# Description of Theory:

For this assignment, the two portion of the code is merged into two functions for easy access in the latter part of the code. moveRobot() function and updatePosition() function. With those two functions created, I can now call robot movement or request sensor position update at any time I want.

The initialization stage includes setting the tool flange point downwards and move towards the above of the gear\_part\_ariac. Using initialization, I am able to also tell if the magic\_object\_finder is functioning correctly or not.

Our path planning code start with a comparison of the target and current position. In order to avoid confusion, I start by first questioning x position, then y position. The whole operation is under a loop until both x and y target has been achieved. In order to avoid useless calculation and step, before the loop starts, I first check if our object is already aligned or not, so that I can avoid step duplication.

During the comparison of sensed value of the gear\_part\_ariac and the target value, I first decide how I want to offset our tool flange. Do I want to offset on the positive direction or do I want to offset it towards negative direction? Once those calculation are completed, I send the command to move robot to ask the tool flange goes to the start point of the action.

I then calculate where do I want the tool flange to end. I also consider the fact we need to offset our tool flange to make the gear\_part\_ariac arrive at the target.

Once the gear part is pushed, I stop and back the tool flange a bit. This is because I noticed if I lift the tool flange right away, the gear part sometimes got brought up the ground. We don’t want this to happen therefore I ask the tool flange to back away and then lift.

Once the gear part is backed away, we lift the tool flange to above the gear part. This serves as two purpose. It is the end mark of an action, and we consider this as a homing position during the run, it also serves as a local checksum for us to see if the object finder is functioning or not.

We then proceed to Y axis operation, similar to X axis operation logic.

Once Y axis operation is completed, we examine where the gear part is, and compare it with the target. We did give the comparison some slack since if we don’t do that, the simulation will freak out due to some 0.00001 difference. Setting a reasonable range is necessary. Once the comparison is completed, it flips the “switch” for controlling if we have reached the target. If not, the loop will force the program to run through movement again.

# Observations:

One thing I noticed is that if you lift the tool flange with contact to gear part, it will lift and drop the gear part as well, causing imprecision. I solved the issue by backing the tool flange out so that it is not contacting the gear part.

The other thing I notice is that sometimes your gear might be already in line with the target, and you might only need to do one axis movement, therefore I added comparison before the loop.

Finally, the accuracy is also affected greatly by the speed of the tool flange. I found my current setting very ideal. I gave the robot enough time to travel with the gear part, and also reduce the time when the robot is traveling alone.

# Precision:

Personally I think this robot has achieved relatively high precision, with ±0.001 accuracy, though I set my accuracy range to be 0.01 just to avoid meaningless robot motion and reduce the time for demos.

# What’s next?

Due to the time limitation, I planned on making a node that constantly checks the gear part position and pre-warn the robot if the gear part gets loose. However, since this is a pretty simple motion, I don’t find it as effective as just move and stop and check. For future more complicated motions, this might be necessary.