Assignment 3 - MeArm

Shlishaa Savita & Benjamin Shields
University of Arizona, ISTA 303 - Creative Coding, with Peter Jansen

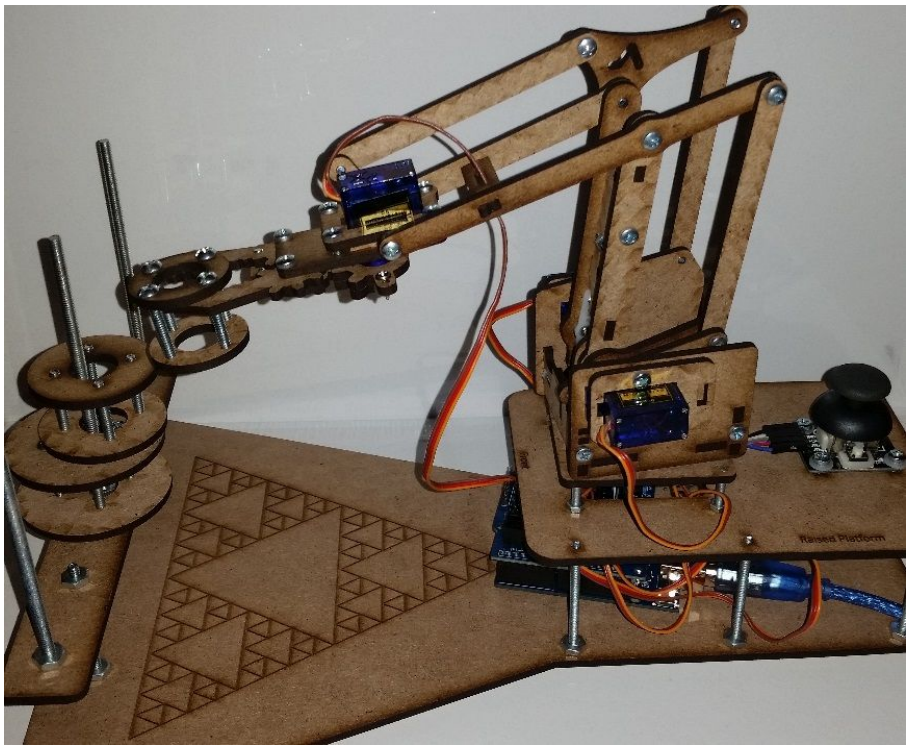

Fig. 1 - constructed MeArm robotically playing Towers of Hanoi

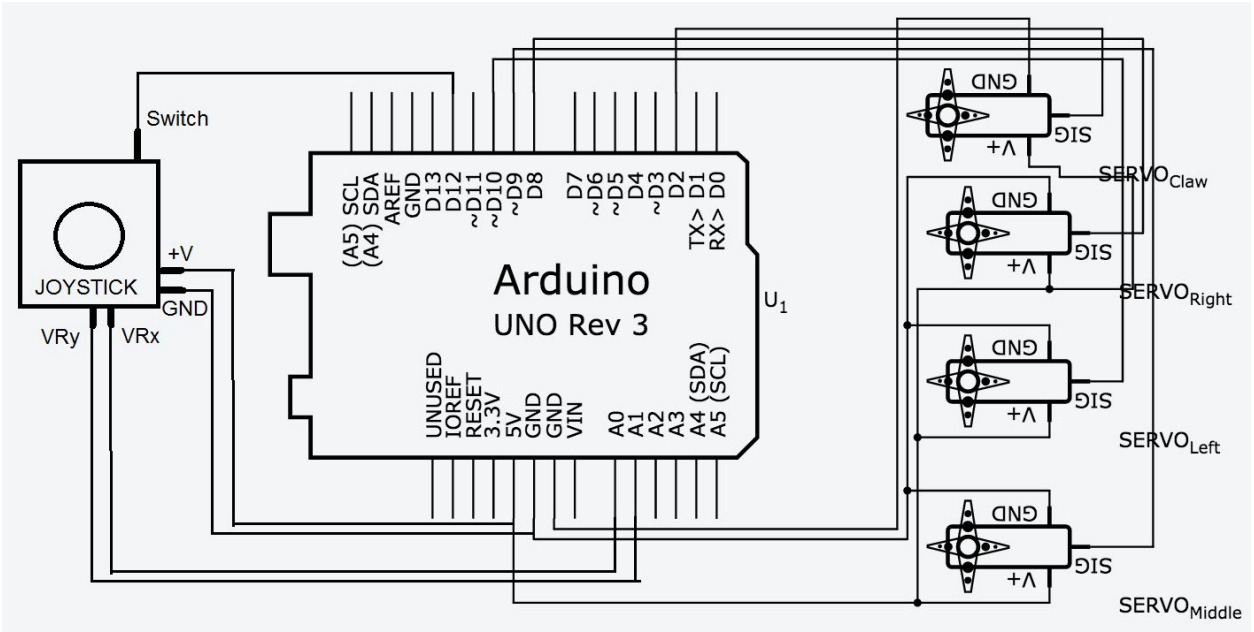

Fig. 2 - Circuit Schematic

Calculations (Kinematics):

