**Abstract 10%**

**https://writing.wisc.edu/handbook/assignments/writing-an-abstract-for-your-research-paper/**

An abstract is a short summary of your (published or unpublished) research paper, usually about a paragraph (c. 6-7 sentences, 150-250 words) long. A well-written abstract serves multiple purposes:

* an abstract lets readers get the gist or essence of your paper or article quickly, in order to decide whether to read the full paper;
* an abstract prepares readers to follow the detailed information, analyses, and arguments in your full paper;
* and, later, an abstract helps readers remember key points from your paper.

**Introduction 10%**

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 5% - explain your research and your sources that you used, research analysis… where it can be applied/used and what sources you used to come to ur proposed solution, all the code you got and where you got it from**

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 compmay 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 mimc 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 and 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 25% - This is the approach you took to solving the problem/proposal, your code in here.**

**Maybe put Algorithm in here**

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

**Psuedocode**

# 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 30% - The program, flowcharts, screenshots etc**

**Flowchart**

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

**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.

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 specfic target using the “Move Linear” mode. This function only supports target items and not poses unlike MoveJ function.

It requries one paramter 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 apart 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 paramters:

* 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**

**Conclusion and future scope/work 10%**

dasd

**Bibliography/References 10%**

**https://robodk.com/doc/en/PythonAPI/examples.html**

**https://robodk.com/doc/en/Basic-Guide.html**

**https://robodk.com/doc/en/PythonAPI/intro.html**

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