Solving the 3-tier Hanoi Tower problem using ABB RobotStudio® Software

Ahmed Abdulgader Salim Assagaf, MAZEN MOHAMED MAHROUS, Omar Awad Abdelhadi Mahmoud, Omar Walaa Hasaan Mohamed Hassan Elmaasri

Faculty of Electrical Engineering, Universiti Teknologi Malaysia

FKE, UTM, 81310 Skudai, Johor

***Abstract*\—** The design and analytical simulation of a pick and place system utilising ABB RobotStudio® software is applied in this report.The purpose of this report is to solve the 3-tier Hanoi Tower problem using ABB RobotStudio® software. Rules and concept of solving the 3-tier Hanoi Tower were detaily explained in this report. By seflearned the software, we were able to program and set the movement of the ABB Robot.

***Index Terms*--pick and place, ABB RobotStudio software, Hanoi Tower, rules, concept**

I. Introduction

Our group had been given a task to solve a 3-tier Hanoi Tower problem using ABB RobotStudio® software. In the Hanoi Tower problem, a stack of 3 blocks labelled with number 1, 2 and 3 on each block will be rearranged at another predefined area according to a planned sequence. The goal is to determine the shortest running time of the given task under nominal operating speed of the robot.

The Tower of Hanoi is a mathematical puzzle or game made up of three blocks, each of which has a unique number. For example, block 1, block 2, and block 3 are all distinct numbers. The blocks are arranged in decreasing numerical order on one section, with the smallest at the top. There are three sections in total, two of which were emptied at the start of the game. The purpose of the task is to move the stacks of the 3 blocks to another section while following the rules below:

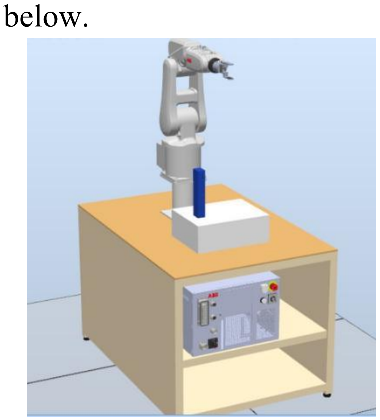
1. Only one block may be moved at a time.
2. Each move consists of taking the upper block from one of the stacks and placing it on top of another stack or on an empty section.
3. No block may be placed on top of a block that is smaller than it.

By programming the ABB RobotStudio® software to move the blocks using planned trajectory motion, this problem can be solved.

In this report we will be able to understand the concept of Hanoi Tower and solve it using ABB Robot, able to program and simulate an industrial robot using RobotStudio, able to understand and implement a trajectory planning motion, able to control the motion of the robot and able to determine the shortest running time for the robot to solve the task

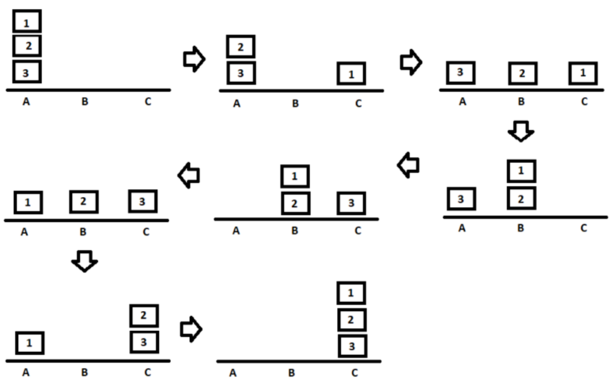
II. Procedures

1. New Solution with Station and Virtual Controller is opened.
2. IRB120_3kg 0.58m were selected and a station was created.
3. Library IRB120_3_58_G_01 were selected.
4. All the tools from the Libraries_UTMstation were imported into the ABB RobotStudio's station and all the tools were arranged as in figure below.



