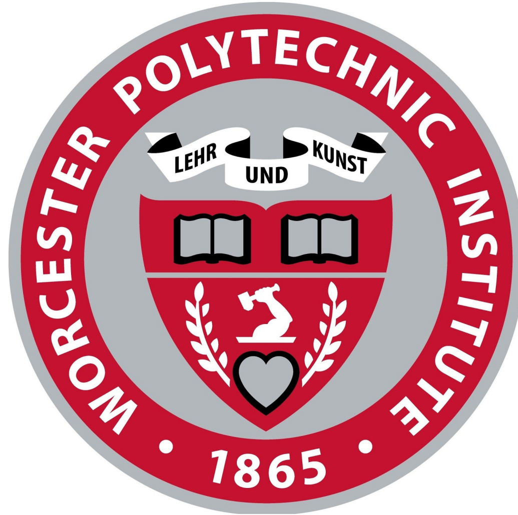

Worcester Polytechnic Institute

RBE4815–Final Project Report

Dembski, Clayton John
Rangel, Steven
van Rossum, Floris
White, Adam

supervised by:
Professor: Zhi Li

Executive Summary

Table of Contents

Executive Summary **2**

Table of Contents **2**

Introduction **3**

Goal Statement **3**

Task Specifications **3**

Workspace Setup **3**

Keyboard 4

Electrical Circuit 4

Gripper Design **5**

Programming **5**

Movement 5

Timing 5

Results and Discussion **5**

Conclusions **5**

Bibliography **6**

Appendices **6**

Introduction

The RBE4815: Industrial Robotics class final project is intended as a capstone for the entire class. Team 3 successfully created a challenge and completed it. The IRB1600 ABB Robotic Arm was used to play songs such as “Fur Elise” and “Mary Had a Little Lamb” on a Casio Electronic Keyboard. Many problems and obstacles, such as timing and configuration errors, were encountered along the way and solutions to those problems were found. The team became more knowledgeable about industrial robotics concepts such as forward and inverse kinematics, singularities and RAPID code. All of these subject can be utilized and applied in the future as they are now more familiar.

Goal Statement

The goal of this project is to utilize the ABB IRB1600 robotic arm to play musical notes in the form of a song on a Casio SA-76 keyboard autonomously and with ease. Additionally, to allow for easy programming of songs in a modular and efficient manner.

Task Specifications

Playing musical pieces can be difficult, even for a person. Music revolves around proper timing and speed. Therefore the task specifications for this project revolve around these topics. The target was to get the robot to play a song as fast as possible. This meant that we had to put the robot into maximum speed and plan our timing accordingly. The goal for our robot was to play a song at about 80 beats per minute (BPM). Although this would sound slow, it would still make a song recognizable. Additionally we wanted to make the song sound as pleasant as possible, this meant reducing things that make additional noise. We also wanted to make programming the songs as easy as possible. Hardcoding different songs was an option, but that would take a significant amount of time if a new song needed to be played. When programming a new song the user should be able to call some method that simplifies the process.

Workspace Setup

The following paragraph discusses the different elements of our mechanical and electrical setup in order to complete this project. The electronic keyboard was positioned flat on the workspace table within comfortable reach of the ABB robot arm. No jig was used to hold the keyboard in place, but a simple manual calibration sequence was used to determine whether the keyboard was positioned correctly. A variable, current-limiting power supply was used behind the arm and keyboard to supply 12V DC limited to 2.5A to the solenoid. A relay that could support the 24V signal from the robot’s control circuitry was mounted onto the arm and wired to the solenoid and power supply. As shown in figure 1, the solenoid was directly mounted

perpendicular to the robot’s end effector and a rubber finger was fabricated and attached to the plunger of the solenoid.

Figure 1: Final Setup of Workspace

Keyboard

The keyboard used for this project was a Casio SA-76 keyboard. It features 44 different keys and about 100 different tones. Because we were afraid of the possibility of the keyboard breaking due to an accident with the robot we chose this cheap keyboard, it was only around 50 dollars. The keyboard was placed in the robot workspace flat and facing the robot. The configuration of the robot when directly over the keyboard was (1,-2,-1,0). This configuration allowed us to encounter no singularities when moving from one end of the keyboard to the other, other configurations did cause problems.

