



CARIBBEAN MARITIME UNIVERSITY

Kingston, Jamaica

PAPER COVERING AUTOMATING AN INDUSTRIAL PROCESS USING PYTHON

Assignment submitted in partial fulfilment of the requirements for the degree Bachelors of Engineering in Mechatronics

To

Mr. E. Bailey

By

Adrian Cunningham (20190600)

Kemoni Jones (20190731)

Leonard Smith (20190134)

Rashid Gordon (20190602)

Date: May 2021

## Table of Contents

Abstract .....	3
Introduction.....	3
Related Studies.....	4
Proposed Solution .....	6
Algorithm .....	6
Psuedocode .....	7
Results and Analysis .....	9
Flowchart.....	9
Program Code.....	10
Brief Rundown of some of the functions used. ....	11
Screenshots and Videos of Results.....	12
Conclusion and Future Scope .....	16
References .....	17

## **Abstract**

This paper details how our group went about using the python programming language to automate a industrial process, the simulation environment that was used was the RoboDK simulation environment. In this paper there will also be information about the considerations that needed to be taken to ensure that the python program would work well enough with the Robotics simulation environment, in this document you will be able to find pictures and videos results of the python program running and the robot arm carrying out the specified actions.

## **Introduction**

In this project we will be making use of our programming knowledge in python in order to program a robot arm in a simulation environment that will be used in this project to demonstrate a simple welding operation using python. In this operations a welding torch the ‘Abicor Binzel’ weld torch will be attached to the ‘ABB IRB 1200-5/0.9’ robot arm. Python programming will be used to weld the area around a target point in a specific shape to simulate welding a shape like hexagonal or circular shaped piece of metal onto another metal, and as such this program could be re-purposed using this same robot arm or one with a longer reach in combination with another tool such as a a engineering scribe to mark out shapes to help engineers to make more precise cuts or a laser cutting tool to laser cut simple shapes out of a material.

The rest of the document will contain information on how we went about accomplishing the welding operation using python programming.

## Related Studies

The goal of this project was to use the python programming language to automate some kind of industrial process, and so our group chose to automate a welding operation using a simulation environment that allows for programming robot arms using the python programming language.

In searching for simulation environments that allowed for programming using python we came across many other options out there such as Webots, Klampt, Gears and some others, but we eventually settled on using the RoboDK simulation environment that was seen being used in a video titled “Offline Programming With Python – RoboDK” (RoboDK, 2015). The RoboDK simulation environment was used because of its userfriendliness and the many tutorial videos that have been uploaded by the company behind the RoboDK software which would greatly aid us in not only learning how to properly use different features of the software but it would also help us if we happen to run into any errors or bugs.

For this project in order to accomplish our goal of automating an industrial process using the RoboDK simulation environment, there were several things we had to take in to consideration and keep in our minds, these include:

- **Degree of Freedom(DOF):** The degrees of freedom of the robot arm had to be at least six(6) in order to accurately mimic a similar operation being completed in real life as a lot of the robot arms that are used in the industry usually have at least 5 or 6 degrees of freedom to complete their jobs.
- **Size of Welding Shape:** The size of the weld shape will be controlled by the python program and the size set in this program has to be set so that the calculated positions will

not fall outside of the reach of the robot arm to avoid encountering an error and having nothing happen.

- **Distance between targets:** For this project since we're trying to accomplish a simple welding operation we're only using two(2) targets namely the 'Home' target position and the 'Target' position which is at the center of where we want to weld around. In our case since we only have to worry about 2 targets we have to manually move the robot to set these two points so that the points are valid points that the robot can actually reach when the program is ran, because if the distance between both targets is set to be too large then the python program will "throw up" an error because where the program is telling the robot to go is out of range.

Some of the code written and used in this project comes from the youtube video from RoboDK and from some of the examples that are available on the RoboDK PythonAPI documentation page.

The next section in this project report will contain the program code, algorithm, psuedocode, flowchart and pictures and videos of the robot simulation running in the RoboDK simulation environment.

## Proposed Solution

Using the RoboDK simulation environment we were able to automate a simple welding operation using the python programming language.

How we went about doing this is by setting up two(2) targets in the environment manually by moving the tool(the Abicor Binzel welding torch) attached to the robot arm to two different points, the first point was the labelled the 'Home' position and the second target was labelled 'Target' and this position is where the welding operation is done around. After the two points were setup all that was left to be done was to write the python program to make the robot arm carry out the weld shape around the 'Target' position.

This section will contain the Algorithm and the Psuedocode.

