Journal of Advanced Research in Intelligence Systems and Robotics
Volume 5, Print Issue 1 - 2023, Pg. No. 9-15
Peer Reviewed Journal

Research Article

Advancements in Automated Warehouse Systems: Streamlining Operations through Innovative Technologies

Atul Jha[1], Aniket Raj[2], Animesh Kudake[3]

[1,2,3]Department of Computer-Science and Engineering, Chandigarh University, Gharuan, Mohali, Punjab, India.

I N F O

Corresponding Author:
Atul Jha, Department of Computer-Science and Engineering, Chandigarh University, Gharuan, Mohali, Punjab, India.
E-mail Id:
jhaatul258@gmail.com
Orcid Id:
https://orcid.org/0009-0005-0172-171X
How to cite this article:
Atul Jha, Aniket Raj, Animesh Kudake. Advancements in Automated Warehouse Systems: Streamlining Operations through Innovative Technologies. *J Adv Res Intel Sys Robot* 2023; 5(1). 9-15.

Date of Submission: 2023-03-03
Date of Acceptance: 2023-04-12

A B S T R A C T

We've all ordered goods online, it arrives with the touch of a switch. Behind the scenes, however, there is a complicated logistics system comprised of many individuals that workday in and day out to deliver the goods you requested. As individuals, we are prone to making faults and cannot always perform at peak efficiency. So, what if all of this was automated? Robots, unlike humans, can work at optimum efficiency all the time and are less prone to errors. So, based on the same concept, we propose to you a fully motorized warehouse. As a firm believer in the Fourth Industrial Revolution, much in this warehouse is computerized!

Keywords: Python, Industry Automation, Ubuntu Javascript

Introduction

The concept is presented as an abstraction of a warehouse management system created in Gazebo, a 3D dynamic simulator that effectively simulates robots in challenging conditions. The arena serves as an automated warehouse where necessary packages must be transported to various locations across a city. Since automation is a major component of Industry 4.0 in this area, there will only be two industrial robotic arms in the warehouse that the teams will use. One robotic arm will choose the items from a shelf and set them on a conveyor belt when the requirements are transmitted to the warehouse, the second robotic arm at the end of the conveyor belt will pick up these products from the conveyor and store them into bins. Each bin denotes a location where the shipment will go. The user will receive email alerts as the products are shipped from the warehouse, letting them know that the package has left the warehouse.

The delivered packages have their own priorities. Packages with a higher priority are meant for pandemic or natural disaster disasters. For general use there are other programmes with lesser priorities. In this theme, participants create their own conductor (controller) for their warehouse to make wise judgements to deliver high priority products as quickly as possible, much like a conductor in an orchestra.

Proposed System

Below represented is the implementation of Task 5, for better understanding project implementation, we have divided the project into few sub tasks and worked equally as a team.

Sub Task 1 (Colour Detection)

For colour detection or object detection we have used QR decoding. First, we have taken the data from 2D camera from the topic /eyrc/vb/camera_1/image_raw and then we cleaned the image using threshold (cv2.threshold). Further we decoded it into useful information. The QR decoder returned few arguments like data, type, rect and polygon. In our task implementation we used rect and data in which rect returned four arguments x, y, w, h where x, y are the corner pixel values where w is width and h is height of the detected box. We used these pixel values to determine the position and colour of the box. To use this information further we encoded it in two different ways as follows:

ur5_2.py(Code implementation)

box_color=[]

for i in qr_result: x,y,h,w=i.rect

if (x>100 and x<200) and (y>250 and y<400): box_color. append((‘packagen00’,i.data))

output of ur5_2 box_color=[(packagen00, red), (packagen 01, yellow), (packag en 02, green) and so on]

Sub Task 2 (Pushing Data on Inventory and Incoming order Spreadsheet)

By the QR detection done in sub task 1 we made an array of tuple. We then sorted data by looping over this array of tuple using package color and package position. Using this data, we updated the inventory spreadsheet.

For taking the online incoming order we subscribed the mqtt topic called as /eyrc/vb/VsMyPsSr/orders and after sorting the data (we obtained by subscribing the mqtt topic) is then updated on the incoming orders spreadsheet.

