Professor: Mr. Joseph Chretien Jr. **Student:** Matthew Terrance Carlos Boyea

Introduction

Mike Cicco, president and CEO of FANUC America, has claimed that FANUC has sold over 750,000 robots globally[1], more than any other industrial robotics supplier in 2021. FANUC may be considered the largest industrial robot manufacturer in the world. It follows that skilled and innovative developers which work with the FANUC America tools are in demand. The purpose of this lab is to improve processes by which students may develop skills for programming FANUC machines using the insight of a student's independent study practices.

The student began this research project with an interest in the machine and without any FANUC machine training. Under the guidance of the professor, he self-studied the *FANUC America Corporation HandlingTool Operations & Programming Student Manual* with an LR Mate 200iD 4s machine arm which was powered by the R-30iB Mate controller. After becoming comfortable with the machine, he designed a manufacturing simulation project which would swap the locations of some cylinders between a set of slots. During this process, he constructed a repeatable procedure which would suffice for other students' development of simple programs for a FANUC machine in the classroom while keeping industry concerns in mind. The procedure is outlined in this lab report.

Procedure

To design any engineering solution efficiently and effectively, it may help to begin by clarifying the problem. The definition of the problem, the constraints of the operation, and the procedure to come upon a solution may all be specified before teaching the FANUC machine. This approach may increase safety and repeatability of the teaching process.

The problem may be defined by a state of "precondition" and a state of "postcondition."

The precondition state is a concise description of everything within the robot's environment that must be considered to describe the robot's actions. This may include the state of any objects and the state of the robot itself. In this case, the precondition was defined as: "A block with slots S1, S2, S3, and S4 is placed in front of the bot. Cylinders C1, C2, and C3 are placed in slots S1, S2, and S3 respectively."

The postcondition state is a description of the change to the precondition state. This may include the change to the state of the given objects and the state of the robot. In this case, the postcondition was defined as: "Cylinders C1, C2, and C3 are moved to slots S3, S1, and S2 respectively."

A procedure may be defined as a series of actions which reach the postcondition state via interaction with items outlined in the precondition state. It is important to consider the constraints of the operation while developing the procedure. The procedure for this program was defined as: "Get C1 in S1. Store C1 in S4. Get C2 in S2. Store C2 in S1. Get C3 in S3. Store C3 in S2. Get C1 in S4. Store C1 in S3."

Together, the precondition, postcondition, and procedure may be referred to as the algorithm. Following this definition of the algorithm, a new Teach Pendant program for the robot was created with the name "SWAP_CYLNDRS." The algorithm was written into this program as Remarks. This left the file in the following state:

swap_cylndrs code

```
      !PRECONDITION:
      --A BLOCK WITH SLOTS S1, S2, S3, AND S4 IS PLACED IN FRONT OF
THE BOT. CYLINDERS C1, C2, AND C3 ARE PLACED IN SLOTS S1, S2, AND S3
RESPECTIVELY.
      !POSTCONDITION:
      --CYLINDERS C1, C2, AND C3 ARE MOVED TO SLOTS S3, S1, AND S2
RESPECTIVELY.

      !GET C1 IN S1
      !STORE C1 IN S4
      !GET C2 IN S2
      !STORE C2 IN S1
      !GET C3 IN S3
      !STORE C3 IN S2
      !GET C1 IN S4
      !STORE C1 IN S3
```

Code documentation via remarks in each file may be helpful to develop when programming a robot. While it is not necessary for the code to run, it is very helpful to have this documentation when a developer is studying or editing this file. With in-code documentation and guidance, they're able to follow easier what each line of the file does without testing the program. Furthermore, planning and documenting the code before writing it can improve the workflow of developing the application by reducing the likelihood of mistakes which could cost time and threaten the safety of the machine, equipment, or personnel.

Following this, the necessary User Frame(s), Tool Frame(s), and Jog Frame(s) should be identified and taught to the robot. This is a standard procedure and will not be covered in this report.

It is especially important to teach relevant Frames before any movement commands are taught to the robot. When a command is taught to a FANUC machine, the User Frame and Tool Frame it is taught with are used to encode the movement of the robot. If an invalid User Frame or Tool Frame is active when a movement command is run by a program, the robot will be unable to perform that command.

