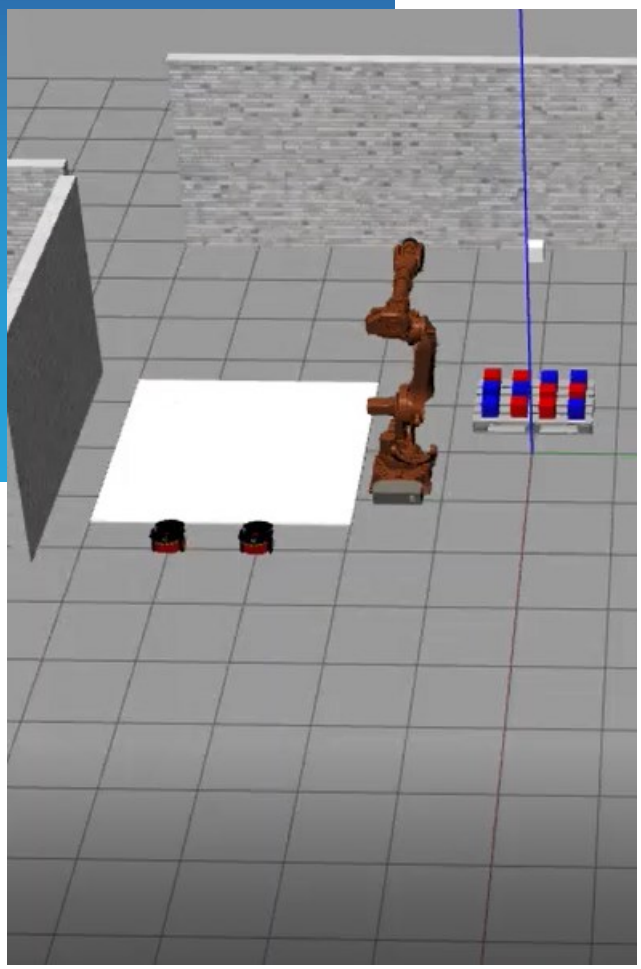


REPORT

Smart Warehouse

University of Naples Federico II

20/07/2020

Eliana La Frazia

Andrea Cavaliere

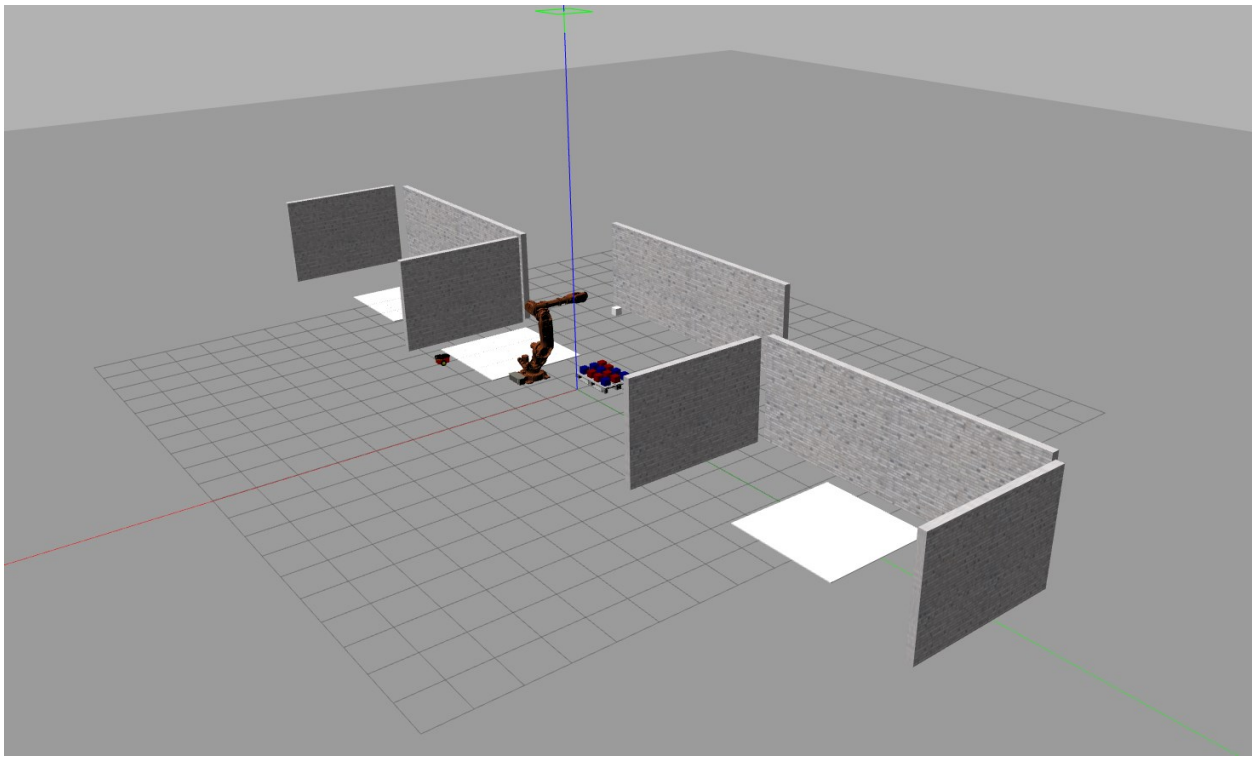


Figure 1 Snapshot from simulation with only one AGV

Abstract

In the following report the students Andrea Cavaliere and Eliana La Frazia present the results achieved in the project "Smart Warehouse 2".
