UNIVERSITY OF MARYLAND

ENPM809B: Final Project

Report By: Group 7
Team Members:

An Li	-	116156356
Andre Ferreira	-	116345828
Nakul Patel	-	116334877
Revati Naik	-	116723015
Sarvesh Thakur	-	116352062
Srujan Panuganti	-	116302319

1 Project Objective

The project revolves around the Agile Robotics for Industrial Automation Competition (ARIAC) which deals with building a tool kit based on an order (a set of shipments) using industrial tool parts available in the environment. These parts are either present in static bins or delivered by a conveyor belt. The parts need to be picked as the order requirement and have to be shipped using an Autonomous Ground Vehicle (AGV). Each order is considered to be fully complete only when every of its shipment has been prepared with the desired, non-faulty parts in the correct poses, and shipped by correct AGVs. In addition to fulfill orders, the competition has challenges including order updates, sensor blackout, unexpected part dropping, part flipping, and multiple shipments within an order. The topics that have been practiced in this project include but not limited to ROS/C++ programming, coordinate transformation with TF library, trajectory planning by MoveIt interface, and strategies development and implementation for an autonomous system.

2 Environment Set-Up

The ARIAC environment comes with a basic set-up which includes the conveyor belt, the bins, the two arms (UR-10), the AGV trays and the Quality Control cameras on the AGV trays
Location of sensors in the environment:

1. Logical Camera 1: Placed above the conveyor belt at a reasonable distance from the ARM1.
2. Logical Camera 2,3,4,5,6,7: Placed above the bins with parts at a reasonable distance from the ARM1 and ARM2
3. Break Beam Sensor 1: Placed on the conveyor belt at a reasonable distance from the logical camera 1 and is used to activate the logical camera to read the incoming parts.
4. Break Beam Sensor 2: Placed on the conveyor belt at reasonable distance from Arm 1 and is used to activate the ARM 1 to pick up the incoming part.
5. Break Beam Sensor 3: Placed on the conveyor belt at reasonable distance from ARM 2 and is used to activate the ARM 2 to pick up the incoming part.
6. Break Beam Sensor 4: Placed at the start of the conveyor belt to keep a check on the incoming products.
7. Logical Camera 8,9: Placed on the AGV trays.
8. Quality Control Sensor 1: This is located on the AGV1 to detect the faulty parts.
9. Quality Control Sensor 2: This is located on the AGV2 to detect the faulty parts.

The role of each of the above mentioned sensors is explained in detail. The Logical Camera placed on the conveyor belt is used to read the part type and the pose of the incoming products on the conveyor belt. This data is used by the ARM to pick up the required part as per the order from the belt. The Break Beam Sensor 1, 2 and 3 are used to complete the loop of picking up parts from the belt. These sensors give an output when the beam is cut by an object. This is helpful in detecting the presence of an object at a particular location on the belt. The Break Beam Sensor 4 which is placed at the starting of the belt is used to detect the incoming parts so that the AMR wich is picking up from the belt, can be used for other tasks when there is no incoming part on the

belt. This is explained in detail in the further sections. The Logical Cameras placed on the bins are used to read the details such as the part type and their respective pose of the parts placed in the bins. This data is useful in completing the order for picking up parts from the bins. The Logical Cameras on the AGVs are used to read the pose of the parts placed on the AGV while kit building process. The Quality Control Sensors available on the AGVs are used to check for faulty parts while building the kit.

3 Challenges

In ARIAC 2019, the following challenges will appear during kit building, so the system needs to be agile and generic to handle all of them. Any combination of the challenges can be present in .yaml file of the ROS package.

3.1 Dropping Parts Challenge

While building a kit, the gripper of a robot might malfunction in various instances, e.g., when a product is being picked from a storage bin or when a product is being placed into a kit tray. The task is to continue building the desired kit, which could include retrieving the dropped product or fetching a new product.

3.2 Update Order Challenge

The challenge is that while the robots are building a kit, another order might come up as order update through the order ROS topic. The task is to build/modify the kit based on the updated one. The update to a previously assigned order is defined by *"order_0_update_0"*.

