Tic-Tac-Tobot

[EECS 149/249A Class Project]

Patrick Scheffe
p.s@berkeley.edu

Nikolas Alberti
nikolas.alberti@berkeley.edu

Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA

ABSTRACT

1. INTRODUCTION

Our charter defines the goal of this project as follows:

"In this project, we want to build a robot that will challenge you to play a game of Tic-Tac-Toe. [...]The robot will control the movement of a whiteboard marker using two servos. The pen will be set up in a construction which has a shoulder and an elbow joint, so it can be moved to a defined x-y-position. The whiteboard is mounted in a fixed position next to the manipulator. First, the robot draws the playing field consisting of nine tiles. A camera will be used for monitoring the player's move (placing a circle or a cross). Autonomously, the robot will react and make its own move. In order to make space for the human player's moves, the whole construction should be able to move away from the field."

This definition specifies well what the robot should be capable of doing. In order to meet the specification, a robotic manipulator must be assembled and controlled. Moreover, a computer vision component must be set up. At the end, a program joins these components, allowing it to keep track of the game and to compute the robot's next move.

2. HARDWARE

2.1 Arduino Uno

The Arduino Uno is a microcontroller board equipped with an Atmel ATmega328P microcontroller [Atm]. Arduino is an open-source hardware and software project originated in Ivrea, Italy [Ard]. Hardware and software have been distributed by the founders of Arduino (known as Arduino LLC) and the producer of the official Arduino boards (Arduino S.r.l.). The goal of the Arduino project is to make the development on embedded systems as accessible as possible. For instance, the Arduino IDE completely hides technical details like header files from the user.

The Arduino has 14 GPIO pins from which 6 support pulse width modulation (PWM). Since PWM is needed for controlling servo motors, the Arduino matched our requirements. A plus point on the Arduino was that it promised to offer an uncomplicated development.

2.2 Servo Motors

For the actuation, we used three Tower Pro MG90S servo motors [Ser]. The connection to the Arduino was established with ordinary jumper cables. The power from a 5V/2A power supply was distributed with a breadboard to provide power for all three servo motors. Because the power supply originally ended in a coaxial power connector, we modified it to have jumper cables as output for positive and negative voltage.

2.3 Additional Materials

Additionally, the following materials have been used:

- A Microsoft LifeCam HD-3000 webcam
- A white board
- Two white board markers
- Some screws, nuts and bolts
- ABS filament for 3D printing the custom parts

3. ROBOTIC MANIPULATOR

3.1 Choosing the dimensions

For accurate two dimensional actuation, a classical robotic manipulator that consists of a chain of dependent joints is not suitable for a low budget approach. The plotclock, a project by Johannes "*Joo*" Heberlein from the Fablab Nuremberg (comparable to a makerspace) impressively proved a way to make low budget 2D actuation work [Joo]. Therefore, we planned to choose a similar manipulator. However, while the plotclock needs to cover a range whose dimensions in x-direction exceed the dimensions in y-direction, it was not possible to blindly adapt the manipulator. For our project, actuation needs to be performed in a square area. Therefore, we needed to find the optimal sizes of the limbs.

This problem can be described as a minimization problem: The area of a square is given. Find the dimensions L_1, L_2 and L_3 that minimize the sum $L_1 + L_2 + L_3$ while not violating the constraints that the given square fits into the reachable range of the actuator.

Unfortunately, the function of the largest reachable square $A(L_1, L_2, L_3)$ is not defined by a function that allows for partly deriving towards L_1, L_2 and L_3 and setting the derivations to zero. Formulating this problem as a minimization problem to solve with MATLAB for example seemed to WAAAAAAS. We therefore chose to model the reachable space in a geometry software called *Geogebra*, an open source software for supporting mathematical education in schools and universities [Geo]. Then, we empirically derived a near optimal sizing that yields a reachable square of approximately 100cm^2 (Figure 2). A constraint was that we