Purpose of the project is the realization of an autonomous warehouse environment where an industrial robot manipulator has to depalletize a pallet of at least 10 boxes (5 red and 5 blue) and at least one AGV is in charge to bring them to specific platforms.

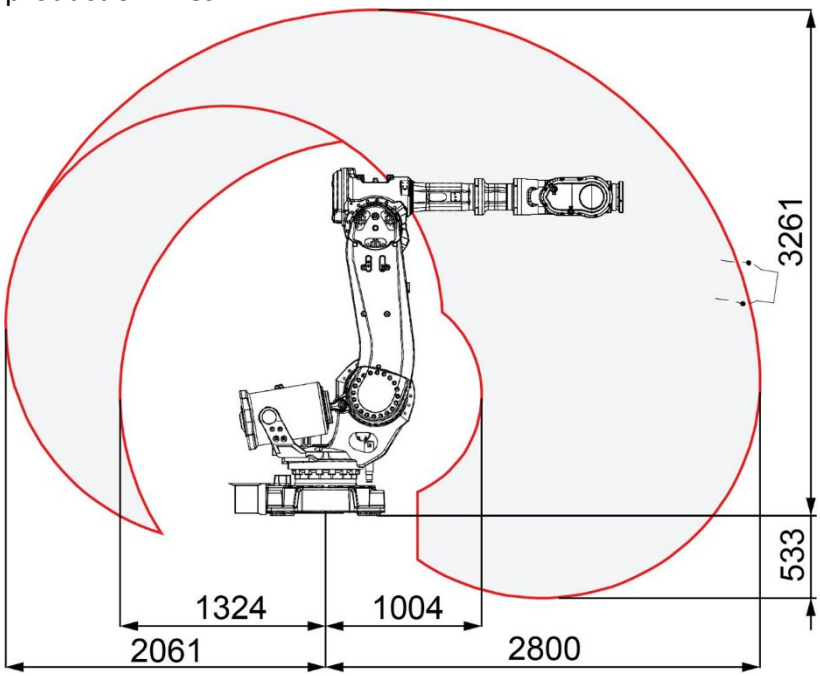
Summary

1 Robots

Various types of robots have been investigated to perform the two different tasks. The manipulator should be robust and with a large working space and the mobile robot should be efficient and versatile.

