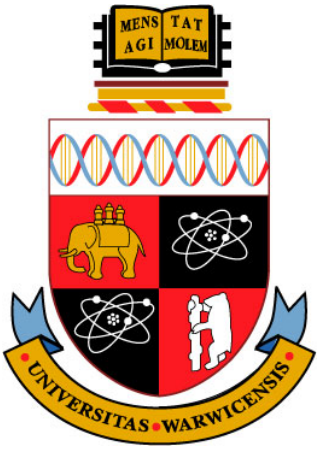

Robot Arm Project: Towers of Hanoi

CS132 Computer Organisation and Architecture

Harry Cunliffe

Manpreet Bahtra

Department of Computer Science

University of Warwick

2022-2023

Contents

1 Aims

The aim of this report is to investigate the function of a robotic arm device made up of five motors that move separately from one another. This is with the objective of using the robotic arm to complete the "Towers of Hanoi" puzzle, wherein the robot arm moves three stacked blocks from a leftmost position to a rightmost position in the same order without picking up more than one block at a given time. In more specific detail, this entailed first investigating the range of motion of each motor on the arm, then designing and implementing functions for moving blocks to three set locations from left to right of the arm. Finally, this report aims to show how the implementation of these functions was used to complete the Towers of Hanoi puzzle as described previously.

2 Design and Investigation

To understand the function of each motor, each motor was individually investigated. The use of each packet sent to and from the Dynamixel motor and use of different parameters was the most complex part of the design, as it required detailed knowledge of the hardware. Parameters such as loc_h and loc_l were increased and decreased to understand how they could be influenced to move the robot left or right. Additionally, functions such as initialLoc were created, which placed the first block in its initial position and was called at the start of each program. Besides, to test the robot's capabilties and to better understand the motors, straight and grabTwo functions were designed. Straight function picked up two blocks and placed them horizontally next to each other.

```
void straight(int connection, int locH, int locL){
    move_to_location(connection,1,locH,locL);  //Rotation
    move_to_location(connection,2,0x00,0x50);
    move_to_location(connection,3,0x02,0x2F);
    move_to_location(connection,4,0x02,0x40);
    wait_until_done(connection, 4);
    wait_until_done(connection, 3);
    wait_until_done(connection, 2);
}
```

GrabTwo function was designed to test if two blocks could be picked up at once.

```
void grabTwo(int connection, int locH, int locL, int Grab){
    move_to_location(connection,1,locH,locL);  //Rotation
    wait_until_done(connection, 1);
    move_to_location(connection,2,0x01,0x0E);
    wait_until_done(connection, 2);
    move_to_location(connection,3,0x01,0x80);
    wait_until_done(connection, 3);
    move_to_location(connection,4,0x01,0x22);
    wait_until_done(connection, 4);
    if(Grab == 0){
    move_to_location(connection, 5, 0x01, 0x20);
    }else{
        letGo(connection);
    }
    wait_until_done(connection, 5);
    lift(connection);
}
```

Furthermore, hexadecimal values were investigated to understand when each motor performs a specific function. For example, motor 5 grabs the block at the value of 013E. To make the robot arm move horizontally, motor 1's values were changed. Reducing the value made the robot arm move to a rightwards position, and increasing the value moved the robot arm to a leftwards position. Motors 2, 3 and 4 changed the vertical position of the robot arm. Increasing the hexadecimal values for these motors made the robot arm move backwards, whilst reducing the values made it move to a forwards position. The purpose of

motor 5 was to expand and contract so that it could grab and release the block. Increasing the hexadecimal value for motor 5 made the grabber expand, hence allowing for the block to fit inside the grabber. Additionally, minimising the hexadecimal value resulted in the grabber contracting, so that the block could be lifted and moved to a different position.

3 Implementation

As detailed in the aims,implementing a solution to the Towers of Hanoi puzzle required a litany of functions to be created first using the information given through the design and investigation. This was broken down into a few different sections of implementation that were then tackled individually leading up to a final solution.

- The robot must be able to lift blocks up from the stack, as well as place them back down.
- The robot should be able to grab blocks from the bottom, middle and top of a stack of blocks.
- The robot must be able to rotate between each of the three locations that blocks can be placed: Right, Middle and Left.

Combining these different functions is then enough to implement a simple solution to the puzzle. From the design it was clear that lifting and placing blocks is almost entirely the work of the fifth motor. The hex values necessary to properly clasp blocks were already investigated and designed previously, and so could be clearly and concisely converted into a function.

```
1. void grab(int connection){
2.     move_to_location(connection, 5, 0x01, 0x3e);
3. }
```

Dropping blocks just required returning the motor to its previous state:

```
1. void letGo(int connection){
2.     move_to_location(connection, 5, 0x01, 0xB8);
3.     wait_until_done(connection, 5);
4.
5. }
```

One issue that was faced during this implementation was the robot often moving to its next function before the block was fully dropped. This led to the block being placed in the wrong location. The solution to this, as shown above was to wait for the fifth motor to finish its motion before the robot could move onto its next command.
Lifting the blocks once grabbed was another simple function to implement, as it only required use of the second motor, using values previous designed.

```
1. void lift(int connection){
2.     move_to_location(connection, 2, 0x01, 0x5E);
3.     wait_until_done(connection, 2);
4.
5. }
```

Rotation only required use of the first motor, and so was fairly simple to implement, however it did take meticulous trial and error to find the exact hexadecimal values for the different points the arm would rotate to. For this we ended up with:

Left – 23E
Middle – 1FF
Right – 1C2

Grabbing blocks from different levels was the most complex part of the implementation, as it required the use of all of the different motors in tandem. Investigating each motor previously allowed a basic understanding of how they affected the arm, and thus how to move the arm into specific positions. This implementation, however, still required thorough trial and error testing, wherein the arm was moved small amounts repeatedly and the results were recorded until values of movement for each motor were found that allowed each block to be lifted.