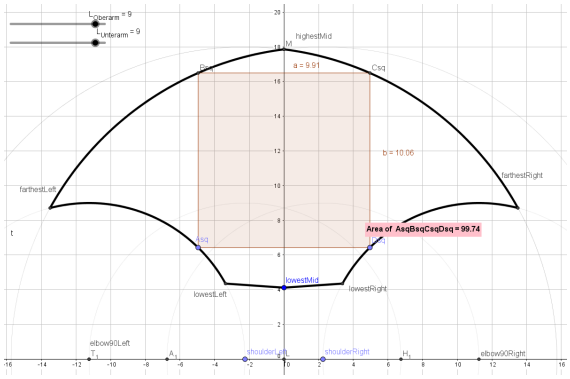

Figure 1: Geogebra Sketch Used for Sizing

wanted the player to face the robot like they would sit on opposite ends of a table. Consequently, choosing orientation of the square freely to allow a better usage of the space was not possible.

3.2 Manufacturing

Finding the right dimensions was a crucial part for building the robot. After finding those, a CAD model of the manipulator was created using Autodesk Inventor [cad]. Those models were converted to STereoLithography (STL) files and printed with the ROSTOCK MAX v3 3D printer in the Supernode lab [3D]. It was important to find the right bed temperature and extrusion speed to avoid warping and have the parts stick to the printing bed.

The joints of the five-bar linkage robot were connected with screws and washers of the right size. To allow the arms of the robot to lift from the whiteboard, the two servomotors actuating the arms can be tilted backwards with a bar connected to their mounting. This bar is also connected to the third servomotor, which is mounted below the whiteboard.

3.3 Simplified Kinematic Model

A simplified sketch of the robotic manipulator is shown in Figure 2. As a first approximation it is useful to determine the angles θ_1 to θ_4 from the given position of the wrist joint at which the two arms coincide (x,y). The key for solving this inverse kinematics problem is to divide it into smaller subproblems that each can be solved individually. For that purpose, the line segments a, b and c are introduced.

From the position of the wrist joint, the following values can be computed directly:

$$\alpha = \arctan\left(\frac{y}{x}\right)$$
$$\beta = \pi - \alpha$$
$$c = \sqrt{x^2 + y^2}$$
$$a = \sqrt{\left(x + \frac{L_3}{2}\right)^2 + y^2}$$
$$b = \sqrt{\left(x - \frac{L_3}{2}\right)^2 + y^2}$$

Now we can solve for θ_2 and θ_3 by either using sine rule or cosine rule. However, the sine rule can be ambiguous in certain setups, which makes case differentiation necessary. Although mathematically steady, in our implementation the

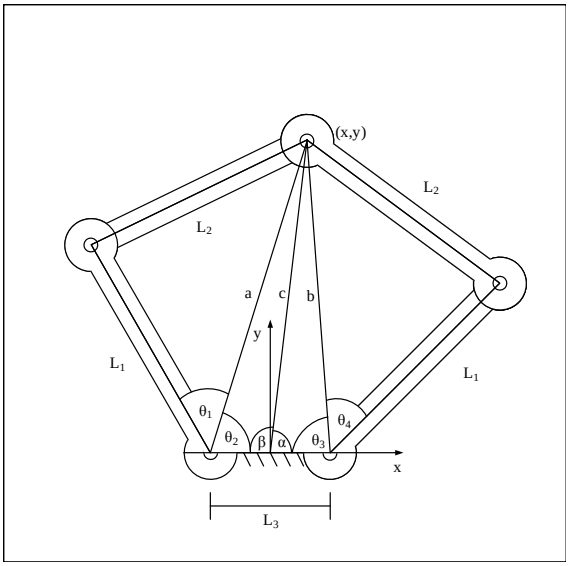

Figure 2: Simplified Version of the Manipulator

domain crossing from one solution to the other resulted in discontinuities of the movement. Therefore, the unambiguous cosine rule solution is preferred:

$$\theta_2 = arccos\left(\frac{a^2 + (\frac{L_3}{2})^2 - c^2}{aL_3}\right)$$
$$\theta_3 = arccos\left(\frac{b^2 + (\frac{L_3}{2})^2 - c^2}{bL_3}\right)$$

Using cosine rule we can also solve for θ_1 and θ_4:

$$\theta_1 = arccos\left(\frac{a^2 + {L_1}^2 - {L_2}^2}{2aL_1}\right)$$
$$\theta_4 = arccos\left(\frac{b^2 + {L_1}^2 - {L_2}^2}{2bL_1}\right)$$

3.4 Complete Kinematic Model

The simplified version of the kinematic model is good for quickly creating a working implementation. However, any movements executed by the manipulator will suffer from distortion. The pen is not mounted *exactly* at the joint's position but in a small distance. Hence, a precise solution is necessary. For that purpose, new definitions are made as shown in Figure 3.