Electrical Circuit

The electrical circuit that had to be created in order to get this project to completion was the biggest set-up challenge. The IRB1600 robot digital out ports that were usable were D652_10_DO1 and D652_10_DO2. These could be toggled by a RAPID program or the user

via the Flex Pendant buttons. However, these digital out ports emitted a signal of 24V and about 0.5A. Our solenoid required 12V and about 2.5A to use. This meant that even if we transformed the voltage from 24V to 12V, we would likely not have enough power to actuate the solenoid. A plausible solution to this problem is to create a seperate circuit to power the solenoid, with a seperate power supply, but actuated by the IRB1600 controller digital out. Initially, we ordered a 12V, 3A power supply from Amazon. However, this item was incorrectly listed and was only able to supply 3 amperes for short peaks 3 seconds apart. Using this power supply was not an option as we had to press keys at least once every second in order to reach 80 BPM. Therefore, we utilized a 12V DC power supply to deliver power to the solenoid if toggled. We set the current limit to 2.5A as this created a sufficiently strong enough push to press down a key on the keyboard, but not break it. The solenoid we were using was rated for a maximum force of 55N. We also purchased a relay from amazon. The relay had a switching voltage of 24V and a maximum current of 5A, so this was adequate for our project. As you can see in the diagram below, the robot controller digital out toggled the relay and connected the power supply to the solenoid causing it to actuate.

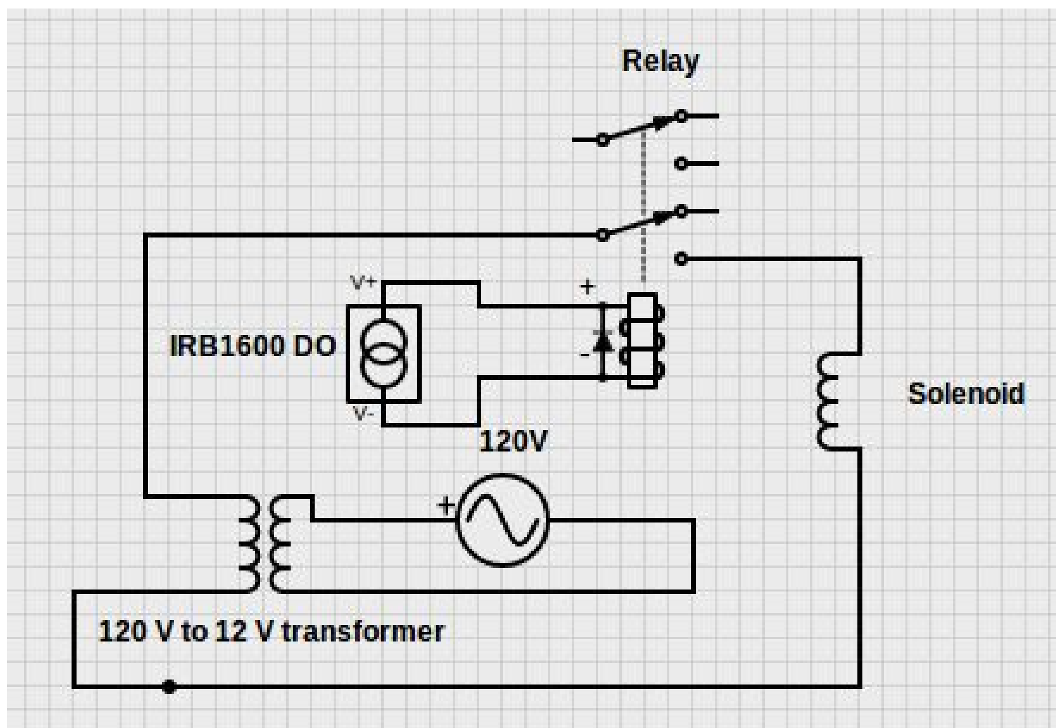

Figure 2: Electrical Diagram of the Project Setup

Gripper Design

Designing and manufacturing the gripper was by far the most challenging mechanical aspect of this project. We had to properly design it to withstand the load of the solenoid and avoid breaking the keyboard. We decided that we wanted to add a solenoid on to the end effector because we assumed this would allow us to press keys faster. This actuation would

allow us to press a key on the keyboard without having to move all the joints on the keyboard. This would save us time as we now had to only actuate the solenoid in order to actuate a key. An image of the end effector is included below.