5. In the layout browser, *MyGripper* was dragged into the robot IRB120 and the gripper was attached onto the robot.

6. Programming the basic station, workobject was created.
7. Motion programming was created by creating the targets on each block.
8. Target oriented was adjusted by rotating the target approximately +180 degrees around theX axis and approximately +180 degrees around the Z axis.
9. *View Tool at Target* and *View Robot at Target* was used to preview how the tool and robot will be oriented around the targets. If the target is reachable, then the robot will automatically jump to the target.
10. By creating empty *paths*, the targets were added to the paths that we created. There were 7 paths for the robot's movement as shown below.



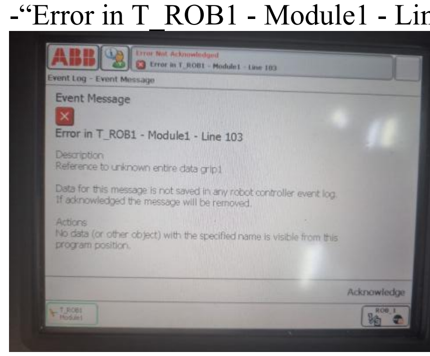
11. In each path, every step was created by setting the positions or location from its starting point until its end point depending on where we want the path to be.
12. *Status Bar* was set according to our preferences whether in changing its variable motion which is liner, joint or circular. Speed, zonedata, tools and the work object being used were also set.
13. To be sure that the robot can reach the target, a new target which we will use as start position and approach/depart target were added.
14. *Auto Configuration* was done to set the start configuration and then RobotStudio will calculate the configuration for the rest of the instructions in order to get as smooth movements of the robot axes as possible.
15. *Station* was synchronised to *RAPID*.
16. Simulation was set up for it to be able to start a simulation. Position of the robot was defined for it to start the execution.
17. Every change in *RAPID* was synchronised to *Station* and every change in *Station* was synchronised to *RAPID.*
18. *Smart Component* - *Gripper* was created foreach block.
19. Base components were added under each *Smart Component* - *Gripper.*
20. Internal signals were defined in the component and I/O signals were connected in the Virtual Controller. I/O connections were also defined.
21. I/O connections werewere set up between the *Smart Component* and the *Virtual Controller.*
22. Grip and ungrip functions were declared in the *RAPID* editor and applied changes.
23. The *Control Panel* was set to auto mode and the *Motor's Button* was pushed.
24. *RAPID* was synchronised to *Station* and the simulation can be played.

III. Data and Results

During the lab, we encountered several problems and error messages from the robot's software due to using an older version of the robot that was not functioning optimally. As a result, we were only able to complete the first few steps of the project. The main issues included:

- Compatibility Issues: The old version of the robot firmware caused several compatibility errors with the ABB RobotStudio® software. This required troubleshooting and attempts to apply patches to bridge the compatibility gap.
- Performance Lag: The software experienced significant lag, affecting the robot's response time and precision. This made it challenging to execute the planned movements accurately.
- Error Messages: Frequent error messages interrupted the workflow, including:

  -"Error in T_ROB1 - Module1 - Line 103."



  -"Two channel fault, RUN CHAIN."

-"The operation was rejected by the controller safety access restriction mechanism."

Despite these challenges, we managed to complete the initial setup and some basic movements. Figures illustrating the encountered errors and partial progress are attached below.

Simulation Results

Due to the software lag, we could only finish the first few steps, which included setting up the station, attaching the gripper, and defining the initial targets. The subsequent steps required more precise movements, which were hindered by the lag and errors in the software.

Despite these setbacks, the initial simulation indicated that the robot could potentially complete the 3-tier Hanoi Tower task if the software and firmware were fully compatible and responsive. The video of the partial simulation can be viewed here: Robot Studio Simulation…..

RobotStudio SImulation Results

The simulation results using RobotStudio was successfull. we have followed the same steps as in II. Procedure to create 2 different simulation using combination of all movements, the difference in both simulations was the different coordinates, Movements, speed and choice of movement.

comparing both simulation we have chosen the one with the shortest time, which takes 29.2 seconds to finish.

the simulation can be watched at :
https://youtu.be/43ik-pjxENo

The code can be found below at Appendix.