Sub Task 3 (Priority Check and order dispatch)

For priority check we first subscribed the topic/eyrc/vb/ VsMyPsSr/orders in the node ur5_1.py that takes the online order, then we put the order_id in a list which is defined globally.

reference code

#This is the callback function for topic ‘/eyrc/vb/Vs My PsSr/ orders’

def mqtt_orders_1(self, data): global lst, count,timer order_list = data.message

goal_message=(order_list.split(“, “))

item=(goal_message[5])[9:len(goal_message[5])-1]order_ id1=(goal_message[2])[13:len(goal_message[2])-1]

order.append((order_id1,order_list)) lst.append(order_id1)

After that we subscribed a new topic /eyrc/vb/camera_1/ image_raw that works parallel with the previous topic. The main motive of this topic is to take the list of orders from the online order topic and operate on the list (“”) of data simultaneously. While processing the orders there may come a time when we can have multiple orders of different priorities. To process this is we check the priority of the order. We have used order_id as a key to sort the orders in high to low priority. As the order id is a string so we used sort( ) function for sorting. By doing this it is assured that High Priority orders are processed as quickly as possible, then Medium Priority and then Low Priority orders are processed.

reference code

#This is the callback function for topic ‘/eyrc/vb/camera_1/ image_raw’

def mqtt_orders_2(self, data): global lst

lst.sort()

while lst: lst.sort() order_id=lst[0]

if(order_id==’3001’):

self.packagen02_box(‘3001’) #This will pic the order and place it on conveyer

self.msg_order_1(order_list) #updating order in OrdersDispatch spreadsheet

lst.remove(order_id) elif(order_id==’3002’):

self.packagen10_box(‘3002’) self.msg_order_1(order_list)

lst.remove(order_id) elif(order_id==’3003’):

self.packagen20_box(‘3003’) self.msg_order_1(order_list) lst.remove(order_id)

.....and so on

Further we Dispatch (pick the package from shelf and place it on the conveyer) the order using ur5_1 arm and then we update the Orders Dispatch spreadsheet.

Sub Task 4 (Ur5_2 and Shipping Orders)

As we have encoded the data in sub task 1 from the topic (eyrc/vb/camera_1/image_raw). This data is stored in an array. Further we subscribe the logical_camera_2 topic (/eyrc/vb/logical_camera_2). The callback of this topic gives the model name and position in the frame of logical camera when any object comes under it. Using the position of the incoming model we stop the conveyer belt and then using the data we encode in subtask 1, we loop over that array. After that we match the model’s name with package

name and identify the colour of the package followed by Shipping (put them into the bins according to the colour of the package) the order using ur5_2 arm and update the Orders Shipped Spreadsheet.

reference code

for i in reversed(list_1):

model = data.models[1].type pkg_n,pkg_clr=i

if(pkg_clr==’red’ and model==pkg_n):

if y>-0.03 and y<0.02: #This condition will stop the conveyer belt

self.conveyor(0)

self.pkg1(x,y,z) #This will place the package from conveyer to bin

elif(pkg_clr==’yellow’ and model==pkg_n): if y>-0.03 and y<0.03:

self.conveyor(0) self.pkg2(x,y,z)

elif(pkg_clr==’green’ and model==pkg_n): if y>-0.04 and y<0.03:

self. conveyor(0) self.pkg3(x,y,z) and so on

Sub Task 5 (Dashboard)

After properly implementing the above four sub tasks and updating the Incoming Orders, Orders dispatched & Orders Shipped sheets, these three sheets are used to update the Dashboard sheet of the Inventory management Master sheet which will be further used in the Dashboard web page that we made using Github Pages. Using Github pages we made a dynamic dashboard which includes a map which tells the location of the incoming orders a graph which tells total time taken to ship a package and a table which shows all the necessary information like Order ID, Item, Priority, Quantity, City, Dispatched, Shipped, Order Date and Time, Dispatch Date and Time, Shipped Date and Time, Time Taken.

Now to update our webpage using values from Dashboard sheet we use that sheet as a JSON endpoint which gives the entire content of the sheet in a JSON format which is then used in our webpage. Update of the JSON data using AJAX (Asynchronous JavaScript and XML) request is then done, which makes requests to the server without reloading the page, receive and work with data from the server so that we don’t have to refresh the webpage.

Rqt-Graph

Workflow

MQTT

With the use of MQTT Warehouse is easily manageable across the globe. MQTT can scale to connect with millions

3. Online Placer

This node is responsible for placing order on mqtt topic.