L_4 and ϵ are fixed and not influenced by the position of the manipulator:

$$L_4 = \sqrt{{L_2}^2 + {L_5}^2 - 2L_5L_2cos\left(\frac{3\pi}{4}\right)}$$
$$\epsilon = arccos\left(\frac{{L_4}^2 + {L_2}^2 - {L_5}^2}{2L_4L_2}\right)$$

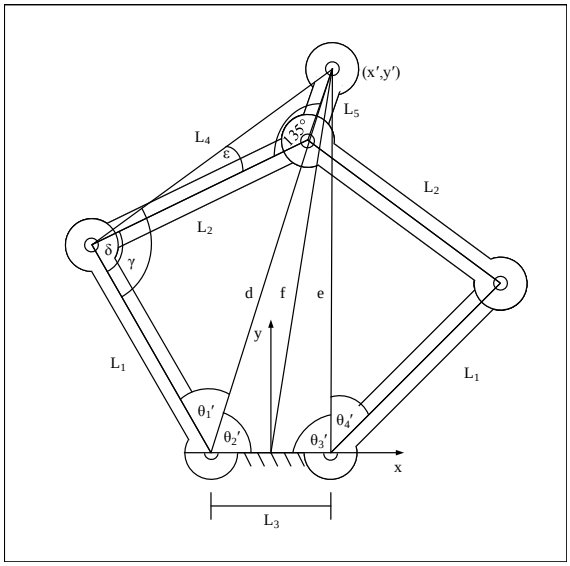

Figure 3: Complete Version of the Manipulator

d, e and f are found with the Pythagorean theorem:

$$f = \sqrt{x'^2 + y'^2}$$
$$d = \sqrt{\left(x' + \frac{L_3}{2}\right)^2 + y'^2}$$
$$e = \sqrt{\left(x' - \frac{L_3}{2}\right)^2 + y'^2}$$

Now, θ_2', θ_3' and δ can be computed using cosine rule:

$$\theta_2' = arccos\left(\frac{d^2 + (\frac{L_3}{2})^2 - f^2}{dL_3}\right)$$
$$\theta_3' = arccos\left(\frac{e^2 + (\frac{L_3}{2})^2 - f^2}{eL_3}\right)$$
$$\delta = arccos\left(\frac{{L_4}^2 + {L_1}^2 - d^2}{2L_4L_1}\right)$$

δ is the sum of ϵ and γ.

$$\gamma = \delta - \epsilon$$

That allows us to calculate some quantities of the simple kinematics model:

$$a = \sqrt{{L_1}^2 + {L_1}^2 - 2L_1L_2cos\,(\gamma)}$$
$$\theta_1 = arccos\left(\frac{a^2 + {L_1}^2 - {L_2}^2}{2aL_1}\right)$$
$$\theta_2 = \theta_1' + \theta_2' - \theta_1$$

Finally, we are able to find the position of the joint at which the two arms coincide:

$$y = a\sin(\theta_2)$$
$$x = a\cos(\theta_2) - \frac{L_3}{2}$$

Now, the methods from subsection 3.3 can be used to solve for θ_3 and θ_4.