Figure 3: The final end effector

We consider the solenoid part of the end effector as it is attached onto the end of the robot. The solenoid runs at 12V and ~2.5 A. The maximum force it can produce is around 55N. However, because we scaled down the current this was much less. Still, the force was significant enough to probably break the keyboard. Therefore we installed multiple different dampers in the system. We had padding and a spring around the top of the solenoid. This can be seen as the white foam on the top. This white packaging protection mainly served as noise reduction as the solenoid made a significant amount of noise when actuated. The spring served to retract the solenoid once it had been actuated. The bouncy ball on the end of the solenoid acted dampened the force on the keyboard. The bouncy ball had to be extremely skinny as it had to press between two keys. Because we didn't want to change our z height between key presses, as it would cost more time, we pressed significantly harder on the black keys as compared to the white keys. The bouncy ball had to accommodate for this by compressing more for black keys than white keys.

Programming

Simulation

The start of the programming in RobotStudio with RAPID code was done with the simulation software. We assumed that we would depend significantly more on the simulation side of RobotStudio than we actually did. CAD models of the end effector and keyboard were made and introduced into RobotStudio. We made targets for the high and low keys on the keyboard using the offset of the end effector tool derived from the STL of the end effector solenoid. This worked relatively well until we found the need to add the bouncy ball and dampening to our end effector. This changed the depressed and nondepressed offset of our end effector. Because redoing the CAD model would have taken to long, we manually jogged the robot into the appropriate position and found the new required z offset for the end effector. We implemented this change in the code. We took on this approach as well for determining the algorithm and hardcoded offsets for the movement. It was just way more accurate and easier to do for our application as compared to simulation. Simulation did prove extremely useful in order to start the project and understand how long it would take for the robot to go from the high key to the low key. Additionally it provided a simple catch of stupid coding mistakes such as syntax or offset errors. The simulation software also caught many of the singularity errors we ran into early on in the project. By manually jogging the robot to a position we were able to avoid these singularities by finding a working configuration. This working configuration was then input into the RAPID code.

Movement

The movement of the arm was controlled by the command GoToKey(keynum), where the valid keys were numbers from 1 to 44 (starting from F) and -1. If the number was -1, the arm would remain in the current position for a single beat and “Hold” the current note. Else, the arm would lift off the key, move to the next key, and, on the beat, press onto the given key. Each key’s position was attempted first to be found algorithmically, but was ultimately found through hard code. The following is the direct code for the algorithm.

```
PROC GoToKey(num keyNum)
    !Some delicious RAPID code that may or may not do things of important nature to do with the robotics


    ! Do some calculations to determine the location of the position
    ! X: Variable
    ! Y: -1183.2 mm
    ! Z: 125.1 mm

    VAR num distWW := 20.31;
    VAR num distBW := 22.78/2;
    VAR num distance := 0;

    IF keyNum MOD 12 > 0 and keyNum MOD 12 < 7 THEN
        distance := distBW * (keyNum MOD 12);
    ENDIF

    IF keyNum MOD 12 = 7 THEN
        distance := distBW * 6 + 2*distBW;
    ENDIF

    IF keyNum MOD 12 > 7 AND keyNum MOD 12 < 12 THEN
        distance := distBW * 6 + 2*distBW + ((keyNum MOD 12) MOD 7) * distBW;
    ENDIF

    distance := distance + Trunc(keyNum / 12) * (10 * distBW + 4 * distBW);
    TPWrite "Distance calculated: " \num:=distance;
    MoveL [[-distance + 26.0,-1183.2,125.1],[0.55982,0.38937,-0.58583,0.43795],[-2,-1,2,1],[9E+09,9E+09,9E
ENDPROC
```

As the key values should ideally repeat every 12 values, if the key is from 0 to 6, offset the key by the distance from one black key to one white key. If 7, offset by the net distance from 0 to 6, and 2 times the distance from a black key to a white key. And so on. This code ultimately failed due to imprecision in the keyboard. On black keys that were grouped in 3's, the first black key was left justified, the middle key was centered, and the right key was right justified. In groups of two, the first was left justified and the second was right justified. This meant that almost half of the keys in each span of 12 would need non uniform values to perform correctly. Finding these unique measurements would have taken the same time, if not longer, than hard coding the position of each key, and hard coding insured the position would be correct each time. Because of this, we decided to jog the robot to each position and collect the values for the position.