4. Action Client

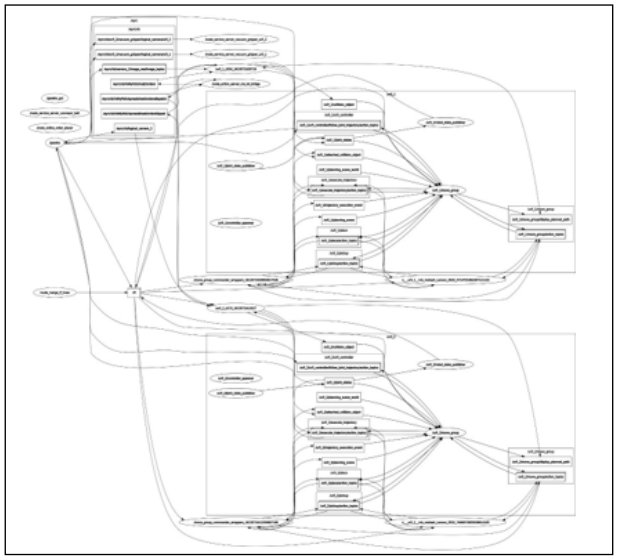

Figure 1

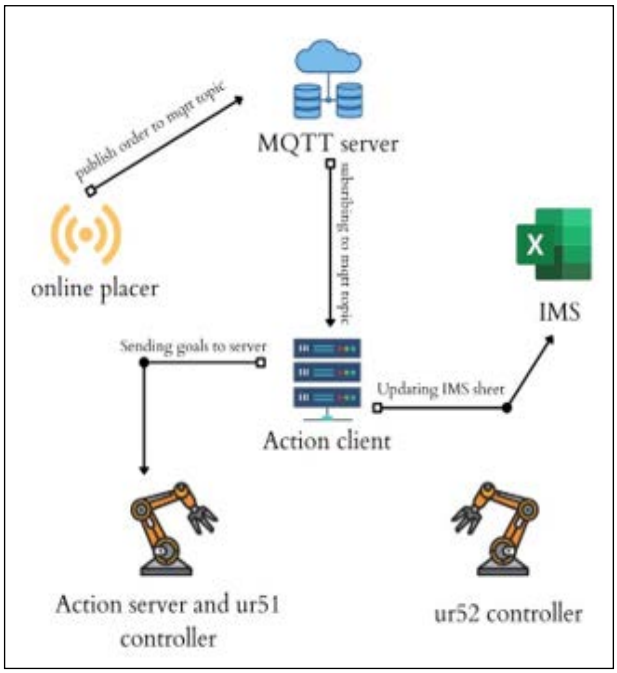

Figure 2

of IoT devices. Many IoT devices connect over unreliable cellular networks. MQTT’s support for persistent sessions and reduces the time to reconnect the client with the broker.

Bi-Directional Communications

MQTT allows for messaging between device to cloud and cloud to device. This makes easy broadcasting of messages to groups of things.

The Action Client subscribes to the topic /eyrc/vb/eTrzIron/ orders for online orders and as soon as an order is placed on this mqtt topic, the action client updates the incoming Orders sheet of IMS and create a goal which contains order details and then this goal is sent to the Action server for further processing.

Action Server And ur5_1 controller

This node receives goals from the action client, this node uses the concept of threading to receive multiple goals, the idea behind this is that the main thread will always be running and ready for receiving the goals, another thread which is "order_handler" which processes the goals separately.

Now as when the very first goal is received, it starts the order_handler thread. Also, this goal and all upcoming goals are received in main thread and these goals are separated in different lists according to their priority to process them later in order_handler thread.

2d Camera

This is responsible for mapping packages with their colour. For mapping the colours, we first take the image of all the packages and then crop this image so that only the region of interest is present in the cropped image. This cropped image is used to get qr_data. For e.g., if we want the color of packagen00, we first take the image, now we know that packagen00 lies in a square made with points (100, 300) [top left vertex] (250, 450) [bottom right vertex] so this is the region of interest for packagen00. Now we crop this image to get only region of interest [ROI] and send this cropped image with only packagen00 to function get_qr_data() which gives us the color of package00. Now this color is mapped with box_name and details in a global list according to the priority and these lists are used while processing the goal.