3.3 Faulty Parts Challenge

In this challenge, the task is to avoid using faulty parts to build kits. The quality checking can be done by subscribing to the quality control sensors above each AGV: the topic will provide the pose of faulty parts in its scope. The team has to replace the faulty one with a new part. Details about our approach will be described in the Approach section.

3.4 Sensor Blackout Challenge

In this challenge, communication with most of the sensors will be lost temporarily, referred to as a "sensor blackout". At the start of the competition, the sensors will be publishing data normally. With a particular condition, most of the sensors will stop publishing for a fixed period of time. This applies to user-defined sensors and sensors that are present by default in the environment.

3.5 Flipped Parts Challenge

Of five types of parts in ARIAC kit building (disk, gear, piston-rod, pulley, and gasket), only pulley might be required to be flipped. For such part, the roll value will be described as 'pi' in the .yaml file of ROS package.

4 Approach

The pipeline has been built on few ground strategies mentioned as below:

1. If one arm is building the kit, the other arm is responsible to grab the parts from the belt.
2. We build one kit at a time. It's sequential, one kit after the other.
3. While one arm is building the kit, if it is unable to reach a part from the bins, the other arm will help in passing the part.
4. Parts coming from the belt are given higher priority while building the kit.
5. While updating an order, if a part needs to be removed, if that part is initially picked up from the belt, it will not be removed, instead it will be placed on the other tray where kit is not being built.

4.1 Kit Building

As the competition starts, the whole ARIAC environment is initialized with along with all the sensors, arms and parts as discussed in the previous sections. After all the sensors are up and running, we first build the frame names for all the parts that are in the bins and save them to a map data structure *product_frame_list_*. The frame names which are saved are obtained by accessing all the subscribers subscribing to the logical cameras that are located over the bins. This *product_frame_list_* serves as an ultimate list locating all the parts over the bins.

Our kit building pipeline subscribes to the */ariac/orders* topic to obtain the orders that need to be built. The order specifies,

1. Number of shipments to be built
2. AGV on which the kit needs to be built corresponding to each shipment
3. Products to be obtained for the shipment
4. Update to the order, if needed. An update to an order can come at any time while building the kit or after building the kit

We process the data structure obtained from the */ariac/orders* and build our custom data structures using the *product_frame_list_* to classify the products into three different groups as follows

1. parts to pickup from the bins that are reachable to the arm that is building the kit. Stored into *parts_from_bin_reachable* data structure.
2. parts to pickup from the bins that are unreachable to the arm that is building the kit. Stored into *parts_from_bin_unreachable* data structure.
3. parts to pickup from the conveyor belt by the arm that is not building the kit. Stored into *parts_from_belt* data structure.

Reachable bins are those bins which are accessible to the arm with confidence. Each arm can access parts only from 4 out of the 6 bins with confidence. The farther two bins are unreachable for the arm of interest. Once the three maps are obtained, we loop through these maps one after the another for building the kit. However, the *parts_from_belt* are looped in parallel with others, as it depends on when the parts on conveyor belt are available. How we pick up parts from the belt is explained in detail in 4.1.2 The process of picking parts from bins that are reachable and unreachable are explained in detail in 4.1.1

4.1.1 Picking up from the bins

For better illustration of this section, lets assume the Kit is to be built on AGV1. Let the arm which is closer to the AGV1 is Arm1 and the other arm is Arm2. Arm1 will be responsible for building the kit, Arm2 is responsible to help Arm1 while building the kit, if necessary. Arm2 is also responsible to pickup parts from the belt.

We loop through the *parts_from_bin_reachable* and *parts_from_bin_unreachable* to pickup parts from the bins. We always loop through the *parts_from_bin_reachable* first and then *parts_from_bin_unreachable*. We use the Arm1 when looping through the *parts_from_bin_reachable*. We use both the Arm1 and Arm2 while looping through the *parts_from_bin_unreachable*. We enforce additional conditions to be met before looping through the *parts_from_bin_unreachable*, as picking these parts needs the help of Arm2 which might be busy picking up parts from the belt. We use several flags to resolve conflicts between the way arms are used to collaborate for a task. Picking up parts form the bins is implemented as a method named PickAndPlace() that follows the following pipeline.