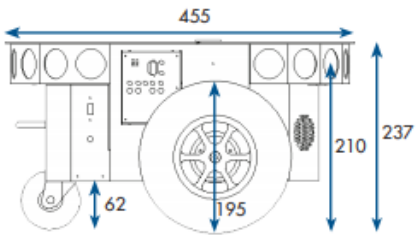
1.1 ABB IRB6640

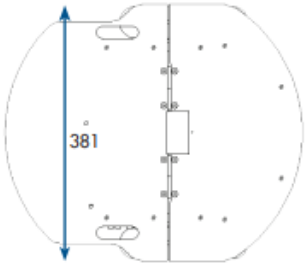
The ABB irb 6640-185 was chosen mainly because of its wide range of action. As the robot can bend fully backwards, the working range is greatly extended allowing it to fit well into dense production lines.



1.2 Pioneer

Pioneer 3-DX is a small lightweight two-wheel two-motor differential drive robot ideal for indoor environment. The model we have chosen is equipped with a 2D LiDAR sensors for Indoor (0.1 m - 4 m), placed on the front of the plate, leaving the necessary space to carry the boxes.





2 Implementation

2.1 Manipulator and Camera

In order to spawn the model of the ABB IRB6640 in the gazebo environment and to control it, we made an adaptation of the package abb_irb6640_support, available in the ABB stack and renamed "abb_irb6640_description". We changed the urdf file adding the dynamics parameters and the configuration file adding the controllers.

Then we created a new package in order to interface the Moveit! control plugin (implemented in the ROS Industrial ABB stack) with gazebo. To do so we started from a working example found on internet of the ABB IRB120 and adapted it to our robot.

Main advantage offered by Moveit! Toolbox is the possibility to define only a goal position and it will plan a trajectory (in this case a Cartesian trajectory) in compliance with dynamic and kinematic constraints properly defined in the so-called "Setup Assistant" window. In this window another important step in the development of our robot is the compilation of the Self-Collision matrix, which allows us to define which pairs of link can be disabled from collision checking due to the fact that they are always in contact (adjacent links on the kinematic chain), never in collision or maybe in collision when the robot assumed a default position.

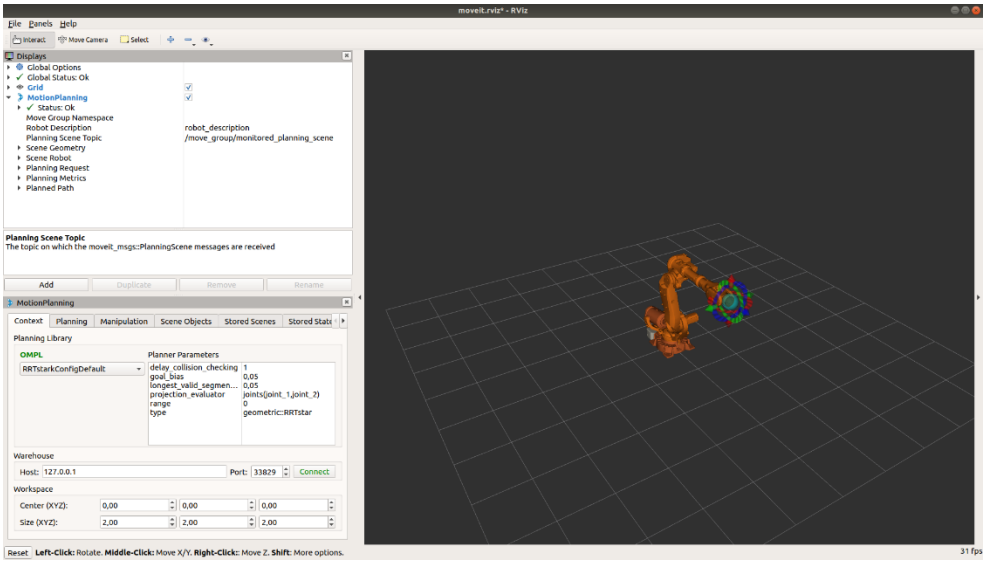


Figure 2 Rviz Moveit! window

Figure 1 shows a window from Rviz software with Move-it! settings. Thanks to it is possible to directly control the end-effector pose by dragging it with the cursor. This window won't start during the simulation but it could be opened following the instructions in the README.MD file.