Main Thread

It updates the inventory sheet of the IMS. With the use of 2d camera, main thread detects the packages and saves them in lists for further processing during dispatching and shipping. It separates all the goals coming from the server in three criteria based on their priority which is used by order_handler thread while processing the goals.

order_handler thread

This thread processes the goals but there is only on one condition, that the higher priority orders should be processed first. For this we first check that if there is any high priority order to process and we find one then it is processed and if not, it than checks for medium priority orders and hence these orders are saved in lists by main thread.

Now for processing any certain goal, at first it's priority is checked and then we use the data of our inventory which is saved in a dictionary by 2d camera to get the position (row and col) of that particular package and then a saved trajectory to that position is sent to the ur5_1 to pick and place it on the conveyor belt and so the dispatching of the package is done and this data is updated in orders Dispatched sheet of IMS and an email alert is sent to the user giving a signal, that the order is dispatched.

Package detection

As the ur5_1 places a package on the conveyor belt, this package comes towards ur5_2 and as soon as it reaches in the range of logical camera, it is detected by the program and stopped at a particular position (home position) and then ur5_2 grips the package. Now package name as detected by logical camera is used to retrieve its colour from the data saved by the 2d camera and as we get the colour of the package a saved trajectory to that colour bin is sent to ur5_2. After this, the ur5_2 picks and drops the package in desired bin and thus the package is shipped and this data is updated in the order Shipped sheet of the IMS and an email is sent to the user sending a signal, that order is shipped.

Planned implementation of the project

So now, let's get into specific nitty gritty details of the implementation.

Incoming Order

The orders are sent via the MQTT Protocol. MQTT or Message Queue Telemetry Transport protocol is a publish/ subscribe, extremely simple and light weight protocol. This publish/subscribe model allows MQTT to communicate as one-to-one, one-to-many and many-to- one. It is popular among cases where there is low bandwidth, high-latency, or unreliable networks. MQTT also provides us with a parameter called Quality of Service (QoS).

Let us discuss few of these terms in some details

1. **Publisher:** Publishes the message
2. **Subscriber:** Receives the message
3. **QoS:** MQTT protocol provides us with 3 QoS, they are as follows.
4. **QoS0:** Message is sent but no confirmation is received.
5. **QoS1:** Message sent at least once, but duplicates may be received.
6. **QoS2:** Most Reliable one, here duplicates are controlled by making sure message is sent only once.

There is also a concept of MQTT Topics (which is like ROS Topics) and MQTT Broker (Broker serves as middleman through which messages are sent and received) which are not discussed in detail here. So, the incoming orders are sent via the MQTT protocol, using QoS0 and under the

topic '/eyrc/vb/iOeCqZLl/orders'. These orders are received by the ROS node 'node_action_server_ros_iot_bridge'.

After receiving them, it publishes them to an ROS topic '/ros_iot_bridge/mqtt/sub'. This is done to make the other ROS nodes listen to what the incoming order is, act accordingly.

The ROS node 'node_action_server_ros_iot_bridge' also publishes a Google sheet.

1. There are three different priorities of orders:
2. **High Priority (HP):** These are the orders for medicines
3. **Medium Priority (MP):** These are the orders for food packages
4. **Low Priority (LP):** These are the orders for clothes

One of the aims is to reduce the delivery time such that HP order takes the lowest time and LP order can take the most time to be delivered among these 3 priorities.

In other words, the delivery time for these different priority orders should be as: HP < MP < LP.

Identifying Packages from Shelf

The process of identifying packages from the shelf runs independently from the Incoming Orders. So, whether we are receiving an order or not, a camera scans the shelf and saves a list of all the available packages along with its priority as a key value pair. This is done only once per run.

The ROS Node used to do this is node_package_info. The packages are colour coded as well as have QR code on them, to help us identify its priority.

The colour code - priority relation is as follows:

1. **Red Colour:** High Priority
2. **Yellow Colour:** Medium Priority
3. **Green Colour:** Low Priority

So, to identify the package and its priority we have two options.

The first, to figure out the package colour by isolating other colours. The second, to read the QR data on the package. We have selected the second approach as it seemed much simpler and faster.

Processing the Camera Feed

As you can see from the above image the camera feed is a bit dull and smooth, basically not great to read the QR Code directly.

As a result, a bit of pre-processing was required. So here are all the pre-processing steps we followed:

1. **Converting to Gray Scale:** Since we are going to read the QR code, there was no need for the colours in the image. So, we stripped the image from its colours.
2. **Blurring It:** It may seem a bit odd to blur out the image, but it helps to reduce all the noise present in the image, we are left with only the major constituents of the image. The Blurring technique we applied was Gaussian Blur.
3. **Binarizing It:** Binarizing means converting the image to black and white i.e., each pixel can either have a value of 0 (black) or 255 (white), nothing in between.