1. PickAndPlace() method is called each time for a product type in *parts_from_bin_reachable* map. The part type and the pose where the product is to be dropped are passed as arguments to the PickAndPlace().

2. In PickAndPlace() method, the frame name of the product is obtained by accessing the *product_frame_list_*.

3. The frame name obtained is used to get the pose of the part in one of the bins. This is assigned as the pick pose. This pick pose is transformed from the logical camera coordinated to world coordinates.

4. The frame name is used to identify if the part is from the reachable/ unreachable bins. If the Item is reachable, skip the steps 5,6,7.

5. If the part is from unreachable bins, if the part type is a pulley part, the Z-coordinate of the pose is incremented by 0.062 before picking the part.

6. Arm2 is used to pickup the part.

7. Arm2 places the part at the center of the rail. The rail center pose is assigned as the pick pose.

8. Arm1 will pickup the part using this pick pose.

9. The part drop pose is now transformed from Kit tray coordinated to world coordinates and checked for flipping condition, if the part needs to be flipped the 4.7 is performed

10. Now the Arm1 will place the part at the drop pose. If the part unexpectedly drops in an unexpected pose due to any external factor, 4.6 is performed.

11. After finishing the previous step, the process explained in 4.4 is performed.

The Above steps are repeated for every product type and pose while looping through the *parts_from_bin_reachable* and *parts_from_bin_unreachable* maps to build the kit. While this process is in progress, 4.1.2 executes in parallel.

4.1.2 Picking up from the belt

For every order, there will be a corresponding arm assignment: a robot arm (called Arm 1) will focus on picking static parts from bins inside Arm 1's reachable range, and the other arm (called Arm 2) will focus on picking belt parts and collaborating with Arm 1. Fig. (1) shows an example of an arm assignment.

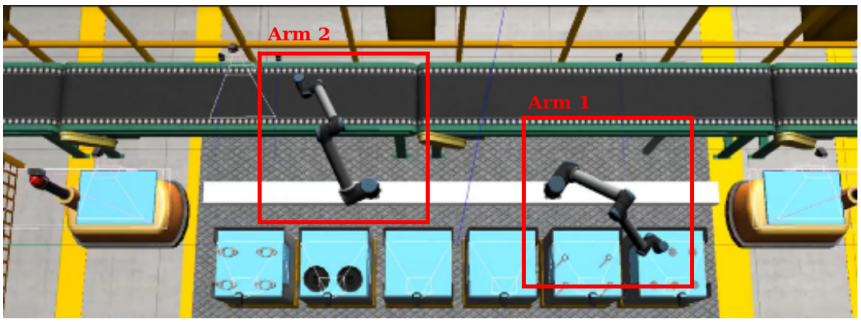

Figure 1: An example of an arm assignment for a specific order.

For Arm 2, it follows rules in bellow priority. We set flags for each scenario to make sure the rules are obeyed.

1. Arm 2 should always finish its current task first, which includes moving or picking and placing one part.

2. Unless the system is experiencing sensor blackout, Arm 2 has collected enough belt parts for all orders, or there is no storage space for belt parts, if there are parts coming from the belt, Arm 2 should go and try to pick desired parts from the belt.

3. Every time there is a belt part pass through a specific break beam sensor, Arm 2 will start counting time. If the counting time exceeds a set time threshold, Arm 2 will raise a *no_part_coming* flag in the system and go back to collaborate with Arm 1.

4. Unless conflicting to the previous rules, Arm 2 should collaborate with Arm 1.

To pick the desired product coming from the belt, we use the so-called "photographing" approach: as long as there is something triggering the break beam sensor 1, its callback function will ask the system to subscribe to the topic of a logical camera above the belt (called *lc_belt*). The callback function of *lc_belt* will then extract the product type of the incoming part from the image massage, and unsubscribe the *lc_belt* topic after this process. The reason for using this approach is to prevent getting undesired image information from *lc_belt*. The incoming product type is then passed into the corresponding break beam sensor callback function for Arm 2 as one of the factors of whether the part should be picked. Fig. (2) shows the configuration of sensors discussed above.