Timing

One of the most important factors for a Piano playing robot is playing in time with the current chosen song. The problem with this is that the time the arm takes to move from key to key is dynamic: the arm could take only .1 seconds to move to a location adjacent to the current one, and .5 seconds to move up an octave. To solve this, we decided to set up an interrupt trap routine that would perform during the move-key phase of the GoToKey function. The trap routine was set to be as computationally small as possible, so that multiple interrupts would not trigger at once. In short, at the start of go to key, a counter would be reset to 0. Every .1 seconds, the counter would be increased by 1. In this way, the time the arm took to move could be found by using the formula counter *.1. Knowing this, the time till the next beat could be

calculated by the following formula: WaitTime = TimePerBeat-ArmTravelTime, or, in our instance WaitTime(.5-counter*.1).

Discussion & Results:

By the end of our final lab session, we were successfully able to play Fur Elise and Mary Had A Little Lamb at 76 beats per minute. The system was open ended enough that one could convert a song to perform by treating 1 as the F key one octave below middle c and creating a list of numbers to iterate through.

The end effector acted as expected. We had to pay significant attention to the wait statements and timers in order to ensure that the solenoid was actuated at the appropriate time. If the wait time was too short, the solenoid would be depressed before the arm stopped moving. This could cause damage to the keyboard as it dragged across it. If this happened it also caused the keyboard to be dragged which meant that we had to readjust and realign it with the high and low joint moves. The keyboard was in a significant amount of danger while we were testing the code. We tried to remove the keyboard from the workspace whenever we tested code for the first time, but, still, we managed to break the keyboard. It was not, however, when we were testing the code but rather when manually jogging the robot. Although the torque limit engaged and the robot froze, it was too late. An electronic panel in the robot had already snapped and the higher keys on the keyboard were no longer functional. This didn't prevent us from completing the project as this failure only affected the highest 8 notes. This was the reason that such as cheap keyboard was used, the possibility of this happening was realized by us.

We expected the the limiting factor for the project to be the direct speed of the arm, instead of the solenoid. The arm itself could move to press keys for a tempo of about 85 BPM. This was what we expected our cap to be, as the robot will not move faster than 100% speed. The solenoid, however, caused problems due to the strength of the spring used to press the keys. When the power to the solenoid was cut, the end effector would take up to .3 seconds to lift back up, and when powered would take roughly .1 seconds to fire. If we were to move the arm before these timings were met, we would end up hitting multiple keys, shifting and misaligning the keyboard, or, in the worst case, breaking the keyboard further.

The electronic circuit performed as expected. After the failed attempt with the Amazon power supply as listed in the section above, the laboratory power supply worked. In order to obtain the specifications of the electrical properties of the digital out port on the robot we performed research online. Still, we were unable to find the voltage of the ports. So, we connected a multimeter across the terminal and found out that the voltage was 12V. Whenever the solenoid of the default gripper used in previous labs was actuated it read a current of 0.5A. With this knowledge we were able to order the appropriate switch for our circuit. We first searched digikey for some kind of transistor, but we found that it was difficult to find a 24V transistor. So we opted for a relay instead. The relay we found was rated for 5A at 24V, so we

assumed it would work for our application. The relay we ordered worked fine when wired up to the robot.