The MeArm is controlled using four servos to move the claw in polar-coordinate space:

R = forward and backward position relative to the horizontal ground

Z = vertical height perpendicular to the ground

Theta = angle in degrees (see Fig. 2 below)

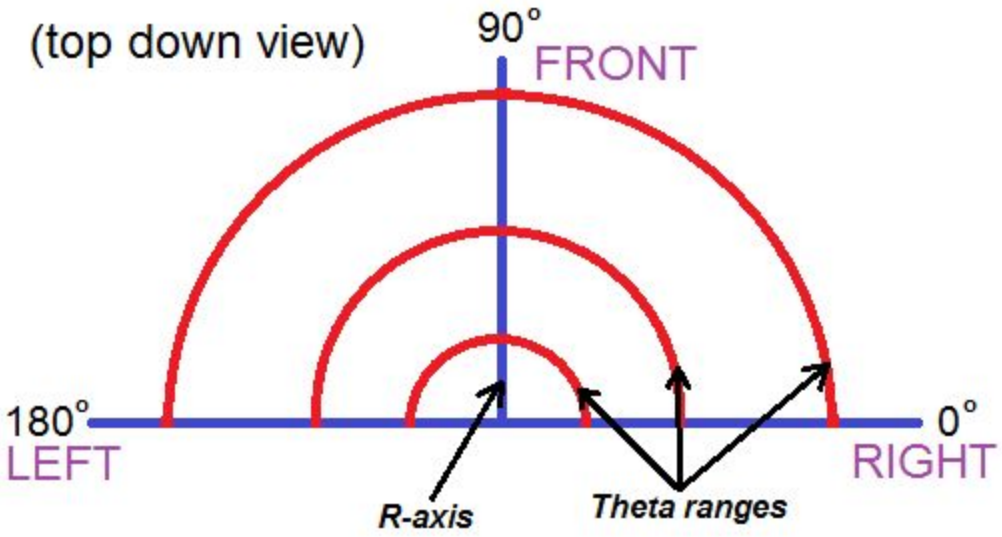

Fig. 3 - visualization of R and Theta coordinate space

The theta position is controlled directly by Servo3, on which the entire meArm box sits. I say directly because the function moveTheta( ) which controls it receives directions in degrees, which are directly written to Servo3 after two simple checks. These checks ensure that the servo will never be given a position which it cannot move to. Corresponding safety checks are also done in moveGripper( ), which controls Servo4, and moveRZ( ) which controls both Servo1 and Servo2.

Servo4 which controls the claw is as simple to control as Servo3. But moving in RZ - space requires trigonometry to calculate the inverse kinematics to move to a given position. This is shown in Fig. 3 below.