IV. Discussion

What is moveL, moveJ, and move C?

- **Move J :** This instruction is referred to as motion by joint movement and is used to move the robot's Tool Centre Point(TCP) toa position or location but does not require the robot to move along a linear path. The robot can take any path as long as it gets to the programmer's specified point. This move instruction is vital for starting its motion process as it really does not matter how the robot gets to its start point. The disadvantage of this move type is that the robot is not intelligent enough to recognise obstacles along the path it chooses to take and as such, can collide with objects or even people when getting to its start point.
- **Move L:** This move type is most commonlyused when designing the path of operation of the robot. With a move L instruction, the robot's TCP moves in a straight line from one point to the next specified point. This move instruction can only be used when the robot is carrying out its main task.
- **Move C:** The Move C instruction is used tomove the robot's TCP along a circular point. This instruction is very vital when theoperation to be carried out by the robot

involves it taking a curved path. While other move instructions take one positionargument, the Move C instruction takes twoposition arguments. For a semi-circular path, the first position argument is the centre of the semi-circle while the second position argument is the end point of the semi-circle. To program a complete (360 degree) circular path, two Move Cinstructions are needed, with each position argument taking a quarter point radius of the circle.

- The major difference between all the move instructions is that Move J moves in a non-linear path, Move L moves in a linear path and Move C moves in a circular path. With regards to when they are used, Move Jis used as the first move instruction when starting the program, Move L and Move C are used when the robot is carrying out its main task.

Programming in RAPID

An example of a RAPID code is shown below:

MoveL * v100,fine,toolPenWObj:=wobj0;

- Move L is the move instruction that tells the robot to move linearly to a point.
- * Is the X,Y,Z coordinates the robot programmed to move to. The value of coordinates are based on the workobject indicated. It is to be noted that at any location the robot is, if a move instruction is added, the coordinate at that point is what isrecorded in * but can be modified if it was the wrong coordinate.
- Fine refers to the Zonedata of that instruction. Fine means the robot should getto that exact position.
- V100 is the velocity or speed therobot is to move with. It is also a variable and it is set by the user. The values from the menu list in the flex pendant range from v5mm/s to above v1000mm/s
- toolPen is the Tool Object specified by the user. It stores the details of the TCP. For this project, a pen as the tool object, hence, the variable name, toolPen.
- WObj:=wobj0 specifies the workobect being used. For this project, several workobjects were created for the several different robot tasks

Based on the functions stated above, we can clearlydesign our robot movement according to our preference and we can determine the speed of the robot in order to execute the problem and solution. In our case, in achieving the fastest speed of the robot, we combine the moveJ, moveL, and moveC function and eliminate all the unnecessary extramovement.

V. Conclusion

To conclude our laboratory session, the objectives of the laboratory were achieved. The action of ABB IRB 120 by using ABB RobotStudio® software has been constructed and successfully tested and demo. The action of trajectory linear and circular is clearly defined. As a result, the experiment was successfully completedby the group.

We are now able to define the concept of Hanoi Tower which refers to the process of moving the stack of blocks to another predefined area while following the rules of Hanoi Tower and solve the Hanoi Tower problem using ABB RobotStudio software using the functions that we have learned.

We are also able to understand and implement the trajectory planning motion. Planning atrajectory motion lets us determine the shortest pathand the shortest time required to run the robot and solve the problem.

VI. References

ABB AB Robotics Products. “RobotStudio Courseware 5.60.” 2012. Ahlam Forqan, et al. “RobotStudio Simulation solving Hanoi Tower for Robotics Lab.”

Youtube, Ahlam Forqan, 5 June 2020, https://www.youtube.com/watch?v=4oe7RK-L8Y&t=215s&ab_channel=AHLAMFORQANA17KE3003.

Mohamad Hafis Izran Bin Ishak UTM. “Additional Materials PBLab Robotik Industrial Robots Student Pack ABB pick & place exercise.”