Figure 4: Relay used in our electrical circuit

Conclusions:

By the end of this project, we learned an effective and efficient way to program an industrial robot arm to play a musical keyboard. Using a solenoid to press the keys meant we simplified the arm's required motion to just one dimension, and hard-coding the targets allowed us to better compensate for varying offsets between keys. There are a number of things we could have done differently, including making a custom mount for the solenoid, would would have allowed us to mount it parallel to the end of the arm, rather than perpendicular. Possible improvements range from making a fixture to support multiple solenoids or using multiple robot arms to support polyphonic playing, or implementing MIDI uploading/streaming for avoid the need to hard-code transcribed sheet music. The most notable parameter of the robot's playback that could use improvement is simply its speed. To perform at human-like speeds, the robot would need to move multiple times faster than we were safely able to achieve during this project. In order to achieve this with the hardware we used, significant improvements would need to be made to the code to insure that higher-speed solenoid actuation could take place safely without unwanted collisions. Overall, the team has learned a great deal about using industrial robot arms in delicate applications.

Contributions:

van Rossum, Floris	Robot Studio, Demo Setup
Dembski, Clayton John	RAPID code, Demo Setup
Rangel, Steven	Hardware, CAD
White, Adam	Hardware, Electrical

Appendix A: Code:

```
MODULE Module1
    CONST jointtarget
High:=[[-87.6,76.6,-15.7,23.3,-63.6,173.4],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST jointtarget
Low:=[[-112.2,80.9,-29,-4.8,-52.6,187.4],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
       CONST robtarget
Target_10:=[[244.89,-1043.63,202.7],[0.000177171,-0.360322418,0.93282749,-0.000773518]
,[-1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
       CONST robtarget
Target_20:=[[233.31,-1043.63,202.7],[0.000177171,-0.360322418,0.93282749,-0.000773518]
,[-1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
       !***********************************************************
       !
       ! Module:  Module1
       !
       ! Description:
       !   <Insert description here>
       !
       ! Author: cjdembski
       !
       ! Version: 1.0
       !
       !***********************************************************


       !***********************************************************
       !
```

```
! Procedure main
!
!   This is the entry point of your program
!
!***********************************************************
VAR intnum timeint;
VAR intnum timecount;
PROC main()


timecount := 0;
CONNECT timeint WITH incCount;
ITimer 0.1, timeint;

GoToKey(36);
GoToKey(35);
GoToKey(36);
GoToKey(35);
GoToKey(36);
GoToKey(31);
GoToKey(34);
GoToKey(32);
GoToKey(29);
GoToKey(-1);
GoToKey(-1);
GoToKey(20);
GoToKey(24);
GoToKey(29);
GoToKey(31);
GoToKey(-1);
GoToKey(-1);
GoToKey(24);
GoToKey(28);
GoToKey(31);
GoToKey(32);
GoToKey(-1);
GoToKey(-1);
GoToKey(24);
GoToKey(36);
GoToKey(35);
GoToKey(36);
GoToKey(35);
GoToKey(36);
GoToKey(31);
GoToKey(34);
GoToKey(32);
GoToKey(29);
GoToKey(-1);
GoToKey(-1);
GoToKey(20);
```

```
        GoToKey(24);
        GoToKey(29);
        GoToKey(31);
        GoToKey(-1);
        GoToKey(-1);
        GoToKey(22);
        GoToKey(32);
        GoToKey(31);
        GoToKey(29);

        Retract;


        ENDPROC

    PROC Path_10()
        MoveAbsJ Low, v500, fine, tool0\WObj:=wobj0;
        MoveAbsJ High, v500, fine, tool0\WObj:=wobj0;
    ENDPROC

        PROC GoToKey(num keyNum)
        timecount := 0;
        !SetDO D652_10_DO1, 1;
        !Some delicious RAPID code that may or may not do things of important nature to
do with the robotics

        ! Do some calculations to determine the location of the position
        ! X: Variable
        ! Y: -1183.2 mm
        ! Z: 125.1 mm