If Arm 2 picked the correct part successfully, it will place it on the closest kit tray (called the temporary tray) for temporary storage. Usually, we can pick a static part by using its last known position, which is registered by logical cameras into the *TF* listener as long as we know its correct frame name. For the belt parts, however, due to the sensor blackout, the counter of belt parts will be blocked and the later part frame names won't be correct. We thus develop an approach: let the system monitor four predefined poses on the temporary tray in its coordinate (which has been

initialized when the Arm 2 object been created) for storing parts picked from the belt. Therefore, we can use the system to find where the desired belt part locates on the tray, and also if there are still spaces to store more belt parts without explicitly knowing the correct product frame name. Fig. (3) demonstrates how the temporary tray and the corresponding predefined poses been used.

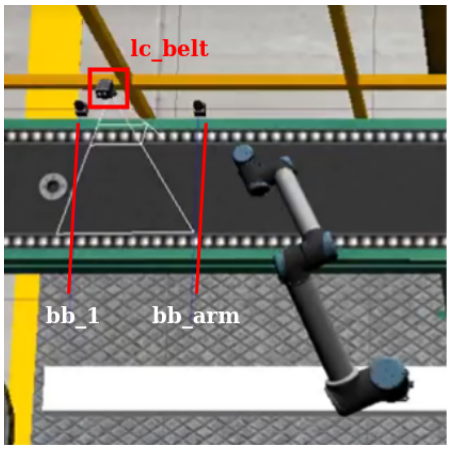

Figure 2: The configuration of sensors deployment on the conveyor belt.

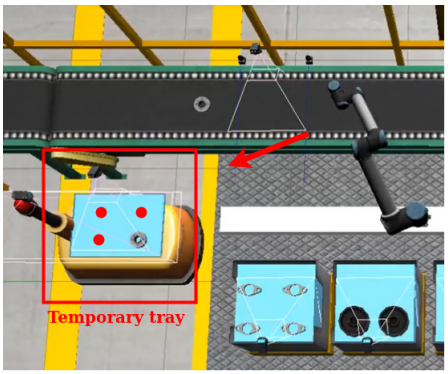

Figure 3: The temporary tray and the corresponding predefined poses.

4.2 Submitting the kit

After performing 4.1.1 and 4.1.2, the tray is checked for any faulty parts to be removed as explained in the 4.5 and necessary steps are taken to replace all the faulty with the good parts using PickAndThrow() and PickAndPlace() methods. Now the kit is ready to submit if there is no order update. If there is an order update the process explained in the 4.8 is performed. The Kit is then submitted

This process is repeated for all the shipments in that order. Once an order is successfully finished, the same process is followed to process the next order.

4.3 Sensor Blackout Challenge

During the sensor blackout, none of the logical cameras and quality control sensors give the feedback. This introduces many uncertainties, if the pipeline highly depends only on the sensors. To overcome this problem we came up with an approach which greatly reduces the dependence on the sensors without compromising accuracy and agility.

We employed a simple trick in detecting and keeping track of sensor blackout. We used a break_beam_coming callback function which subscribes to the break_beam_coming topic, which runs even during the sensor blackout. This way we kept a track of when the sensor blackout occurs and when it ends. Based on this track, we absorb different strategies when performing tasks like, quality check, pose correction and dropped part detection. we have a common approach dealing with the sensor blackout while addressing the challenges like pose correction, drop and detection. We implemented quite a different approach dealing with quality checking during sensor blackout. These approaches are discussed in more detail in subsequent sections.

4.4 Quality Checking

Verify the presence of a faulty on the tray its an important step to guarantee the correctness of the builded kit. This step is performed inside the function pickAndPlace which allow us to use the same part type and position to substitute the desired part using the recursive call. Every time a new part is added to the kit, the size of the message from the quality camera topic is checked and if its values is bigger them zero, it means that the last part added is faulty. After identifying the faulty part, the pickAndTrow is function called to remove the part from the tray and after this step ends the function pickAndPlace is called recursively to add the same part type again.