The Binarization aka Threshold technique we used, was an adaptive thresholding, consisting of adaptive thresh Gaussian method. Binarizing makes identifying QR code very much simpler.

Dispatching the Package

Dispatching here means, picking up the ordered package from the shelf and placing it on the conveyor belt. Dispatching is done by a UR5 robotic arm. This arm has a vacuum gripper as its end effector (EE). The ROS node which controls this arm is named node_ur5_1 and it is subscribed to the ROS Topics '/pkg_task5/package_info'. and '/ros_iot_bridge/mqtt/sub' hence it has all the necessary information like incoming orders and priority of the packages on the shelf.

How to Dispatch HP Orders Packages First?

Once an order is received it saves the order in a dictionary and saves it in a separate list based on its priority. There are 3 different lists for 3 different priority orders. After saving the order, it will check the list of High Priority packages, if it is not empty (meaning there is a high priority order left to dispatch) it will start processing it. And if the list is empty, it will check the list of Medium Priority packages, again, if it is not empty, it will process it. Lastly, it will check the list of the Lower Priority packages and if they are not empty it will process it. The whole process strictly follows the above order. This helps us to dispatch the higher priority packages first, medium priority packages second, lower priority packages at last. Thus, helping in the goal of delivering HP orders first, then MP, at last LP

Moving the UR5 Arm towards the Package

Once the node selects which order to process, it will find the required package from the available packages. After figuring out which package will be used to fulfil the current order, it will play the corresponding saved trajectory to reach there. What is a saved trajectory and why is it used? A trajectory is the path that the arm will take to reach the destination pose. This trajectory can be calculated while executing or it can be played from a saved file, when played from a saved file it is called saved trajectory. The reason of using it is to save time. Calculating trajectories can take anywhere from 5 seconds to 90 seconds. Furthermore, sometimes the trajectory planner may fail to find a solution. Hence, using a saved trajectory is a much more reliable and faster method. After playing the saved trajectory to

reach the package, the arm activates the end effector (EE) in this case (a vacuum gripper) which grips the package with the help of suction.

After gripping, the node plays yet another saved trajectory to put the package on the conveyor. On reaching on top of the conveyor it deactivates its EE and drops the package on top of the conveyor. With this a package is dispatched successfully. The node then uploads the dispatched package information like, dispatch date and time along with other order information to the google sheet by sending it as a goal to the 'node_action_server_ros_iot_bridge'. This node also publishes the information of the package which it just dispatched on a ROS topic called "/eyrc/vb/onConveyor ''.

Shipping the Package

Shipping here means putting the package inside a bin from where it can be delivered. Shipping is done with the help of yet another UR5 Robotic Arm and a logical camera. The ROS node used to do the shipping is called node_ur5_2. And this node is also subscribed to ROS topic '/pkg_task5/package_info'. Hence it has the package info, i.e., which package name has what priority. ROS topic "/eyrc/vb/onConveyor" is subscribed by this node, hence it knows which package is currently on the conveyor, its order details. A logical camera is placed on top of the conveyor and is pointing on the conveyor belt. As soon as it detects a package, it slows down the conveyor belt a bit and triggers the UR5 to activate its vacuum gripper and grab the package. After grabbing the package, since it has the order details of the package it finds out its priority and puts it inside the respective priority bin. Just like for dispatching the package. Here also, we have used saved trajectories to save time.

What are Priority Bins?

There are 3 different bins, for 3 different priority orders. One for HP, one for MP and one for LP. The node then uploads the shipped package information like, shipped date and time, estimated date of delivery along with other order information to the google sheet by sending it as a goal to the 'node_action_server_ros_iot_bridge'.

Results

At present we have made our automated warehouse system and programmed it using Python and built its used open-source modules for simulation. We have to work on all the IoT protocols and connect the whole system so that all the data will be presented on one Dashboard as well using IoT terminal admin which will be able to command UR5 arms to perform assigned tasks.