        ! Retract the pusher

        ! Wait a bit
        IF keyNum = -1 THEN
                WaitTime(0.5 - timecount * 0.1);
        ELSE
                Retract;
                IF keyNum = 1 THEN
                MoveL
[[26.79,-1183.27,140.6],[0.55992,0.389101,-0.585975,0.437862],[-1,0,1,1],[9E+09,9E+09,
9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

                ELSEIF keyNum = 2 THEN
                MoveL
[[14.85,-1183.27,140.6],[0.559918,0.389098,-0.585979,0.437863],[-1,0,1,1],[9E+09,9E+09
,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

                ELSEIF keyNum = 3 THEN
```

```
            MoveL
[[5.31,-1183.27,140.6],[0.559924,0.389092,-0.58598,0.437858],[-1,0,1,1],[9E+09,9E+09,9
E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 4 THEN
            MoveL
[[-8.21,-1183.27,140.6],[0.559916,0.389099,-0.585978,0.437865],[-1,0,1,1],[9E+09,9E+09
,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 5 THEN
            MoveL
[[-18.97,-1183.27,140.6],[0.559917,0.389102,-0.585977,0.437861],[-1,0,1,1],[9E+09,9E+0
9,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 6 THEN
            MoveL
[[-30.26,-1183.27,140.6],[0.559916,0.3891,-0.585977,0.437865],[-2,0,1,1],[9E+09,9E+09,
9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 7 THEN
            MoveL
[[-40.11,-1183.27,140.6],[0.559916,0.389103,-0.585977,0.437862],[-2,0,1,1],[9E+09,9E+0
9,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 8 THEN
            MoveL
[[-54.40,-1183.26,140.6],[0.559916,0.389104,-0.58598,0.437859],[-2,0,1,1],[9E+09,9E+09
,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 9 THEN
            MoveL
[[-67.37,-1183.26,140.6],[0.559916,0.389097,-0.585984,0.437859],[-2,0,1,1],[9E+09,9E+0
9,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 10 THEN
            MoveL
[[-77.56,-1183.27,140.6],[0.559923,0.389094,-0.585981,0.437857],[-2,0,1,1],[9E+09,9E+0
9,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 11 THEN
            MoveL
[[-88.65,-1183.27,140.6],[0.559917,0.389103,-0.585978,0.43786],[-2,0,1,1],[9E+09,9E+09
,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 12 THEN
            MoveL
[[-98.67,-1183.26,140.6],[0.559919,0.389099,-0.585983,0.437854],[-2,0,1,1],[9E+09,9E+0
9,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 13 THEN
```

```
            MoveL
[[-114.10,-1183.26,140.6],[0.559921,0.389099,-0.585979,0.437857],[-2,0,1,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 14 THEN
            MoveL
[[-125.81,-1183.27,140.6],[0.559917,0.389097,-0.585981,0.437861],[-2,0,1,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 15 THEN
            MoveL
[[-135.83,-1183.27,140.6],[0.559921,0.389104,-0.585969,0.437867],[-2,0,1,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 16 THEN
            MoveL
[[-147.13,-1183.27,140.6],[0.559918,0.389099,-0.585973,0.437869],[-2,0,1,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 17 THEN
            MoveL
[[-158.12,-1183.26,140.6],[0.559918,0.389099,-0.58598,0.437859],[-2,0,1,1],[9E+09,9E+0
9,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 18 THEN
            MoveL
[[-169.81,-1183.26,140.6],[0.559918,0.389103,-0.585978,0.437859],[-2,0,1,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 19 THEN
            MoveL
[[-179.66,-1183.27,140.6],[0.559919,0.389102,-0.585973,0.437866],[-2,0,1,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 20 THEN
            MoveL
[[-194.37,-1183.26,140.6],[0.559918,0.389095,-0.585984,0.437857],[-2,0,1,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 21 THEN
            MoveL
[[-206.24,-1183.27,140.6],[0.559918,0.389101,-0.585977,0.437863],[-2,0,1,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 22 THEN
            MoveL
[[-217.46,-1183.27,140.6],[0.559919,0.389103,-0.585972,0.437866],[-2,0,1,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 23 THEN
```

```
            MoveL
[[-229.47,-1183.26,140.6],[0.55992,0.389097,-0.58598,0.437858],[-2,0,1,1],[9E+09,9E+09
,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 24 THEN