4.5 Quality Checking during Blackout

During the blackout is not possible to verify the quality of the parts. when the sensor returns to its normal operation the simple approach used to check for inappropriate tools inside pickAndPlace is compromised once it is not possible to guarantee that the faulty part is the last one. The strategy developed was monitor the occurrence of blackout and once detected the faulty check does not happens inside the pickAndPlace function anymore . The quality check after blackout is performed only before submitting the AGV. This step consists of comparing the positions of the faulty parts provided by the quality camera with the positions provided by the logical camera of the same AGV.Performing this step allow us to know which part types need to be replaced and its positions. After retrieving all the necessary information all the faulty parts are removed from the tray using the function pickAndThrow and after this step is over we place the new parts using the pickAndPlace Function.

4.6 Dropping Parts Challenge

Dropping part challenge is a potential challenge that can occur in real industrial kit building scenarios. We employed a reactive approach in identifying such drops of parts during Pick and Place operations while kit building. We use the gripper state of the arms to identify if a part is dropped even before it is placed into the kit.

If a part is dropped even before it is actually placed in the tray, We find out where the part is dropped using the logical cameras. As we also take a snapshot of the logical camera before placing a part and after placing a part, these snapshots help us to identify the most recently added part.

We check if that part is located where it is supposed to be. If it is not where it is supposed to be, we fix its pose.

4.6.1 Pose Correction During Sensor Blackout

A major challenge for pose correction for dropped parts is the sensor blackout. As discussed in previous sections, a comprehensive approach is devised. Whenever, we encounter a dropped part during a sensor blackout, we do not correct the pose in a reactive fashion, instead we record this abnormality and reconsider it when all the sensors become live.

4.7 Flipping Parts Challenge

When a part's desired pose has *pi* rotation in either roll or pitch direction, we have to flip that part. To approach this challenge, we have used both arms to flip the part. The arm that has the part in hand, comes in a predefined transfer position on the rail. The other arm, if it is not busy, comes to its transfer position on the rail. Then they approach each other until the pulley touches both arms. After that, the other arm grabs the part and the first arm release it. The other arm will then place the part on the rail for the first arm to pick it up again.

While approaching the Flip Part challenge, we had to think about four scenarios. Scenarios are as follows:

1. If the kit has to be build on AGV1 and the part is in reachable position of Arm1.
2. If the kit has to be build on AGV1 and the part is in unreachable position of Arm1.
3. If the kit has to be build on AGV2 and the part is in reachable position of Arm2.
4. If the kit has to be build on AGV2 and the part is in unreachable position of Arm2.

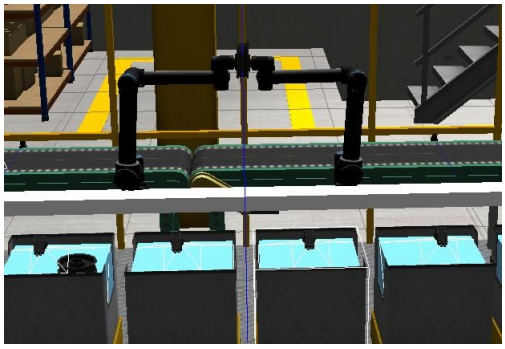

Figure 4: Flipping in progress

The code for *FlipPart()* function is modular in the sense that it can be called under these various configurations. The arm which has the part attached to it is called the owner and the other arm is called the helper. The owner transfers the part to the helper, which drops it on the rail for the owner to pick it up again. This way the part is flipped accordingly.

4.8 Order Update

Order Update approach is one of the most complex of this project and The group faced a lot of problems during its implementation. Once we finish building the original kit and update is detected we create two maps where we store the part type and poses of the original order and the update respectively . The map has the part type as key value and its correspondent is a vector with all the poses from that specific part type, which means that the size of the vector is the number of each part inside the order. The following image is a representation of the 2 maps. For understanding purpose the positions are only represented by a number instead of Cartesian coordinates.