## Algorithm

Variable initializations

Get the name of the robot object and information on it from the RoboDK simulation environment

We then need to get information on where the two targets are in the simulation environment and store that information

Next we need to get the relative position of the 'Target' position in the RoboDK simulation environment(in RoboDK this is given in the form of a 4x4 matrix to represent the pose in the 3D space)

We tell the robot to move to the 'Home' target position and then to the 'Target' position

Next we will make the robot tool follow the path around the 'Target' position using some calculations to make this possible

Finally we will move the robot arm back to the center of the shape that was mapped out and then finally to the 'Home' position

## **Pseudocode**

```
# Declaring and initializing variables
```

```
from robolink import *
```

```
from robodk import *
```

```
RDK = Robolink()
```

```
robot = RDK.Item('ABB IRB 1200-5/0.9')
```

```
home = RDK.Item('Home')
```

```
target = RDK.Item('Target 1')
```

```
# Getting the Pose of the target position
```

```
poseref = target.Pose()
```

```
# Moving the robot
```

```
robot.Move(home)
```

```
robot.Move(target)
```

```
vertices = mbox("Enter the number of vertices")
```

```
# Calculation to calculate different angles and distance to be moved around the
```

```
# 'Target' position
```

```
for i in range(vertices+1):
```

```
    ang = i*2*pi/vertices
```

```
    posei = poseref*rotz(ang)*transl(300,0,0)*rotz(-ang)
```

```
    robot.Move(posei)
```

```
# Moving the robot back to the 'Target' position and then finally to the
```

```
# 'Home' position.
```

```
robot.Move(target)
```

```
robot.Move(home)
```



## Results and Analysis

### Flowchart

The flowchart was created from the psuedocode, so it will look different from the finished program code.

