the least amount of energy and concluding the task at a record time. For these purposes we just use classical results from undergraduate courses in Exact Sciences, especially Plane Geometry, Complex Analysis and Genetic Algorithms.

References