Original

Part Type	Position	Position	Position
X	1	2	3
Y	4		
Z	5		

Updated

Part Type	Position	Position	Position
X	1	4	
Y	2		
Z	3		
W	6		

Figure 5: Maps Example

Once the two maps are fulfilled, its possible to compare them to identify what need to be updated in the order. This Tasks is separated in 3 steps which are described as following.

4.8.1 Identify and Remove Extra Parts

The first step of the update order is to identify how many parts need to be removed from the remaining Kit. For this process the size of the vector of each part type from original and updated orders are compared. If the size of the vector of updated order is bigger than the original it means we need to remove parts from the kit. We perform a simple subtraction calculation to know how many parts need to be removed from the original kit. In this step we also compare the poses from the original kit and update order to make sure we don't remove any part that are already in the desired position according to the update order. Using the maps from the example the size of the vector of the part type X are 3 in the Original and 2 on the update, which means that one part x need to be removed from the tray. There are 2 part X that are eligible for removal, parts from position 2 and 3 since both are in a not desired position. In this example, lets choose the part from position 3 to be removed. Once the part is removed from tray, we erase it from the original map.

Original

Part Type	Position	Position	Position
X	1	2	3
Y	4		
Z	5		

Updated

Part Type	Position	Position	Position
X	1	4	
Y	2		
Z	3		
W	6		

Figure 6: Removing Position 3 from original map

4.8.2 Identify Parts that need to modify position

The first part of this task is to erase from the MAPS the parts that are already in the right position according to the update order, because there is no need to do any modification with them. Part

type x in the position one is already in the right place and because of that we can erase from both maps.

Original

Part Type	Position	Position	Position
X	1	2	
Y	4		
Z	5		

Updated

Part Type	Position	Position	Position
X	1	4	
Y	2		
Z	3		
W	6		

Figure 7: Part in the Desired position

After this step ,we loop through all the part types and positions from the original map to check if there is any part in the position where other part should be according to the update kit. If any part is found in this situation it is removed from the tray and erased from the original map. In the example the Part Y is in the position where a part X should be according to the update and it is necessary to be removed from the tray. Also the part X is in a position where a part Y should be and needs to be removed from the tray. Once all the removals are done , its also necessary to erase these parts from the original map.

Original

Part Type	Position	Position	Position
X	2		
Y	4		
Z	5		

Updated

Part Type	Position	Position	Position
X	4		
Y	2		
Z	3		
W	6		

Figure 8: Parts located in the position where other needs to be.

For the previous step if a part which was collected from the belt needs to be removed, a different approach is used to take this part out from the tray. Parts that are found on the bin are just removed and trowed on the ground, however parts that comes on the belt might not show up again so for that purpose we take this part out and place it in the opposite tray. After the placing in the other tray we erase it from both maps, and create a new one with this parts since we are going to use this new map to add it back in the final step.

After this step, whatever is left in the Original Map needs to have its position modified. From the example the part Z is the only one that the position should be modified,It need to be changed from position 5 to position 3.

Original

Part Type	Position	Position	Position
X			
Y			
Z	5		

⇨

Updated

Part Type	Position	Position	Position
X	4		
Y	2		
Z	3		
W	6		

Figure 9: Parts which the positions need to be changed .

Once the position is modified, it is also necessary to erase Part Z from both maps since it is in the right position and there is no modification needed to this part anymore.

4.8.3 Identify what Parts to add

Once he have been thru all the previous steps, whatever is left in the update order map, needs to be added to kit in its corresponded position. From the example the parts X and Y needs to be added since they were removed previously since they were in the position of other part, but this time they will be added to the right position. Another part that needs to be added is the Part W that was not in the original order and was added in the update.