Mohamad Hafis Izran Bin Ishak UTM, 30 April 2020, https://drive.google.com/drive/folders/1r5KngUGKCpgBXFjEBBtG3IEVEk0xXv-

Mohamad Hafis Izran Bin Ishak UTM UTM. “Labsheet Robotics - Hanoi Tower problem.”

Mohamad Hafis Izran Bin Ishak UTM, 30 April 2020, https://drive.google.com/drive/folders/1r5KngUGKCpgBXFjEBBtG3IEVEk0xXv-

“Industrial Robot Programming: ABB IRB 1200.” UK Essays, 23 September 2019, https://www.ukessays.com/essays/computer-science/industrial-robot-programming-abbirb-1200.php

VII. Appendix

```
!***********************************************************
!
! Module:  Module1
!
! Description:
!   <Insert description here>
!
! Author: User
!
! Version: 1.0
!
!***********************************************************


!***********************************************************
!
! Procedure main
!
!   This is the entry point of your program
!
!***********************************************************
```

```
PROC main()
    Home;
    Path_start;
    Path_Block1;
    Path_Block1place;
    Path_Block2;
    Path_Block2place;
    Path_Block1place2;
    Path_Block3;
    Path_Block3place;
    Path_Block1place3;
    Path_Block2place2;
    Path_Block1place_4;
    Path_start;
    Home;
ENDPROC
```

```
PROC Path_Block1()
    ungrip_block;
    MoveJ Appr_pBlock1,v500,fine,tMyGripper\WObj:=Wobj_Block1;
    MoveJ pBlock1_10,v300,fine,tMyGripper\WObj:=Wobj_Block1;
    grip_block1;
    MoveJ Appr_pBlock1,v500,fine,tMyGripper\WObj:=Wobj_Block1;
ENDPROC

PROC Home()
    MoveJ HomePos,v500,fine,tMyGripper\WObj:=wobj0;
ENDPROC

PROC Path_start()
    MoveJ pStartPos,v500,fine,tMyGripper\WObj:=Wobj_StartPos;
ENDPROC

PROC Path_Block1place()
    MoveJ Appr_pBlock1place,v500,fine,tMyGripper\WObj:=Wobj_Block1place;
    MoveJ pBlock1_10place,v300,fine,tMyGripper\WObj:=Wobj_Block1place;
    ungrip_block;
    MoveJ Appr_pBlock1place,v500,fine,tMyGripper\WObj:=Wobj_Block1place;
ENDPROC
```

```
    PROC grip_block1()
        WaitTime 0.5;
        SetDO grip1,1;
    ENDPROC

    PROC ungrip_block()
        WaitTime 0.5;
        SetDO grip1,0;
    ENDPROC

    PROC grip_block2()
        WaitTime 0.5;
        SetDO grip2,1;
    ENDPROC

    PROC ungrip_block2()
        WaitTime 0.5;
        SetDO grip2,0;
    ENDPROC
    PROC grip_block3()
        WaitTime 0.5;
        SetDO grip3,1;
    ENDPROC

    PROC ungrip_block3()
        WaitTime 0.5;
        SetDO grip3,0;
    ENDPROC
```

```
        ungrip_block2;
        MoveJ Appr_pBlock2,v500,fine,tMyGripper\WObj:=Wobj_Block2;
        MoveJ pBlock2_10,v300,fine,tMyGripper\WObj:=Wobj_Block2;
        grip_block2;
        MoveJ Appr_pBlock2,V500,fine,tMyGripper\WObj:=Wobj_Block2;
    ENDPROC

    PROC Path_Block2place()
        MoveJ Appr_pBlock2place,v500,fine,tMyGripper\WObj:=Wobj_Block2place;
        MoveJ pBlock2_10place,v300,fine,tMyGripper\WObj:=Wobj_Block2place;
        ungrip_block2;