To visualize and correctly find the position of the boxes on the pallet, a RGBD camera (Kinect like) has been used. Many tools from OpenCV ROS-wrapper has been to evaluate the correct shape and color of the boxes and then thanks to depth-sensor obtain their distance from the camera in order to establish their position with respect to the camera frame.

As we know this kind of architecture, called **eye-to-hand**, needs to transform the points defined in the camera frame to points in the robot frame and this purpose has been successfully achieved thanks to "tf" tools and "Eigen" C++ library.

Currently the camera node transforms the coordinates of the chosen boxes to grasp in a common world frame called "map" (this frame is shared with all the robots in the environment) and then thank to a ROS-Action paradigm(in which the camera plays the client role) it sends the

result to the robot. Again, the manipulator will transform the incoming data with respect to its base frame and then will begin the pick-and-place routine.
After understanding the logic behind the implementation, we go down to the technical details. We implemented the package *smartwarehouse_opencv* which uses the library OpenCV. After the calibration, the camera detects all the objects whose colour is in the defined range and with a rectangular shape. The implemented algorithm allows the camera to detect boxes of different size (obviously the dimensions must differ slightly from the starting ones to work with the rest of the project) because the camera doesn't have to know the dimensions a priori and neither the number of the boxes but it can calculate them by itself. It would be better if the boxes are placed with a gap between them. The camera node is subscribed to a Gazebo topic in order to achieve the gazebo models names and once it detects a box position it calculates the box model nearest to this position and identifies its name.
We also implemented a node to manage the manipulator pick and place task that is the action server. The ABB action server node receives the goal from the camera node, consisting of the box position, colour and the gazebo model name. The ABB node provides to reach the box position using Moveit!. To make the attach between the end effector and the box we use a package in which is implemented a virtual link. This service requires to know the gazebo models names (provided from the camera node) and the boxes to be dynamic, so we had to modify the sdf file of each model and the warehouse2.world.
We assumed to have two conveyor belts, each of them takes a different colour, so the ABB manipulator places the boxes in sequence on them. The sorting policy of boxes consists of selecting before all the red boxes and then the blue boxes. After the end of the depalletization, the manipulator is immediately ready to depalletize another pallet.