Because of this property of movement commands, it is good practice to always set the correct User Frame and Tool Frame programmatically before any movement commands are called by a program. In the example of the SWAP_CYLNDRS program, the following lines are added to the code:

swap_cylndrs code

```
{...}
      !POSTCONDITION:
      --CYLINDERS C1, C2, AND C3 ARE MOVED TO SLOTS S3, S1, AND S2
RESPECTIVELY.

      !SET FRAMES
      UFRAME_NUM=0
      UTOOL_NUM=2
      !GET C1 IN S1
      !STORE C1 IN S4
{...}
```

Next, the robot must be brought to a state where it is prepared to execute the algorithm. This includes the state of the machine arm and any of its accessories such as End of Arm Tooling. This process should consider the objects in the environment which may be in the way of the path to the start location. Because a FANUC program may be run from any initial position, it is important to write strong safeguards to prevent it from colliding with objects within its workspace when the program begins. When the algorithm ends, the robot is prepared to run the next algorithm.

In the case of the SWAP_CYLNDRS program, the consequence for colliding with an object outlined in the precondition is that the block will be knocked over. Because of this, for the sake of brevity, the procedure to prepare the bot simply moves the arm to a high location which will likely avoid collision with the block but doesn't necessarily ensure it. The procedure to ready the bot for the next algorithm is just as simple; moving it to the initial position will suffice.

The following lines are added to the code:

swap_cylndrs code

```
{...}
      !SET FRAMES
      UFRAME_NUM=0
      UTOOL_NUM=2
      !PREP BOT
J     P[1] 100% CNT100
      RO[3]=ON
      !GET C1 IN S1
      !STORE C1 IN S4
{...}
      !GET C1 IN S4
      !STORE C1 IN S3
      !READY BOT
J     P[1] 100% FINE
      RO[3]=ON
```

In a manufacturing setting, a more sophisticated safeguard against collisions with the environment is important; the consequence of collisions could be to harm the machine arm, the building, or even human personnel.

There are many approaches to managing this concern. For example, the robot could move itself vertically to a height which is above all objects in its workspace, then navigate to the starting position of the program. Alternatively, the machine could detect when it's in a dangerous position and refuse to run the program, displaying a message to the user which asks them to safely jog the machine to a location which is within the specifications of the program. For machines with visual peripherals, the machine could avoid objects procedurally.

After this, the procedure of the algorithm should be taught to the robot as a set of programmed commands. During this process, the programmer may consider reusing previously taught positions and implementing a time-efficient path between target locations and tasks. Any repeated logical procedures may be extracted to an external program or macro and called within this code (there was no use case for this feature in this project). Because the procedure of the algorithm has already been designed, the teaching process to produce this code was straightforward.

The following lines are added to the code:

swap_cylndrs code

```
{...}
      !PREP BOT
J     P[1] 100% CNT100
      RO[3]=ON
      !GET C1 IN S1
L     P[2] 250mm/sec FINE
      RO[4]=ON
L     P[1] 250mm/sec FINE
      !STORE C1 IN S4
J     P[3] 100% CNT 100
L     P[4] 250mm/sec CNT100
L     P[5] 250mm/sec FINE
      RO[3]=ON
L     P[3] 250mm/sec FINE
      !GET C2 IN S2
J     P[6] 100% CNT100
L     P[7] 250mm/sec FINE
      RO[4]=ON
L     P[6] 250mm/sec FINE
      !STORE C2 IN S1
J     P[1] 100% CNT100
L     P[2] 250mm/sec FINE
      RO[3]=ON
      !GET C3 IN S3
J     P[8] 100% CNT100
L     P[9] 250mm/sec FINE
      RO[4]=ON
L     P[8] 250mm/sec FINE
      !STORE C3 IN S2
J     P[10] 100% FINE
J     P[11] 100% FINE
J     P[6] 100% CNT 100
L     P[7] 250mm/sec FINE
      RO[3]=ON
L     P[6] 250mm/sec FINE
      !GET C1 IN S4
J     P[3] 100% CNT100
L     P[4] 250mm/sec CNT100
L     P[5] 250mm/sec FINE
      RO[4]=ON
L     P[4] 250mm/sec FINE
L     P[3] 250mm/sec CNT100
      !STORE C1 IN S3
J     P[8] 100% FINE
L     P[9] 250mm/sec FINE
      RO[3]=ON
      !READY BOT
J     P[1] 100% FINE
      RO[3]=ON
```