3.5 Modeling the Servo Motors

Servo motors are motors that do not support continuous motion but can be driven to a desired angle precisely. Three cables with different colors connect to the servo motors. The black cable should be connected to ground and the red one to V_{DD} (roughly 5V). These two cables supply the servo motor with the necessary power. The third cable (white, yellow or orange) carries the control signal. The control is done by PWM. The servo motor expects to receive a pulse every 20 ms with a pulse width between 1 ms and 2 ms. The servo motor assumes its most positive position at the the pulse width of 1 ms and the most negative position at a pulse width of 2 ms as defined in the mathematical direction of rotation. The range in between these extrema can be assumed to be linearly covered, i.e. a pulse width of 1.5 ms should yield the position in between these positions. The function that maps the pulse width to an angular position of the servomotor is:

$$\text{Angle}(t_{\text{pulse}}) = \frac{t_{\text{pulse}} - 1\,\text{ms}}{2\,\text{ms} - 1\,\text{ms}} \cdot (\text{Angle}_{max} - \text{Angle}_{min}) + \text{Angle}_{min}, \qquad 1\,\text{ms} \leq t_{\text{pulse}} \leq 2\,\text{ms}$$

This is just a model of the movement of the servo motors. There are three causes making the actual behavior deviate from the model:

- A high torque forces the servo motors from leaving its desired position.
- The backlash of the gears in the servo motors adds an inaccuracy to the position.
- Non-linearities make the servomotor cover the range of movement not evenly.

Furthermore, when the pulse width modulation signal is created by a digital signal, a quantization error occurs. As you can see, the model makes some approximations. The inner of the servo motor is treated as black box. However, adding the details would bloat the model and the gain is questionable. The proposed model is precise enough to be useful but not so complex that it becomes cumbersome.

4. SOFTWARE ON THE ARDUINO

4.1 Overview over the software structure

Every program written with the Arduino IDE consists of the function `void init()` and `void loop()`. The `init()` function is executed once at startup and afterwards the Arduino keeps looping through the `loop()` function. That is, because as a typical microcontroller, the Atmel ATmega328P, does not have an operating system and can not just end a task. In addition, following major functions have been defined:

- `void lower()` lowers the marker.
- `void lift()` removes the marker from the surface.
- `void calculateServoMap(double x_prime, double y_prime, double* servoLsoll, double* servoRsoll)` calculates the values that should be written to the servo motors given an x-y-position.

- `void set_servos(double x, double y)` calls `calculateServoMap` and writes to the servo motors.
- `void go_to(double xto, double yto)` interpolates between the current position and the desired position of the manipulator and successively sets the position of the servo motors using `set_servos()` in order to achieve a smooth movement between the origin and the destination position.
- `void draw_{cross,circle}(double x_offset, double y_offset)` draws cross/circle to any position defined by a passed x-y-position.
- `draw_field()` draws the playing field.

The main functionality is implemented in the `loop()` function: The processor waits for an instruction coded in a byte sent from the master. Once it receives the command, it is executed and followed by sending an acknowledgment signaling that it has finished the execution.

4.2 Controlling the Servo Motors

The Arduino Uno has six PWM pins available. Very conveniently, the Arduino IDE already is equipped with a library `Servo`. This library allows us to instantiate objects of the type `Servo`. The most important functions on this object are `attach()` and `write()`. The `attach()` function assigns the Servo object to a GPIO pin. The `write()` maps an angle in the range of 0° to 180° to a pulse width and makes the attached GPIO pin output the according PWM. Inherently, a reachable range of 180° is assumed for the servo motor. However, our servo motors only are capable of spinning 150°. Signals that exceed the range of approximately 15° to 165° are simply ignored.

Additionally, the servo hat can only be placed onto the servo motor at certain angles. Therefore, the angle due to this offset also needs to be considered.

```
#define leftMax 161
#define rightMax 16
...
void calculateServoMap(...){
...
*servoLsoll = leftMax - (pi -
    (theta1+theta2) ) * 180/pi;
*servoRsoll = rightMax + (pi -
    (theta3+theta4) ) * 180/pi;
}
```

Listing 1: Finding the right angle

5. SOFTWARE ON THE COMPUTER

5.1 Computer Vision

The Python package OpenCV [CV] was used to process the images from the camera in order to register the user input. The basic steps to find the lines drawn on the whiteboard are to convert the image to grayscale and then to binarize it to black and white.