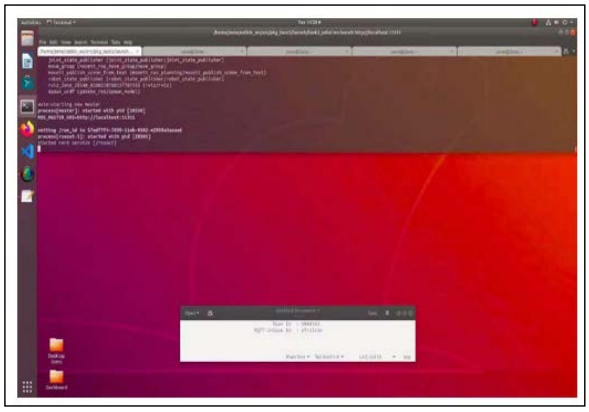

Figure 3

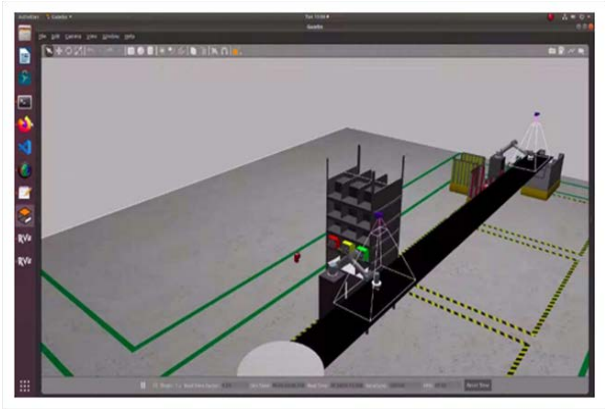

Figure 4

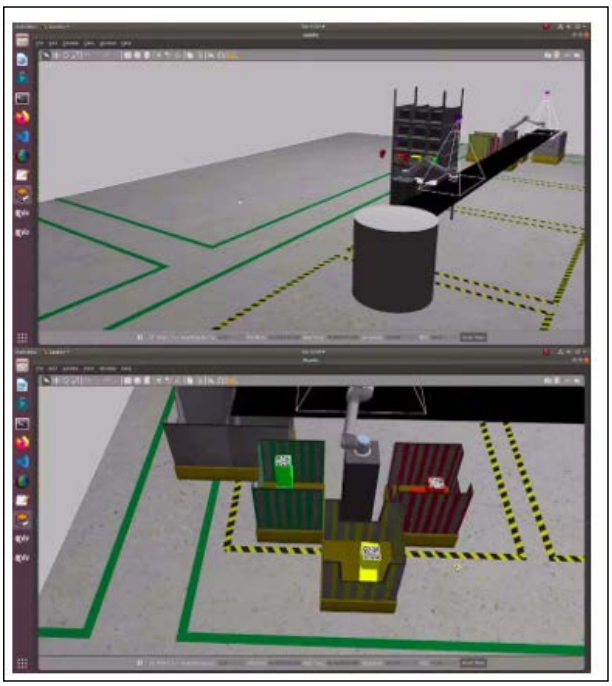

Figure 5

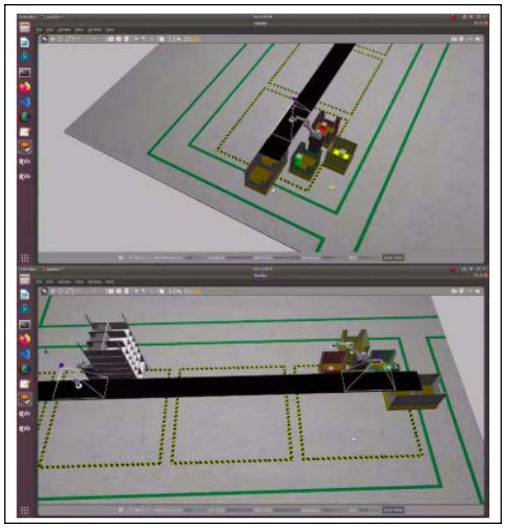

Figure 6

Conclusion

We've now effectively implemented entire projects and created the entire computerized simulation environment. We believe that we managed to clarify every detail of the project.

References

1. http://wiki.ros.org/Documentation
2. https://answers.gazebosim.org/questions/ https://askubuntu.com/
3. https://portal.e-yantra.org/storage/xyBeoIQDWX_vb/home/home.html
4. https://stackoverflow.com/
5. https://www.w3schools.com/
6. https://www.mkdocs.org/#getting-started
7. https://github.com/ros/ros_tutorials
8. http://docs.ros.org/en/melodic/api/moveit_tutorials/html/index. html