Original

Part Type	Position	Position	Position
X			
Y			
Z			

Updated

Part Type	Position	Position	Position
X	4		
Y	2		
Z			
W	6		

Figure 10: Parts to be added in the update map

4.9 Orientation Challenge

To solve the Orientation Challenge, we had to calculate the angle by which the part is currently offset w.r.t its desired orientation. We get the current orientation of the part using the Logical Camera that we have placed over the AGVs. This pose is currently in the *logical camera frame*, so we convert this in world frame, say *pose-current-world*. We convert the desired orientation of the part from the kit tray frame to world frame as well, say *pose-desired-world*. Then we get the End-Effector pose in the world frame to get the offset end-effector has w.r.t. part its holding to. Next step is to convert these quaternion poses to *R-P-Y* values and rotate the *wrist_3* for the final Y offset which is the difference between end-effector's current yaw(world frame) and offset of part pose(current and desired in world frame). This final yaw value is fed to the end-effector so that the part's current orientation matches the desired orientation of the part. After this, the part is simply dropped by the gripper. Formulae for final rotation about the *wrist_3* is calculated as:

rot_wrist_3 = current_yaw_wrist_3 - (pose_desired_world_yaw - pose_current_world_yaw)

5 Challenges Faced in the Pipeline Design

5.1 Identify a good strategy to fix Quality check in Blackout

One of the problems faced by the group while developing the code was finding a good strategy to perform quality checking after blackout. In the previous approach we were checking the size o the message posted on the quality checking topic every time an part was placed in the tray , if the size of the message was different them zero it would mean that the last part added is faulty and it would replace it. However once blackout was over there was no way to ensure that if the size of the message was not 0 ,that the last one added was the faulty part. With this approach what was happening is that when the blackout was over and a faulty part was detected, the software would keep replacing

the last part added repeatedly even dough it was not a faulty part . Another problem is that the Quality camera does not provide the part type of the faulty part . The solution for this problem was stop checking the quality of a part every time a new one was added if a blackout occurred. With the new strategy once blackout is detected the quality check would only be done before submitting the AGV. To know the part type of the faulty tool in order to replace it , we used the logical camera placed in the top of the AGV to figure what is the part type that match's the position provide by the quality camera.

5.2 Idle Robot arm

In our original strategy the arm which was picking up parts from the conveyor belt would only be available to help to grab the tools that were not reachable to the other arm when all the desired conveyor belt parts were picked up. This would lead to high amount of wasted time because whenever there were nothing coming on the conveyor belt or a blackout was happening, the arm would just be waiting which it is not a good strategy since it could be using this time to help the other arm to grab the unreachable parts instead of just leave both arms waiting without performing any tasks. The solution for that would be create a flag that would set the robot arm picking up from the conveyor belt free whenever a blackout occurred or nothing were cumming on the belt. For this purpose the armGrab flag was created and a break beam positioned on the beginning of the conveyor belt to identify if parts were coming. If there was a long period of time without the break bean being activated or a blackout occurred this flag would be set to false which releases the arm to help the other arm . Once there is no blackout and the break bean sensor starts to identify new parts cumming from the belt the arm would stop helping build the kit and would comeback to pick up the parts from the belt since this part is a priority.

6 Group Roles and Responsibilities

Andre Ferreira - He was involved in the majority part of the project, his role was provide ideas , develop strategies to overcome the challenges and help to implement this strategies in c++ using ROS. Andre worked actively as explained previously in the following topics of the project: Basic pipeline (Belt and Bin part picking and placing, arm assignment); Dynamical arm assignment; Blackout Solutions; Quality checking; Order update; Orientation; Drop part challenge; Code integration; He contributed significantly in every part of the project.
An Li - His role was to provide, optimize, and implement the strategies in dynamically assigning robot arm workflow, conveyor belt product collecting, and also to support various aspect during project development. He adapted the basic ARIAC framework from the professor Zeid Kootbally and constructed the fundamental one for the project. The majority of his contribution is in the following topics: basic pipeline, conveyor belt product collecting, robot controller class modification, sensor deployment, orientation, and code integration.
Nakul Patel - He was actively involved in discussing the approaches for various challenges of the project. Main contribution was to develop the flipping part challenge as well as orientation challenge. Involved in brainstorming different ideas and finalizing one approach for the flipping part challenge.Moreover, he contributed actively in integrating the flipping part challenge to main pipeline as well as debugging the errors, if any. Also, running the main code multiple times to verify whether the kit building is as per the sequence and parts mentioned in .yaml file of the ROS package, and reporting the issues found to respective team mates, so that the functionality can be debugged and tested again to ensure everything works fine. Involved in making the README.md file for the ROS package for project submission.