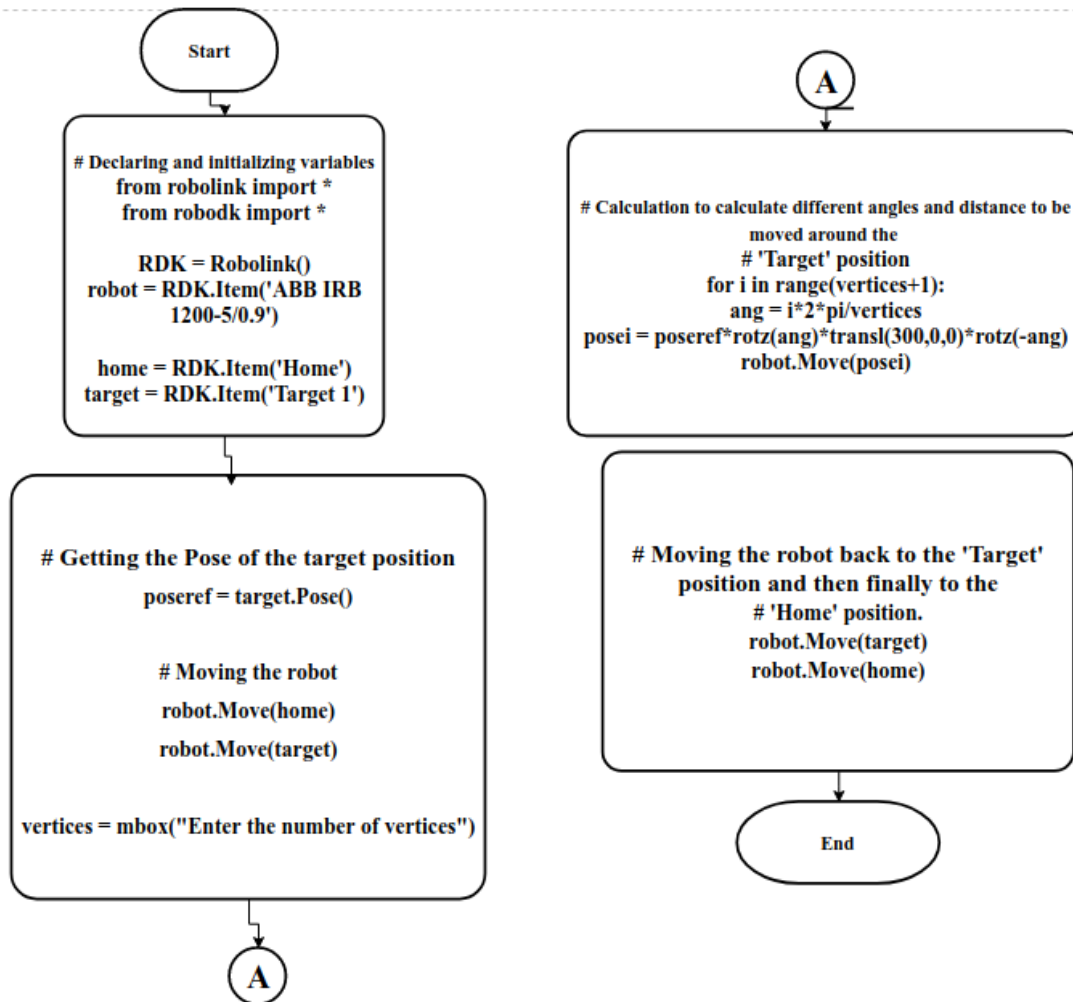
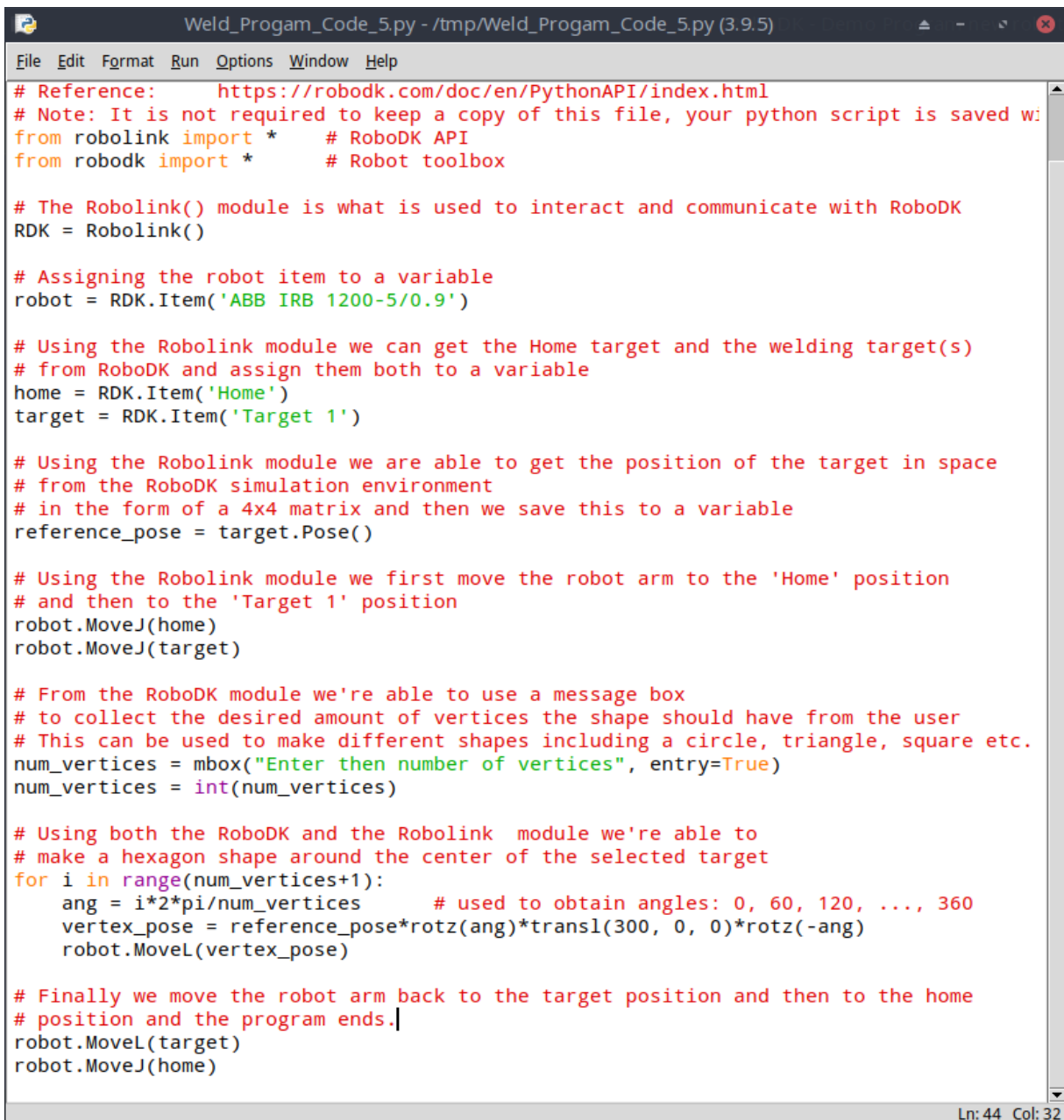


Figure 1: Diagram showing the flowchart that was created from the psuedocode..