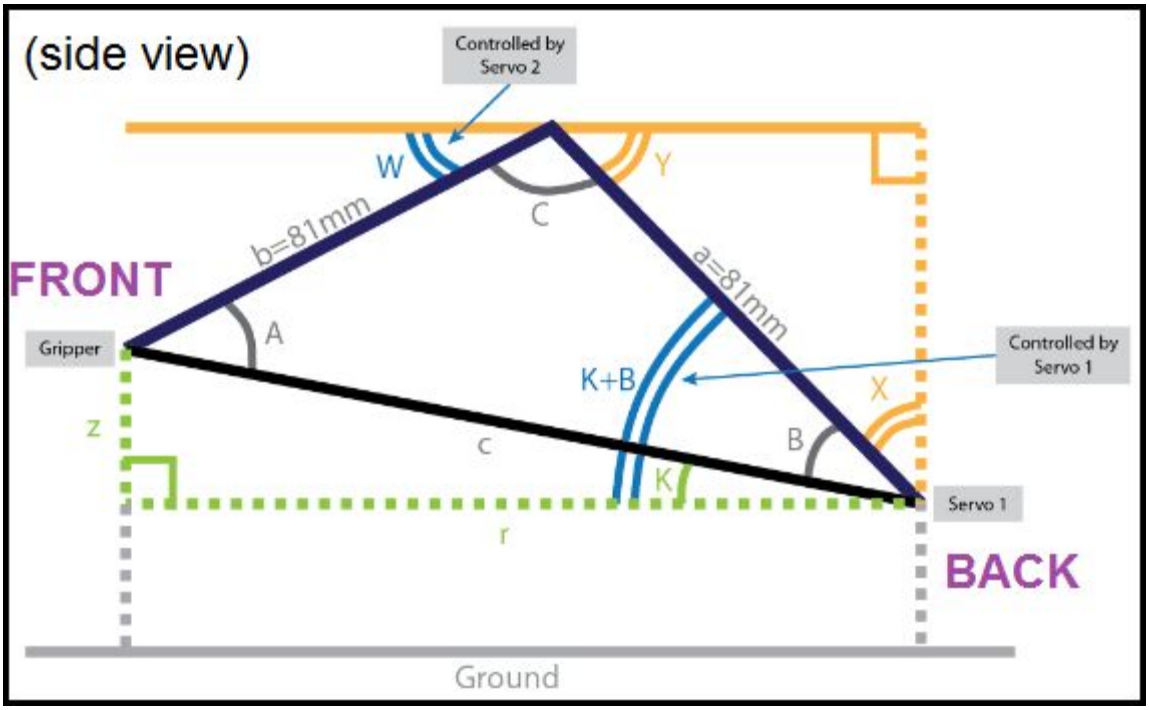

Fig 4. - inverse kinematics diagram

In the diagram, angle **K+B** is controlled by Servo1, which serves as the shoulder angle of the arm. Angle **W** is controlled by Servo2, and serves as the elbow angle of the arm. Due to the nature of where the MeArm has hinges and fixed screws, angle **A**, the wrist angle, automatically stays relatively horizontal in any R, Z position.

The shoulder angle is what is written to Servo1, and the elbow angle is written to Servo2. To solve them, several trigonometry theorems are needed to move to a given r and z position:

side c = square root(side a ^2 + side b ^2) *from Pythagorean theorem*
sides a (the bicep) and b (the forearm) are of fixed length on the arm
angle K = arctan(z / r) *from tan(angle) = opposite / adjacent*
angle A = angle B *from isosceles triangle, in this case sides a and b are of equal lengths*
angle B = arccos((a^2 + c^2 - b^2) / (2ac)) *from cosine law*
angle C = 180 - A - B *from interior angles of a triangle sum to 180 degrees*
angle shoulder = K + B *from the diagram*
angle X = 90 - shoulder *from the diagram, since the green and yellow dotted lines are perpendicular and therefore are at 90 degrees to each other*
angle Y = 180 - 90 - X *from the diagram, and the sum of triangle angles equals 180*
angle W = 180 - C - Y *from the diagram, and since W, C, and Y sum to a straight line*
angle elbow = angle W *as defined above and according to the diagram*

So after checking that the elbow and shoulder angles are valid, they are written to servos 1 & 2.

Tower of Hanoi:

"The Tower of Hanoi (also called the Tower of Brahma or Lucas' Tower, and sometimes pluralized) is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

With three disks, the puzzle can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is $2^n - 1$, where n is the number of disks."

Cited: "Tower of Hanoi." Wikipedia. Wikimedia Foundation, 30 Mar. 2017. Web. 31 Mar. 2017.

We thought this task was perfect, because computers/robots are so much better than humans at it!

Software Design Decisions:

The Arduino code used was designed with two key features in mind:

(1) Readability - for ease of development and feature extension by anyone

(2) Modularity - where functions are kept short, accomplish small goals that are easily understandable

To maximize readability, global variables are clustered and spaced by how they are conceptually used, and long descriptive variable names were created. Functions were also clustered by conceptual use: Servo, Joystick, Robotic, Arduino core (setup, serial interface, & loop). Finally, comments in plain english explain the purpose of less obvious code, and mark off clusters with subsection names.

Modularity began as a desired feature, but after we decided to implement algorithmic play of the game, it became a crucial feature. Because parameters for the MeArm's path to grab, lift, and move disks at three heights on three different pillars had to be hardcoded, the related functions had to be flexible and reusable to maintain readability, and to keep the length of the code reasonable. The key ways that this was accomplished were:

(1) Building basic robotic functions out of the simple serial-controlled functions
(2) Storing the position state variables and basic movement functions by address in two arrays, so that they can be accessed flexibly by index
(3) Using step (2), the robot movement is largely controlled by calls to a single function called autoMove(int directionIndex, int coordinate), which handles checking the validity of the coordinate parameter, and moving to it smoothly and accurately
(4) Building several simple functions that accomplish subtasks in the Towers of Hanoi: move to ready position, face a specific pillar, position claw next to a specific disk, grab a disk, lift disk off pillar, position claw above a specific pillar, lower and drop disk
(5) The functions in (4) were combined to create the final robotic function which could move any disk at any height to any other pillar. This function is called in the algorithmic solution of Towers of Hanoi, which uses simple recursion.