Revati Naik - Her role was to work with the team to come up with different strategies for handling the various tasks in the assigned projects. Following the principle for pair programming, she worked well as a code navigator and contributed in developing a pseudo code. Her major contribution was in handling the challenges which included sensor blackout condition, testing the various types of sensors available and how their readings can be used during sensor blackout, strategy development and code navigation for flipping part and dropped part handling. She also worked on documenting the reports.
Sarvesh Thakur - He was involved actively in most parts of the project. He brainstormed ideas for challenges and implemented strategies through coding. He helped every member of the group in their individual tasks as and when required. He implemented strategies for flipping, drop part and orientation challenges as well. He helped integrating all the code chunks together. If any member required help in their part, he helped them to get through their part. Working as main debugger, he tested and debugged the pipeline through variety of yaml file configurations and testing each code section thoroughly.
Srujan Panuganti - He was involved in majority part of the project. His main contributions were, Collaborating with group members and brainstorming ideas in approaching solutions for the problems; Converting ideas into working code with little debugging and high readability; Making the code very generalized and robust by implementing reusable methods; integrating individually developed methods by group members into one single complete package; Keeping track of all programming changes with version control; Unifying all individual ideas and merging into the pipeline while making sure the pipeline is properly augmented. He made contributions towards every part of the project.

7 Results

The project has been successfully completed in due time. The package is built in a robust, generalized and agile fashion. The package handles the challenges like flipping parts, unexpected drops, faulty parts, sensor blackouts very well. The arms were able to build the kit for different configuration files on different combinations of AGV's. The package is tested for a single order with multiple shipments to build, which works really well. It has also been tested for multiple orders with single and multiple shipment configurations, the package works well. However, a potential bug was discovered while testing different configurations. One of the arm waits indefinitely for few specific adjustments in the config file which is unexpected. Technically, the implemented arm assignment strategy takes care of this problem which is proven to work perfectly for single order configuration. It is suspected that the problem is associated with the usage of AsyncSpinner, which is an area needs to be acquired knowledge about. Hence, this problem is considered to be one of the aspects of future scope of this project. The package also has an outstanding problem with orientation correction. Considerable efforts are spent to solve this problem which eventually gone into vain. Hence, it is also considered as an aspect of future scope.

8 Conclusion

There are still rooms for improvement for our system: part orientation while placing in kit trays; retrieve unnecessary but quality parts back to its own containers based on order updates instead of disposing of them; and programming more generically to adapt various situations. Every approach we use for the challenges also has to examine deeper for avoiding potential issues. In spite of the above inefficiencies, we learn and practiced several crucial topics as robotic software engineers:

ROS/C++ programming in an object-oriented manner, coordinate transformation with TF library, and trajectory planning by MoveIt interface. Strategies development and implementation for an autonomous system are also important for the robot industry. Last but not least, we have a better understanding of the workflow of a medium software project, how to control its version as well as maintain it. Overall, it was an excellent learning experience.

9 Thoughts about the class

The class fulfilled the purpose expected and all the students were able to learn a lot of ROS and Agile Robotics, which is going to be very useful in our future careers. In our opinion, there are some points that we consider that can be improved in order to increase the learning and the performance of the students in the projects.

1. The first point are the in class exercises that sometimes were not done in the class because of time. It is a great opportunity for students to develop doubts and solve it right away in class, instead of figuring out the problems at home and solve this problems thru email which is much harder.

2. The second point is explain better about *Async* spinner and some other ROS tool like *MoveIt* that are vital for the development of the project.

3. It may be better if students are allowed to form groups of their choices, definitely with a certain number of students in each group, because this way it would be their responsibility if part of the group does not cooperate.