## Program Code

The final program code that was used in the RoboDK simulation environment was written using the psuedocode above with a few minor changes for clarity and adhering to the syntax of the python programming language.



```
Weld_Program_Code_5.py - /tmp/Weld_Program_Code_5.py (3.9.5)
File Edit Format Run Options Window Help
# Reference: https://robodk.com/doc/en/PythonAPI/index.html
# Note: It is not required to keep a copy of this file, your python script is saved w
from robolink import * # RoboDK API
from robodk import * # Robot toolbox

# The Robolink() module is what is used to interact and communicate with RoboDK
RDK = Robolink()

# Assigning the robot item to a variable
robot = RDK.Item('ABB IRB 1200-5/0.9')

# Using the Robolink module we can get the Home target and the welding target(s)
# from RoboDK and assign them both to a variable
home = RDK.Item('Home')
target = RDK.Item('Target 1')

# Using the Robolink module we are able to get the position of the target in space
# from the RoboDK simulation environment
# in the form of a 4x4 matrix and then we save this to a variable
reference_pose = target.Pose()

# Using the Robolink module we first move the robot arm to the 'Home' position
# and then to the 'Target 1' position
robot.MoveJ(home)
robot.MoveJ(target)

# From the RoboDK module we're able to use a message box
# to collect the desired amount of vertices the shape should have from the user
# This can be used to make different shapes including a circle, triangle, square etc.
num_vertices = mbox("Enter then number of vertices", entry=True)
num_vertices = int(num_vertices)

# Using both the RoboDK and the Robolink module we're able to
# make a hexagon shape around the center of the selected target
for i in range(num_vertices+1):
    ang = i*2*pi/num_vertices # used to obtain angles: 0, 60, 120, ..., 360
    vertex_pose = reference_pose*rotz(ang)*transl(300, 0, 0)*rotz(-ang)
    robot.MoveL(vertex_pose)

# Finally we move the robot arm back to the target position and then to the home
# position and the program ends.
robot.MoveL(target)
robot.MoveJ(home)
```

Ln: 44 Col: 32

Figure 2: Showing the entire program code that was written for the robot arm.

### **Brief Rundown of some of the functions used.**

MoveJ – This function moves a robot to a specific target using the “Move Joint” mode.

It requires one parameter :

target → This is the target to move to, it can be the robot joints, the pose in the form of a 4x4 matrix or a target (item pointer) (RoboDK, n.d)

MoveL – This function moves a robot to a specific target using the “Move Linear” mode. This function only supports target items and not poses unlike MoveJ function.

It requires one parameter like MoveJ:

target → This is the target to move to, in this case it can be robot joints or a target (item pointer) (RoboDK, n.d)

transl – This is a part of the RoboDK simulation environment and through using the Robolink module a translation matrix with units in mm is returned, and the information that it contains tells the program the location of the ‘Target’ position in the 3D space. (RoboDK, n.d)

It takes in three parameters:

- tx (float) – translation along the X axis
- ty (float) – translation along the Y axis
- tz (float) – translation along the Z axis

Pose – This returns a relative position of an object, targets or reference frames, this information is in the form of a matrix object that represents the pose of the object or target in the 3D space

(its position and orientation). Poses are commonly used in robotics to place objects, targets or reference frames with respect to each other's position.

rotz – This returns a rotation matrix around the Z axis and is measured in radians.

It takes a single parameter ry – rotation around the Y axis.

## Screenshots and Videos of Results

The YouTube playlist link below will carry you to a YouTube playlist that shows all these results from the pictures below being carried out by the robot arm in the simulation environment.

The playlist of program results are available on YouTube at the following link:

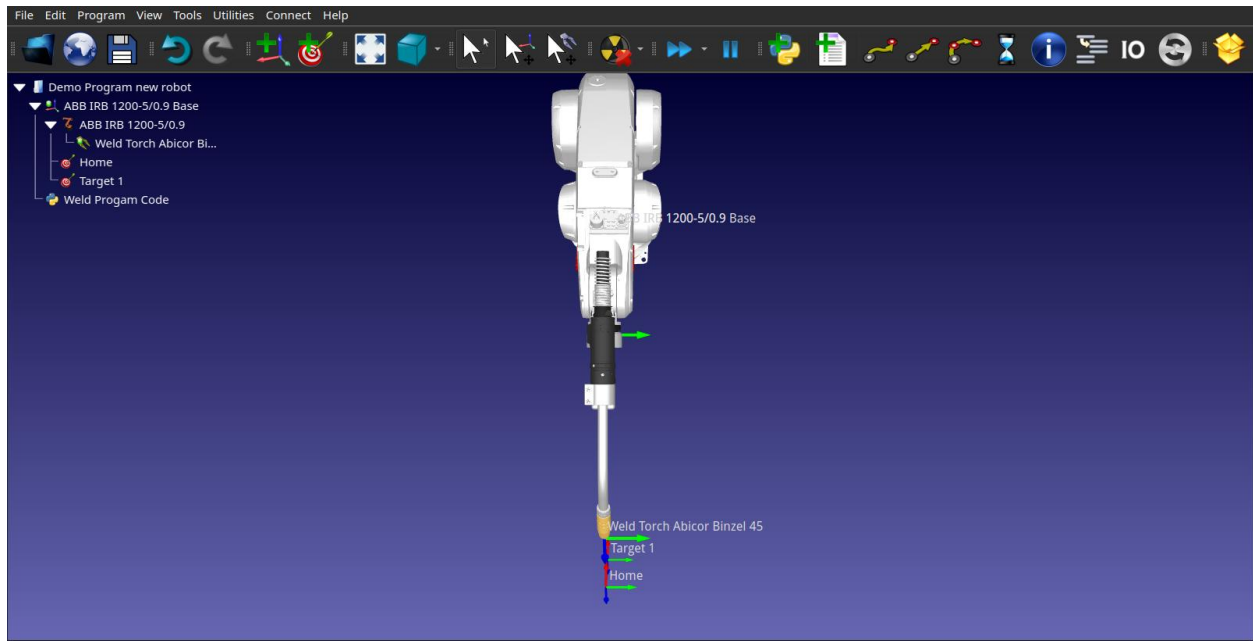
[https://www.youtube.com/playlist?list=PL7smhVSLZEp\\_GKh1Pa5sNTp2cRLHTOZMK](https://www.youtube.com/playlist?list=PL7smhVSLZEp_GKh1Pa5sNTp2cRLHTOZMK)

The program code and other documents can also be obtained from our Github repository at the following link:

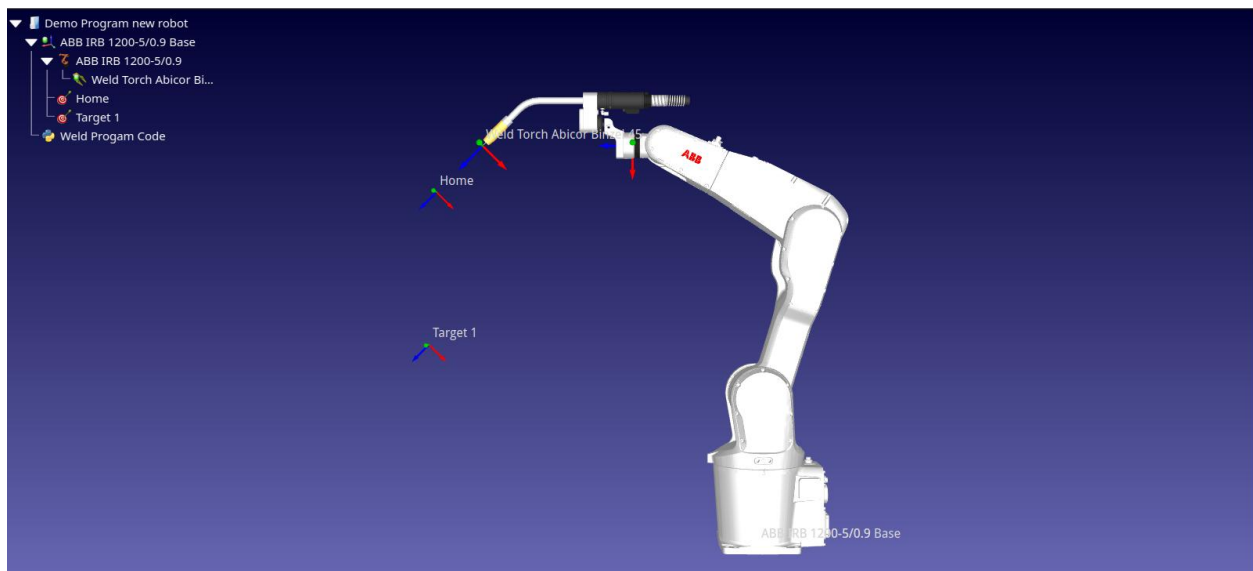
<https://github.com/RealYaddie/robot-project>

The pictures below were taken from within the RoboDK simulation environment, they're the result obtained when the numbers of vertices were set to different numbers to obtain different shapes and the pictures were taken from two angles for each test case.