2.2 AGVs

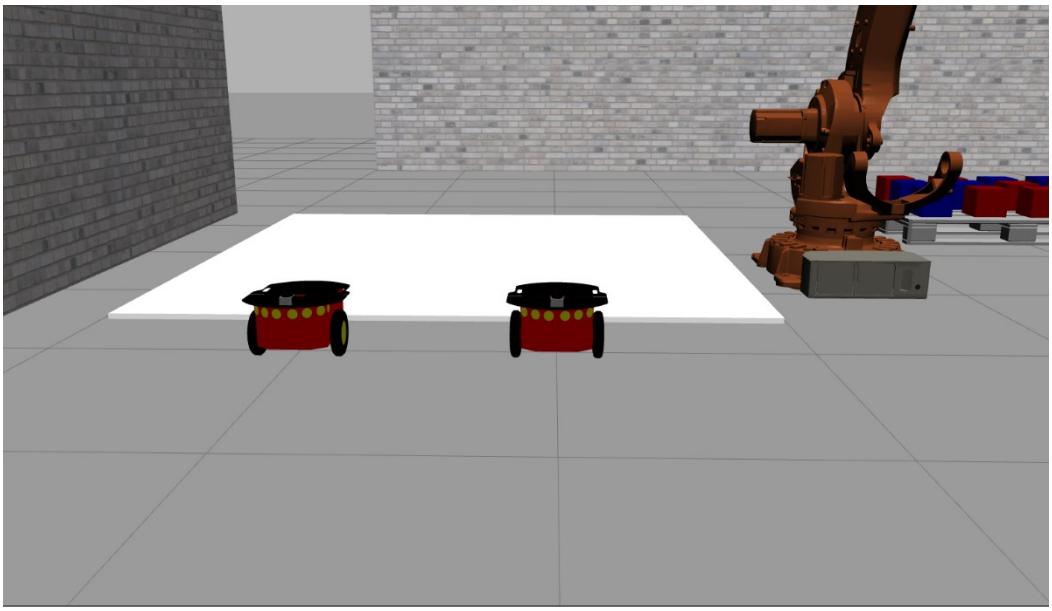


Figure 3 Snapshot of the two Pioneers in their idle position

Regarding the Pioneer, from the work of Rafael Berkvens and Mario Serna Hernández we got the Pioneer 3DX description and control packages.
Then we created the p3dx_move package with the implementation of move base node, AMCL and FastSLAM (thanks to gmapping) algorithms.
The task of the mobile robot(or robots) is to deliver the boxes to their corresponding platforms. We assume that the boxes will be directly placed above the robot when it is ready.

At the beginning, since the robot has no knowledge of the environment, we gave him some relevant pose inside the warehouse using move base and gmapping, in order to create a map of the place. Since it is an open space, the maps has no sufficient landmarks to create a perfect map of the environment(in fact during the simulation we added a “fake wall” also to reach a realistic description of a typical environment) but the resulting map includes all the position of interest to deliver the boxes.

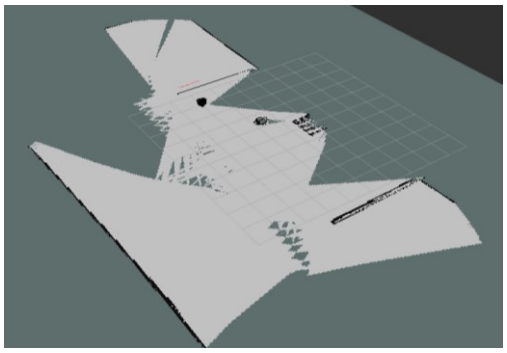


Figure 4 Resulting Map

After that we substituted the FastSLAM node with the AMCL and let the AGV drive autonomously.
To realize a robust infrastructure which can support a routine with one or two Pioneers we created a "simple navigation node" for each Pioneer and a sort of "manager node" that coordinates the delivering and communicates with the ABB manipulator thanks to a publish subscribe policy where every time the manipulator places a new box on one of the conveyor belts it publishes a string that will be parsed from the *pioneer_manager* in order to get model name and colour. Thanks to this information the node will update a queue with the names of the models and their color. The first information is needed to implement the virtual link between the boxes and the top plate of the Pioneer and the second one to choose the right Pioneer to whom assign the box.
When only one Pioneer is available the Pioneer will be aware of that and then will just assign the boxes in the exact same order in which it got them.

2.3 Results

We give the opportunity to tune the colour of the boxes, just in case the user wants to redefine the exact shades of colours.
Thanks to the settings defined in the warehouse2.launch file and the usage of tf and Eigen tools robot and camera can successfully share the positions of each box even if they are not in their original positions.
The robot only needs to know where is each one of the conveyor belts to correctly place the boxes on them(in a realistic scenario the robot doesn’t have to place the box along a fictional line but always in the same position instead the conveyor belt will have to drag it all the way to the Pioneer.
The boxes sorting policy exploits 2 critical information. The first one is the number of blue and red boxes while the second one regards the height of each box from the ground, then the resulting algorithm is the following:

```
while(pallet is empty){
        Wait;
        }

If (number of red ≥ number of blue) {
        Choose the red box on the highest position
        If (a blue box is above the chosen red box) {
                Choose the blue one;
                }
        Else{
                Keep choosing the previous box;
                }
        }
Else {
        Choose the blue box on the highest position;
        If (a red box is above the chosen blue box) {
                Choose the red one;
                }
        Else{
                Keep choosing the previous box;
                }
        }
```