We are satisfied with how our code accomplishes our two key goals, and will proudly upload our program files as open source.

See the following section for the design decisions related to the Processing code.

User Interface Design Decisions:

We wanted to create a visual user interface through Processing that allowed users to select different modes of the claw machine we had implemented and be able to interact with the machine. The first aspect of this design was to create a game-like starting screen that showed the instructions for Tower of Hanoi and allowed the user to select (using keyboard input) from the following modes: Manual, Joystick, and Automatic.

Fig. 5 - User Interface via Processing (Start Screen)

Selecting Manual allows the user to input the R and Z coordinates, which is then sent back to the MeArm and is executed.

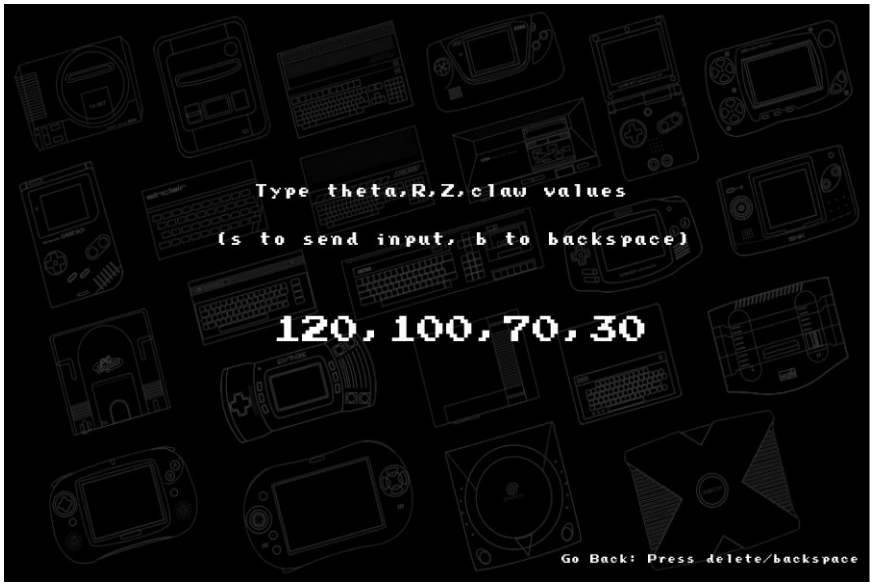

Fig. 6 - User Interface via Processing (Manual Mode Screen)

Joystick mode allows the user to have full control of the movement via the joystick.

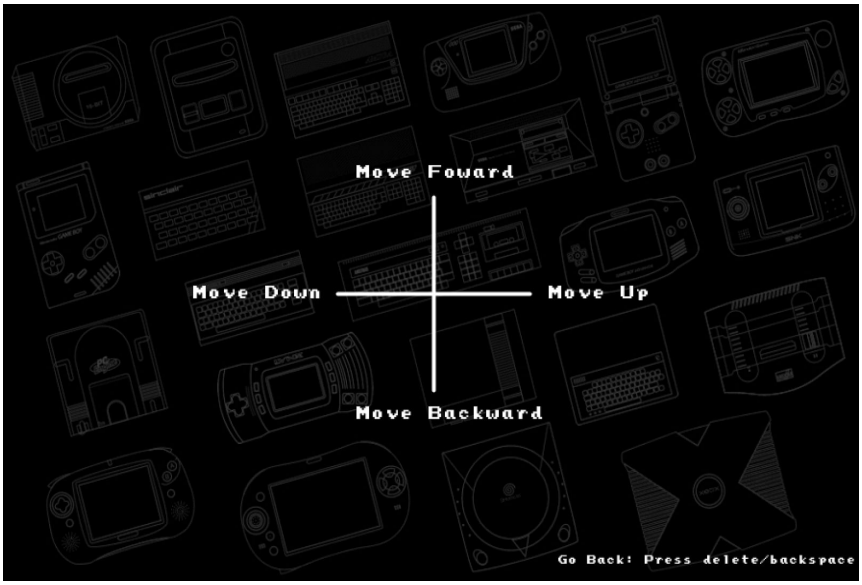

Fig. 6 - User Interface via Processing (Joystick Mode Screen)