There are two main tasks for the vision: The first is to detect where the field is drawn, so the circle or cross drawn by the user can be assigned to a specific tile of the field. This is achieved by looking for the innermost contour of the lines drawn. Wrong detection is avoided by thresholding with the contours enclosed area. The corners of this inner field is then transformed so that the field coordinates are mapped to pixels.

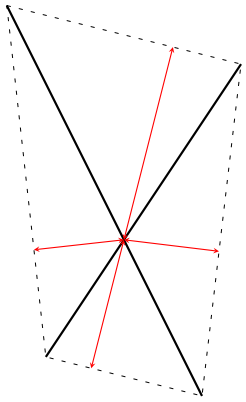

Figure 4: Convexity defects for a cross

The second task is to decide if the input drawn by the user is a cross or a circle. The convexity defects were used as the property of those signs to tell them apart. The convexity defects quantify the local maximum of the deviation of the contour to its convex hull as shown in Figure 4. For a circle or a different convex shape, there should be no convexity defects at all. Every convex shape will be detected as a circle consequently. This method of detection can therefore produce wrong rights, but the differentiation between circles and crosses is robust.

5.2 Tic-Tac-Toe AI

The simple and well known game Tic-Tac-Toe is already fully explored and algorithms to play the game optimally are available. Also, the relatively small state space of the board make exhaustive search for an optimal solution possible. The concept of the algorithm used in our implementation is nicely explained in [NSB]. Furthermore, [vdS] provides a similar implementation of the game in Python. Although it heavily relies on the functions only available in a GUI environment, it served as a useful reference.

Since the detailed explanation can be found in [NSB], only the principle is summarized here: A 3×3 2D-array represents the nine tiles in the field. On each move, the algorithm iterates over the tiles and when it encounters an empty tile, it recursively calls itself to simulate the progress of the game. Doing so, an altered board with the new move and another active player is passed. The break condition is either that one of the players has won or there is no free gap anymore. The original caller chooses the move that offers the best outcome with the minimum number of recursive calls in case of a win and the maximum number of recursive calls in case of a loss. It is a typical example of a minimax algorithm.

The described algorithm plays optimally, in a meaning that it is impossible to win against it. Also, in case a win can be enforced, the computer finds this optimal path to the win. It lacks an understanding for common human mistakes and assumes the opponent to play optimally himself. In order to offer some kind of a sense of achievement to the player, the algorithm is disabled with a probability of 15%

and a random move is made.

5.3 The Game Program

Having described each part separately, the only thing left is combining each part to form a robot that autonomously plays Tic-Tac-Toe. A superordinate loop sets up the playing field and reads the player's decision whether he wants to start or not. In order to completely decouple the game from an input on the computer, we made the design choice that the first player's sign is the cross and the second player's sign is the circle. With that definition, the decision whether the player or the computer should start the game is made by drawing either a cross ("I want to start!") or a circle ("The computer should start!") to the right of the playing field.

Once the setup is done, a subordinate loop is started that breaks after nine subsequent moves or when on one of the players has won.

For the serial communication with the Arduino, the library `pyserial` [PyS] has been used.

6. REFERENCES

[Atm] Atmel: *ATmega328P Manual*

[Ard] Arduino LLC: *Arduino UNO & Genuino UNO*

[Ser] Adafruit: *Micro Servo - MG90S High Torque Metal Gear*

[Joo] Johannes "Joo" Heberlein: *Plotclock*, Nuremberg(2014), published under the Creative Commons - Attribution license

[Geo] Markus Hohenwarter et al.: *Geogebra*, software published under a free, non-commercial license; source code published under the GNU General Public License

[cad] Autodesk: *Inventor*

[3D] SeeMeCNC: *ROSTOCK MAXâĎć V3 DESKTOP 3D PRINTER DIY KIT*

[CV] Itseez: *OpenCV*, software published under a BSD-license

[NSB] Never Stop Building LLC: *Tic Tac Toe: Under standing The Minimax Algorithm(2013)*

[vdS] Mauritz van der Schee: *Python TicTacToe with Tk and minimax AI*(2013)

[PyS] Chris Liechti:pyserial(2001)