Once these values had been recorded, a function was made for lifting and placing blocks at the top of a stack:

```
1. void topBlock(int connection, int locH, int locL, int Grab){
2.     move_to_location(connection,1,locH,locL);  //Rotation
3.     wait_until_done(connection, 1);
4.     move_to_location(connection,2,0x01,0x36);
5.     move_to_location(connection,3,0x01,0x94);
6.     move_to_location(connection,4,0x00,0xEB);
7.     wait_until_done(connection, 4);
8.     if(Grab == 0){
9.         grab(connection);
10.     }else{
11.         letGo(connection);
12.     }
13.     wait_until_done(connection, 5);
14. }
```

This function would serve as a framework for the middle and bottom blocks also. Besides the connection, the function takes 3 integers, locH, locL and Grab. locH and locL are the hex values for the first motor which allows for rotation. Grab is either set to 1 or 0 and serves as a check for whether the arm should pickup a block or drop a block.

After the creation of similar functions for middle and bottom blocks, a Towers of Hanoi function could finally be implemented. By first researching out the seven moves needed to complete the puzzle most efficiently, it was then trivial to implement this into a final function.

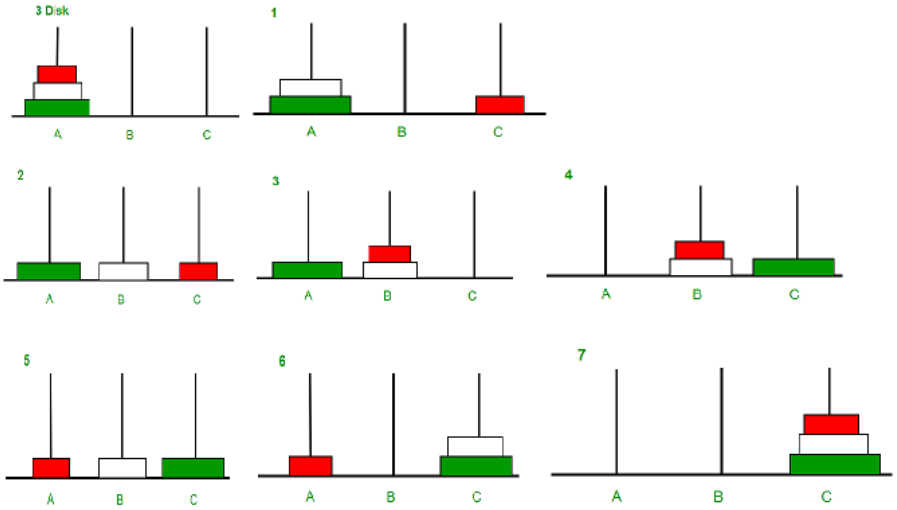

Figure 1: Reference image for puzzle solution
[1]

Each move followed the same code pattern:

```
1. topBlock(connection,0x01, 0xFF, 0); //Middle
2. lift(connection);
3. wait_until_done(connection, 2);
4. bottomBlock(connection,0x02, 0x3E, 1); //Left
5.
6. lift(connection);
7. wait_until_done(connection, 2);
```

This lifts the first block, places it into the next position, and then lifts the arm again ready for the next movement. Combining seven of these led to the final implementation of the Towers of Hanoi solution as seen in the code.

4 Results

The aim of the report as defined previously was to investigate the function of the robot arm and show how it could be used to implement a solution to the given puzzle. This is a result that is clearly reached within the report. Each motor was investigated, and the way changing the hexadecimal inputs of each motor affects the arm was clearly documented. Therefore, through the use of many clearly designed and defined functions, the robot arm can complete the "Towers of Hanoi" puzzle from left to right without any errors or mishaps.

5 Reflection

If more time was provided, straight function could have been expanded upon so that blocks are placed in the reverse order to how they were picked. As an extension to the project, if more blocks were provided, a Towers of Hanoi with 5 blocks could have been implemented along with Towers of Hanoi from a rightmost position to a leftmost position. Attempts were made to increase the speed of the robot arm. To do this, limits for speed in the limits.txt file was changed, which didn't reproduce any results hence proved futile. Secondly, maximum voltage of the robot was increased, since it corresponded to the speed, however in doing so the robot was non-operational. Therefore, speed and maximum voltage were changed to their original values. Lastly, sleep value for the robot was reduced from 1,000,000 to 900,000 microseconds, to reduce latency and increase efficiency of the robot. One issue that was faced during experimenting with the sleep values was that robot often moved to its next function before the current function was fully implemented. Moreover, it knocked down all the stacked blocks while doing so, hence providing a limitation to reducing the sleep value to only 900,000 microseconds.

References

[1] GeeksforGeeks. C program for towers of hanoi. https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi/, 2014.