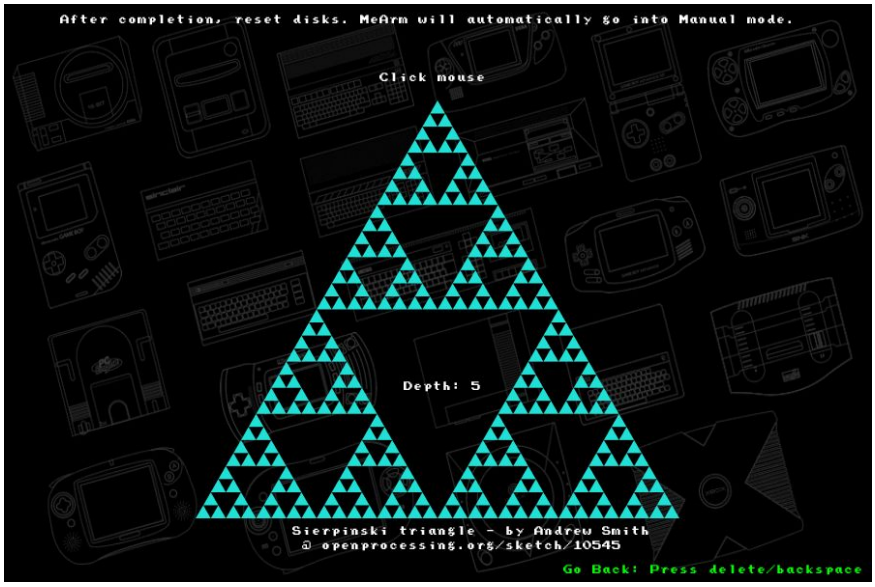

Fig. 7 - User Interface via Processing (Automatic Mode Screen)

And lastly, Automatic mode runs an algorithmic code that allows the MeArm to play the game fully robotically without input from the user. Selecting any one of these modes presents a different screen relative to the user selection. For example, when the user choses Joystick mode, a compass appears creating a visual map of which joystick mode is being used (claw/theta vs. R/Z servo movement). The individual direction of the compass will light up green if that direction is being executed via the joystick.

Hardware Design Decisions:

Our main design decisions consisted of making the Tower of Hanoi easily playable while keeping the design sleek and allowing the machine to be optimally efficient. To manage all the wires so that interference between micro servo wires, joystick wires, and the MeArm movement was kept at a minimum, we created a scaffold so the platform would be raised. This allowed us to store the Arduino, shield, and wires neatly away. On the raised platform, we wanted the joystick to be centered directly behind the MeArm to create a joyful experience and an intuitive user interface.

For the Tower of Hanoi aspect we used bolts to act as the pillars as this was the easiest and most stable method. We raised the pillars so it was level with the MeArm and equally spaced them so that our disks wouldn't come in contact with one another. The three differently sized disks were made by connecting two disks with four screws, allowing the MeArm to easily pick them up and transport them to their next location, regardless of how they are rotated on the pillar. Our last design choice was to make it more aesthetically appealing by imprinting the base layer with the Sierpinski triangle. The reason for this is because the Sierpinski triangle is a

visualization of the mathematical concepts related to solving the Tower of Hanoi. It is equivalent to the graph of movements taken when moving all disks from one pillar to another.

Note: The actual design of the laser-printed MeArm and assembly instructions was provided to us from: http://codebender.cc/user/MeArm.

Extensions:

I. Hardware Extension: Creating a raised platform so shield + wires are tucked away
II. Hardware Extension: Creating Tower of Hanoi
    A. Platform + board design (Sierpinski Triangle)
    B. Bolts + Disks (lasercut)
III. Software Extension: Creating three different modes which can be switched at any time
    A. Manual: type (theta,R,Z,claw) input on serial monitor (or on Processing)
    B. Joystick: allowing user to control arm through joystick
    C. Automatic: MeArm algorithmically + fully-robotically plays through game
IV. Software Extension: User Interface using Processing
    A. Welcome screen: User can choose which mode to play via keyboard input
    B. Joystick screen: visually maps joystick mode + direction
    C. Manual screen: with clever hacks to simulate textbox input in Processing
    D. Automatic screen: implemented open source code to allow the user to play with Sierpinski triangles while meArm is busy
    (credit to Andrew Smith @ https://www.openprocessing.org/sketch/105453)

YouTube link for video of extension:
https://youtu.be/md4RQzFbGR0

(which is public, and has public access download links for code and resource files)