            MoveL
[[-240.77,-1183.27,140.6],[0.559922,0.389096,-0.585979,0.437859],[-2,0,2,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 25 THEN
            MoveL
[[-254.82,-1183.27,140.6],[0.55992,0.389094,-0.58598,0.437862],[-2,0,2,1],[9E+09,9E+09
,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 26 THEN
            MoveL
[[-265.42,-1183.27,140.6],[0.559921,0.389096,-0.585981,0.437857],[-2,0,2,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 27 THEN
            MoveL
[[-275.80,-1183.27,140.6],[0.559918,0.389101,-0.585978,0.437861],[-2,0,2,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 28 THEN
            MoveL
[[-287.09,-1183.27,140.6],[0.55992,0.3891,-0.585975,0.437864],[-2,0,2,1],[9E+09,9E+09,
9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 29 THEN
            MoveL
[[-298.55,-1183.26,140.6],[0.559921,0.389094,-0.585983,0.437856],[-2,0,2,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 30 THEN
            MoveL
[[-309.35,-1183.26,140.6],[0.559923,0.389097,-0.58598,0.437856],[-2,0,2,1],[9E+09,9E+0
9,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 31 THEN
            MoveL
[[-321.31,-1183.27,140.6],[0.559919,0.389096,-0.585979,0.437863],[-2,0,2,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 32 THEN
            MoveL
[[-334.77,-1183.26,140.6],[0.559922,0.389094,-0.585981,0.437858],[-2,0,2,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 33 THEN
```

```
            MoveL
[[-345.67,-1183.26,140.6],[0.55992,0.389101,-0.585978,0.437858],[-2,0,2,1],[9E+09,9E+0
9,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 34 THEN
            MoveL
[[-358.48,-1183.26,140.6],[0.55992,0.389096,-0.58598,0.43786],[-2,0,2,1],[9E+09,9E+09,
9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 35 THEN
            MoveL
[[-369.05,-1183.26,140.6],[0.559923,0.389091,-0.585981,0.43786],[-2,0,2,1],[9E+09,9E+0
9,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 36 THEN
            MoveL
[[-379.23,-1183.26,140.6],[0.559922,0.38909,-0.585983,0.437858],[-2,0,2,1],[9E+09,9E+0
9,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 37 THEN
            MoveL
[[-394.65,-1183.26,140.6],[0.55992,0.389099,-0.585978,0.43786],[-2,-1,2,1],[9E+09,9E+0
9,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 38 THEN
            MoveL
[[-404.42,-1183.27,140.6],[0.559923,0.389099,-0.585976,0.437859],[-2,-1,2,1],[9E+09,9E
+09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 39 THEN
            MoveL
[[-417.26,-1183.27,140.6],[0.559921,0.389097,-0.585975,0.437864],[-2,-1,2,1],[9E+09,9E
+09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 40 THEN
            MoveL
[[-426.28,-1183.26,140.6],[0.559923,0.389094,-0.585981,0.437857],[-2,-1,2,1],[9E+09,9E
+09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 41 THEN
            MoveL
[[-438.71,-1183.27,140.6],[0.559921,0.389092,-0.585981,0.43786],[-2,-1,2,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 42 THEN
            MoveL
[[-449.26,-1183.26,140.6],[0.559918,0.3891,-0.585978,0.437861],[-2,-1,2,1],[9E+09,9E+0
9,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 43 THEN
```

```
            MoveL
[[-460.44,-1183.26,140.6],[0.55992,0.389103,-0.585976,0.437859],[-2,-1,2,1],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;

            ELSEIF keyNum = 44 THEN
            MoveL
[[-478.25,-1183.26,140.6],[0.559921,0.389098,-0.585978,0.437861],[-2,-1,2,1],[9E+09,9E
+09,9E+09,9E+09,9E+09,9E+09]], v1000, z1, tool0;
            ENDIF

            ! Wait a bit
            WaitTime(0.5 - timecount * 0.1);
            Push;
      ENDIF

      ENDPROC

      PROC Path_20()
      MoveL Target_10,v1000,fine,tool0\WObj:=wobj0;
      MoveL Target_20,v1000,fine,tool0\WObj:=wobj0;
      ENDPROC

      PROC Retract()
      SetDO D652_10_DO1, 1;
      WaitTime 0.2;
      ENDPROC

      PROC Push()
      WaitTime 0.2;
      SetDO D652_10_DO1, 0;
      WaitTime 0.2;
      ENDPROC

      TRAP incCount
      timecount := timecount+1;
      ENDTRAP

ENDMODULE
```