        MoveJ Appr_pBlock2place,v500,fine,tMyGripper\WObj:=Wobj_Block2place;
    ENDPROC
    PROC Path_Block1place2()
         ungrip_block;
        MoveJ Appr_pBlock1place_2,v500,fine,tMyGripper\WObj:=Wobj_Block1place_2;
        MoveJ pBlock1_10place_2,v300,fine,tMyGripper\WObj:=Wobj_Block1place_2;
        grip_block1;
        MoveJ Appr_pBlock1place_2_4,v1000,fine,tMyGripper\WObj:=Wobj_Block1place_2;
        MoveC Appr_pBlock1place_2_5,Appr_pBlock1place_2_2,v500,fine,tMyGripper\WObj:=Wobj_Block1place_2;
        MoveJ pBlock1_10place_2_2,v300,fine,tMyGripper\WObj:=Wobj_Block1place_2;
        ungrip_block;
        MoveJ Appr_pBlock1place_2_2,v300,fine,tMyGripper\WObj:=Wobj_Block1place_2;
    ENDPROC
```

```
    PROC Path_Block3()
        ungrip_block3;
        MoveJ Appr_pBlock3,v500,fine,tMyGripper\WObj:=Wobj_Block3;
        MoveJ pBlock3_10,v300,fine,tMyGripper\WObj:=Wobj_Block3;
        grip_block3;
    ENDPROC
    PROC Path_Block3place()
        MoveJ Appr_pBlock3place1,v500,fine,tMyGripper\WObj:=Wobj_Block3place;
        MoveC Appr_pBlock3place2,Appr_pBlock3place4,v500,fine,tMyGripper\WObj:=Wobj_Block3place;
        MoveJ pBlock3place3,v300,fine,tMyGripper\WObj:=Wobj_Block3place;
        ungrip_block3;
        MoveJ Appr_pBlock3place4,v500,fine,tMyGripper\WObj:=Wobj_Block3place;
    ENDPROC
    PROC Path_Block1place3()
        ungrip_block;
        MoveJ Appr_pBlock1place_3_1,v500,fine,tMyGripper\WObj:=Wobj_Block1place_3;
        MoveJ pBlock1_10place_3,v300,fine,tMyGripper\WObj:=Wobj_Block1place_3;
        grip_block1;
        MoveJ pBlock1_10place_3,v300,fine,tMyGripper\WObj:=Wobj_Block1place_3;
        MoveC Appr_pBlock1place_3,Appr_pBlock1place_3_2,v500,fine,tMyGripper\WObj:=Wobj_Block1place_3;
        MoveJ pBlock1_10place_2_3,v500,fine,tMyGripper\WObj:=Wobj_Block1place_3;
        ungrip_block;
    ENDPROC
```

```
    PROC Path_Block2place2()
        ungrip_block2;
        MoveJ Appr_pBlock2place2,v500,fine,tMyGripper\WObj:=Wobj_Block2place2;
        MoveJ pBlock2_10place2,v300,fine,tMyGripper\WObj:=Wobj_Block2place2;
        grip_block2;
        MoveJ Appr_pBlock2place2_2,v500,fine,tMyGripper\WObj:=Wobj_Block2place2;
        MoveJ pBlock2_10place2_2,v300,fine,tMyGripper\WObj:=Wobj_Block2place2;
        ungrip_block2;
        MoveJ Appr_pBlock2place2_2,v500,fine,tMyGripper\WObj:=Wobj_Block2place2;
    ENDPROC
    PROC Path_Block1place_4()
        ungrip_block;
        MoveJ Appr_pBlock1place4,v500,fine,tMyGripper\WObj:=Wobj_Block1place_4;
        MoveJ pBlock1_10place4,v300,fine,tMyGripper\WObj:=Wobj_Block1place_4;
        grip_block1;
        MoveJ Appr_pBlock1place4_2,v500,fine,tMyGripper\WObj:=Wobj_Block1place_4;
        MoveJ pBlock1_10place4_2,v300,fine,tMyGripper\WObj:=Wobj_Block1place_4;
        ungrip_block;
        MoveJ Appr_pBlock1place4_2,v500,fine,tMyGripper\WObj:=Wobj_Block1place_4;
    ENDPROC
ENDMODULE
```