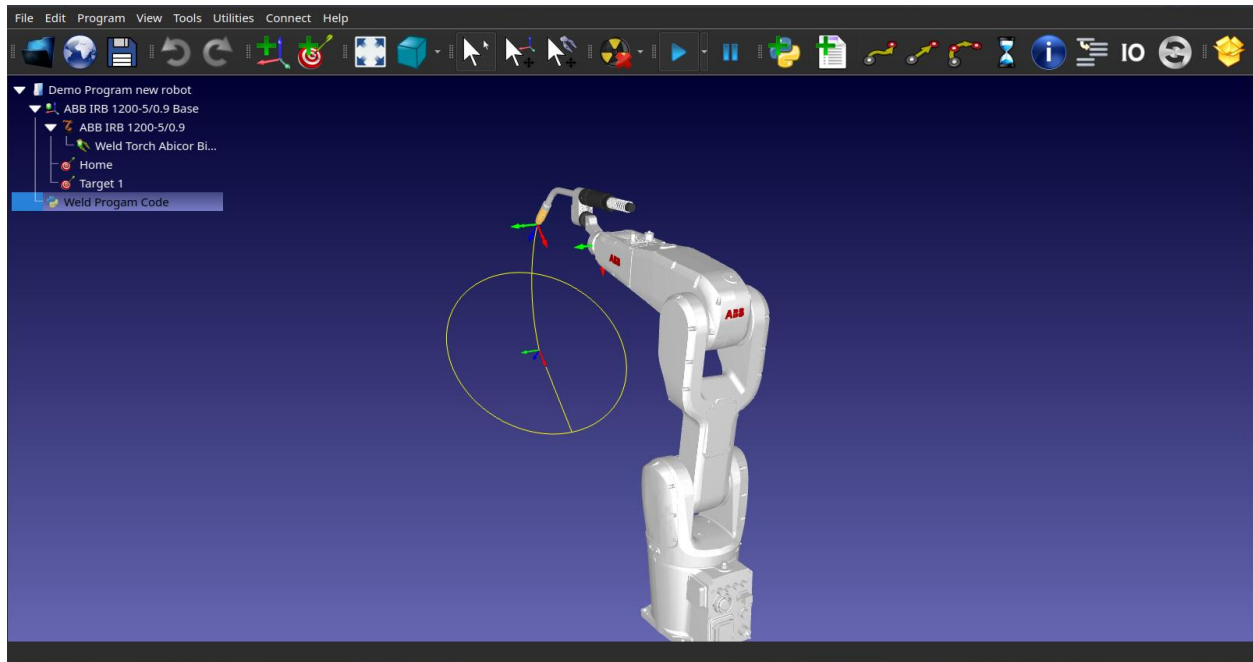
**Note:** Although these were the only pictures taken this is not all that the program is limited to other shapes can be obtained depending on the number entered for the number of vertices.



