Max Bauer, Matthew Chang, Samuel Young
11/18/2021
CSDS 373
Professor Lee

Theory of Operation

Lab 6

Building off of Lab 5, the objective of the lab 6 code is to manipulate the arm in ARIAC by using waypoints and trajectory matrices. In addition, the ur_kinematics package is also crucial to manipulating the 6 joints. In addition, when we use the transformation matrix, we know that there is a matrix that causes the vacuum gripper to point sideways, and another matrix that causes the vacuum gripper to point downwards. Let us also assume that the code will tell the arm and by extension ROS to wait a moment before performing certain tasks, and that we can omit these in the description since they are not as conceptually important.

First we start off by defining our package dependencies. Next, we move on to defining our logical cameras.

In our main method, we take an int and a char, and we initiate ROS "trigger node". Next, as we adjust the logical cameras and their vectors. We then use the Tf2 to initialize the transformation matrix.

After initializing the subscriber node, we then set up a while loop for if ROS = OK. We have the subscriber note listen for a successful start. IF it won't start, we stop ROS, ELSE we state that the competition has started successfully OR it can't start.

Next, we initialize a spin up. We do this because we need to call the callbacks and handle messages. In this case, we are handling arriving messages. Specifically, we are receiving 8 solutions from Inverse Kinematics that will be used to set waypoints for the joints of the arm.

Then, we set up another while loop that runs while ROS = OK. While ROS is still running, we use a single if statement to check whether or not we have a service order. If there IS a service order or more, then we use several if else statements to use the logic cameras to check if the item we are looking for is in any of the bins, the AGV trays, and checking if our specific part is there. We then use a for loop to determine the part's pose for future reference. The arm is now ready to be moved.

To move the arm, we need to try a few different poses. While first_model is greater than zero, we try different transformations using lookup_transformation for both the arm base and linear

actuator. We catch any exceptions that might occur. The Linear Actuator is not a part of what we need to use for Qpose, so joint_state_position[1] is not included. Furthermore, we also attempt to set waypoints for the joints.

Part of finding waypoints is to find the best solution for q_way_indx and q_sols_indx, but this feature might not factor into the final project as much.

Finally, the last section of our code deals with being able to find our way back and reverse our waypoints. Lastly, after completing the task(s), ROS waits for shutdown, and returns 0.

Phase 1

The task is to lift every object up with the vacuum gripper and then put it back down in the bin.

For the first phase of the project, we take our existing code from Lab 6 and ensure that not only can we move the arm into the right position using the code from lab 6, but that we can also position our vacuum gripper properly.

After positioning the arm, the vacuum gripper must come close enough to the object to pick it up, and the arm must invert the waypoints it used to move from its original position to get the vacuum gripper over the part to return to its original position. The arm will then move the part back to the bin by positioning the vacuum gripper over the bin, depositing the item, and then once again inverting the waypoints to return to its original position.

Phase 2

Now, the task is to move the base, pick up a part, put it on an AVG tray and then submit it.

Taking the Lab 6 code, the first objective should be to find where the part is using the logic cameras. If they cannot find the part at all, then we just move on. Assuming the part is in a bin, the code will then use a transformation matrix to determine where the part is with respect to the arm. The arm must then first be moved via Linear Actuator. Next, the arm must position and orient the vacuum gripper over the part, invert the waypoints to return to its original position, and move towards the AVG tray.

At this point, since the phase has a shipment of one part, the shipment should be completed. Since the shipment is completed, the code should check if the order is completed, and since the order is complete it should move onto the next phase.

Phase 3

Next, the task is further complicated with more logical cameras, an order of two shipments, and each arm's vacuum gripper must deliver said parts.

Similar to Phase 2, the arm needs to find the part (if it exists, which it should), move the linear actuator, move the arm and orient the vacuum gripper close enough above the part, invert the waypoints to return to its original position, move the linear actuator to the AGV, move the vacuum gripper and held part above the tray of the AGV, deposit the part, and return to original position. Then, the code should check if a shipment is completed.

If it is, then the arm is free to check if the order is complete or there are more shipments to be fulfilled. If the order is complete, the code should return 0; if the order is not complete then the code should move to the next shipment and repeat the above. If the shipment is incomplete, then the arm must repeat the above until the shipment is complete, and then check whether or not the order is complete and another shipment needs to be fulfilled.

Phase 4

Finally, the last task is to submit two orders with two shipments each and multiple parts using both AGV. This task is basically the same as Phase 3, just with more orders, shipments per order, and parts per shipment.

It should be noted that instead of executing in distinct phases, the code just executes everything at once for convenience. Specifically, phases 2, 3, and 4 are lumped together because it was easier to get them working together.

Block Diagram depicting Theory of Operation

Maxwell Bauer, Matthew Chang, Samuel Young