While writing this code, the movement should be periodically tested to ensure that it runs at a slow speed; after this rough draft of movement code is outlined, it should be tuned to operate with consistency at 100% of the robot's speed in its fastest movement mode. This process may show some flaws with precision and speed such that values which manipulate how the robot moves should be tuned.

An example issue follows: when the robot's compressed air tank was reached a low pressure, it took too long to open or close the end of arm tooling grippers. When the arm would move to pick up or put down a cylinder, the robot would push the block over or release a cylinder at the wrong location. To help prevent this, a short wait time was added after the cylinders open or close to ensure the arm could consistently perform its next action while the air compressor was unplugged.

Finally, redundant code (`RO[3]=ON`) was identified in the end of the algorithm under `!READY BOT`, which was removed. This program could be improved by creating an OPEN_GRPR and CLOSE_GRPR macro or program with the above time waiting process to further reduce the redundancy and improve the readability of the code.

The program was completed at the following state:

swap_cylndrs code

```
      !PRECONDITION:
      --A BLOCK WITH SLOTS S1, S2, S3, AND S4 IS PLACED IN FRONT OF
THE BOT. CYLINDERS C1, C2, AND C3 ARE PLACED IN SLOTS S1, S2, AND S3
RESPECTIVELY.
      !POSTCONDITION:
      --CYLINDERS C1, C2, AND C3 ARE MOVED TO SLOTS S3, S1, AND S2
RESPECTIVELY.

      !SET FRAMES
      UFRAME_NUM=0
      UTOOL_NUM=2
      !PREP BOT
J     P[1] 10% CNT100
      RO[3]=ON
      WAIT  .05(sec)
      !GET C1 IN S1
L     P[2] 250mm/sec FINE
      RO[4]=ON
      WAIT  .05(sec)
L     P[1] 250mm/sec FINE
      !STORE C1 IN S4
J     P[3] 20% CNT100
L     P[4] 150mm/sec CNT100
L     P[5] 250mm/sec FINE
      RO[3]=ON
      WAIT  .05(sec)
L     P[3] 150mm/sec FINE
      !GET C2 IN S2
J     P[6] 25% CNT100
L     P[7] 250mm/sec FINE
      RO[4]=ON
      WAIT  .05(sec)
```

```
L     P[6] 250mm/sec FINE
      !STORE C2 IN S1
J     P[1] 10% CNT100
L     P[2] 250mm/sec FINE
      RO[3]=ON
      WAIT  .05(sec)
      !GET C3 IN S3
J     P[8] 10% CNT100
L     P[9] 250mm/sec FINE
      RO[4]=ON
      WAIT  .05(sec)
L     P[8] 250mm/sec FINE
      !STORE C3 IN S2
J     P[10] 10% FINE
J     P[11] 7% FINE
J     P[6] 10% CNT 100
L     P[7] 250mm/sec FINE
      RO[3]=ON
      WAIT  .05(sec)
L     P[6] 250mm/sec FINE
      !GET C1 IN S4
J     P[3] 25% CNT100
L     P[4] 150mm/sec CNT100
L     P[5] 250mm/sec FINE
      RO[4]=ON
      WAIT  .05(sec)
L     P[4] 250mm/sec FINE
L     P[3] 150mm/sec CNT100
      !STORE C1 IN S3
J     P[8] 20% FINE
L     P[9] 250mm/sec FINE
      RO[3]=ON
      WAIT  .05(sec)
      !READY BOT
J     P[1] 10% FINE
```

Sources Cited

[1] - https://www.fanucamerica.com/news-resources/fanuc-america-press-releases/2021/07/01/global-automation-leader-fanuc-announces-production-of-750-000th-robot