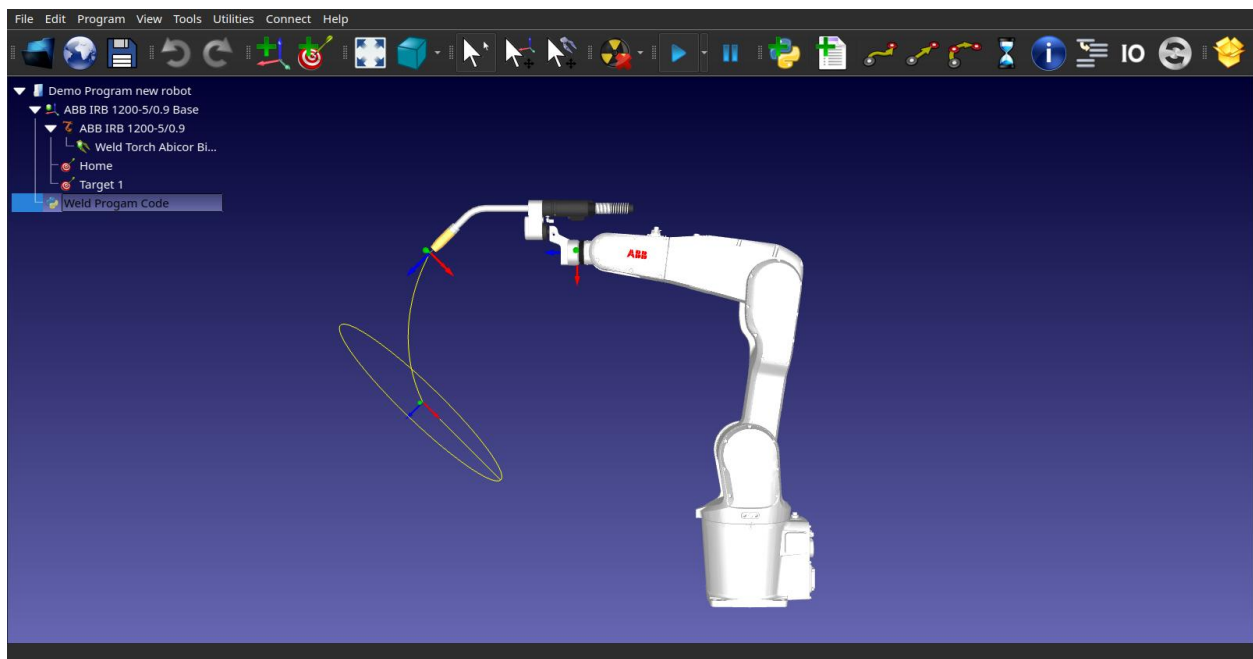
*Figure 3:* Picture showing a plan view of the robot arm in the RoboDK environment with the different targets and reference frames shown.



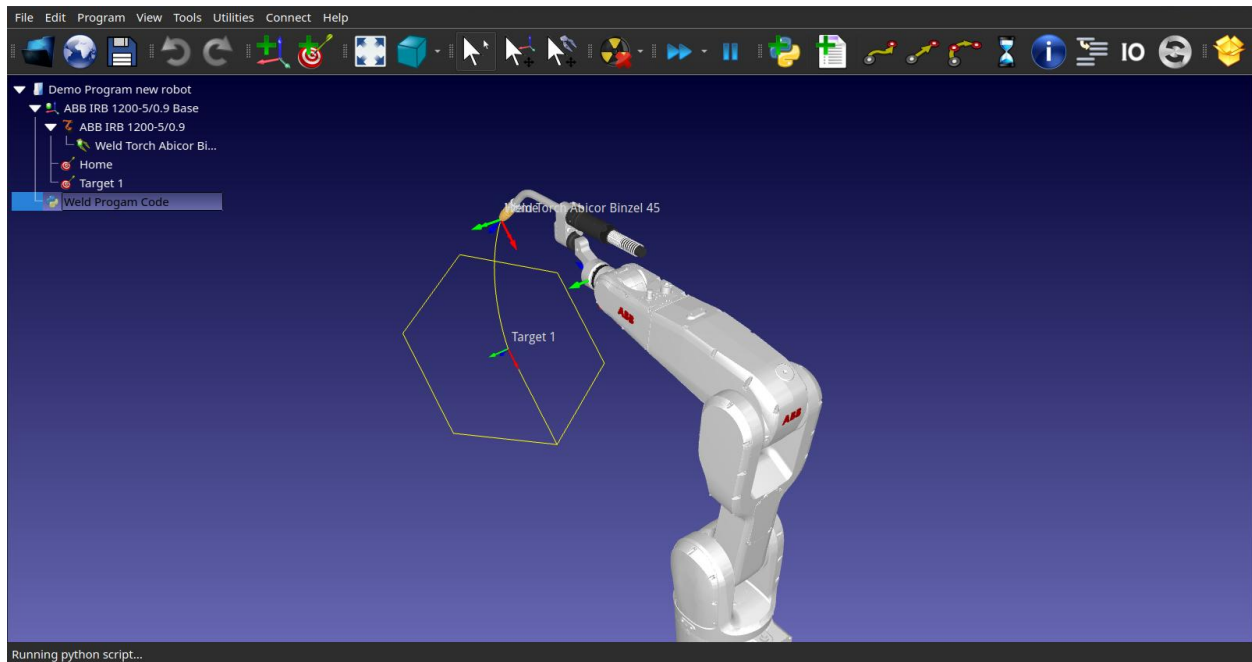
*Figure 4:* Picture showing a side view of the robot arm in the RoboDK environment with the different targets and reference frames shown.



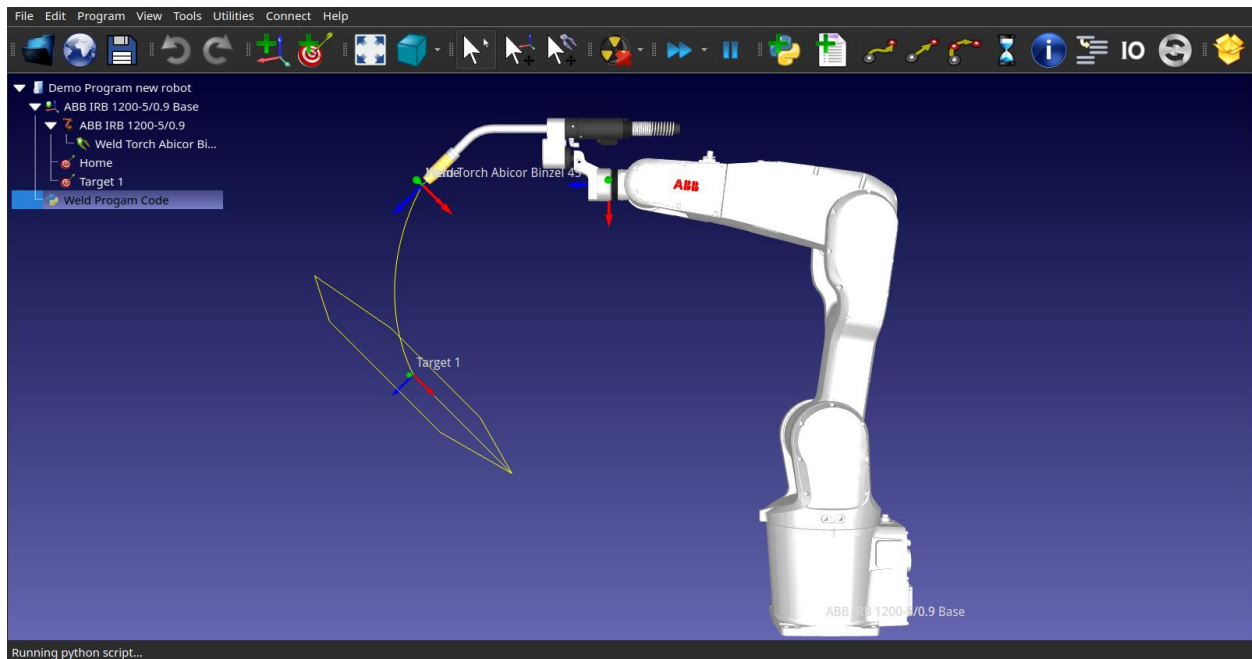
*Figure 5:* Picture showing an isometric view of the path followed by the robot tool after the python program was run and the number of vertices was set to 100.



*Figure 6:* Picture showing a side view of the same path followed above in Figure 5 with vertices set to 100.



*Figure 7:* Picture showing an isometric view of the path followed by the robot tool after the python program was run and the number of vertices was set to 6.



*Figure 8:* Picture showing a side view of the same path followed by the robot tool in Figure 7 with vertices set to 6.

## Conclusion and Future Scope

In conclusion the goal of this project was to use the python programming language to automate some kind of industrial process, our group decided to use the RoboDK simulation environment to perform a kind of welding operation using python. A proposed solution was put forward in the form of the algorithm and psuedocode that was written from it and that was used to achieve our goal in the RoboDK simulation environment with the use of python.

The solution that the group went with allowed for the code to be relatively easy to understand and manipulated as our project did not make use of any functions that were too complicated or too heavily based on any of the harder and more complex geometry mathematics.

For the future of this program, it was written in such a way that the program code can be easily and quickly re-purposed for other tasks such as laser cutting or a marker can be attached to the robot arm to draw out planning lines on materials to act as a guide and allow for precise cuts to be made by either another robot or a human. This program code can also be updated and worked on so as to allow for more complex welding operations that you would more likely need to be done in real life.



## References

RoboDK. (n.d.). RoboDK help redirection. Retrieved May 19, 2021, from robodk.com

website: <https://robodk.com/doc/en/PythonAPI/>

RoboDK. (n.d.-a). Basic Guide - RoboDK Documentation. Retrieved May 19, 2021,

from robodk.com website: <https://robodk.com/doc/en/Basic-Guide.html>

RoboDK. (2015, April 20). Offline Programming With Python - RoboDK. Retrieved

May 19, 2021, from [www.youtube.com](https://www.youtube.com) website:

<https://www.youtube.com/watch?v=Ic-iKGSc7dk&t=1s>