Table 1 The pseudocode describes the box sorting policy made by the camera

The p3dx_manager node loads all the necessary information about the position of each conveyor belt and both the delivery stations.
At the beginning of the simulation, based on how many pioneers are available it will choose the delivery policy of the boxes as following:

```
If(There are two Pioneers){
        While(running){

                While((no new boxes)OR(Both Pioneers aren't ready)){ wait; }
                If((still a red box)AND(p3dx_2 is ready){
                        Attach the red box to the Pioneer and send it to red_station;
                        }
                If((still a blue box)AND(p3dx_1 is ready){
                        Attach the blue box to the Pioneer and send it to
                        blue_station;
                        }
        }
}
Else{ //there is only one pioneer
        While(running){

                While((no new boxes)OR(p3dx_1 isn't ready)){ wait; }
                If(next box is red){
                        Attach the red box to the Pioneer and send it to red_station;
                        }
                If(next box is blue){
                        Attach the blue box to the Pioneer and send it to blue_station;
                        }
        }
}
```

Table 2 This pseudocode describe the delivering policy made to p3dx_manager node

3 Testing

3.1 Depalletize: typical scenario

We tested the algorithms:

- Changing the number and positions of blue boxes and red boxes
- Adding other boxes after the end of the depalletization
- Putting all red boxes or all blue boxes on the pallet.
- Moving and rotating the pallet and/or the camera

The position recognition algorithm and the pick and place algorithm work also if the boxes are placed one above the other but it's hard to test it in simulation because of the boxes are dynamics.

During the testing there were some failures of trajectory planning between two distant points, these failures are related to the simulation environment because all the points the robot has to reach are in the working space and are absolutely reachable (both position and orientation assigned).

For this reason the failures are supposed to be rare so if the planning fails during a pick phase the robot try to return in start position and the action client resend the box position, instead if it fails during a place phase the robot try to return in a start position and to replan the trajectory to reach the place pose and if it fails again then it aborts.

3.2 Obstacles avoidance

Since a warehouse is a semi-structured environment where human beings can stand on the robot's navigation path we tested how the mobile robots react to unexpected obstacle(not marked on the map), and then tried to tune the planner parameters (like simulation time of the local parameters, tolerance, dimensions of the local map) to get the best trade-off between obstacle avoidance(to ensure operators safety) and efficiency.
Since the local map save the laser readings into the local map it could happen that even if the obstacle gets away from the path the robot tries again to “avoid” it. This can happen near a destination pose like the base or one of the platforms. Obviously, it can happen only one time since the local map is updated after a while.
the equipped sensor only detects obstacle on the (x,y) plane of the sensor frame. For this reason, any obstacle placed under or above this plane will be ignored by the path planner.
the consequences of this kind of limitation could lead to a collision between the robot and the obstacle. A hypothetical solution could be to move the sensor to the front of the robot. Even if it seems to be a good solution, it might result in a very particular problem. Just think about a hypothetical scenario where there are obstacles like suspended pipe above the ground, something like a table or any obstacle placed at the same height of the box on the robot, they might bump into the box an make it fall from the robot and the latter won’t even notice and will continue its task.

4 Details

This project has been completely developed in ROS Melodic on Ubuntu 18.04 and the simulation runs on Gazebo 9.0.
Many tool of the visualization software Rviz has been used during the preliminary stages of the project.
85% of the files in the repository has been written in C++ and 15% in Python.
Follows the list of all the toolbox included in the project, starting with the ones used by camera and manipulator

- Moveit!
- OpenCV
- Tf
- Eigen

Navigation tools used by Pioneers:

- Move-base
- Map-server
- Gmapping
- AMCL

4.1 Limits

Since the warehouse is a semi-structured environment it’s user’s duty to realize a realistic environment.
Some examples:

- pallet e station has to be placed inside the workspace of the manipulators and if they were not, the robot would abort.
- camera can be placed anywhere it can successfully see the pallet. It is clearly better to position it so that it can see the pallet from above.
- For any new environment the robot needs a related new map, this is not true if the user makes only “small” changes that preserve the warehouse structure(for instance you can easily switch blue and red stations, updating their position values in the ROS

parameter server will allow the Pioneers to successfully reach them without any problem.

Relatives position between camera and conveyor belts are particularly relevant.
In Figure 5 you notice the camera can see part of the station so if the manipulator places a box (in this particular scenario a red one) too near the base, it will be detected from the camera node and at some point will send again its position as a goal to the manipulator.

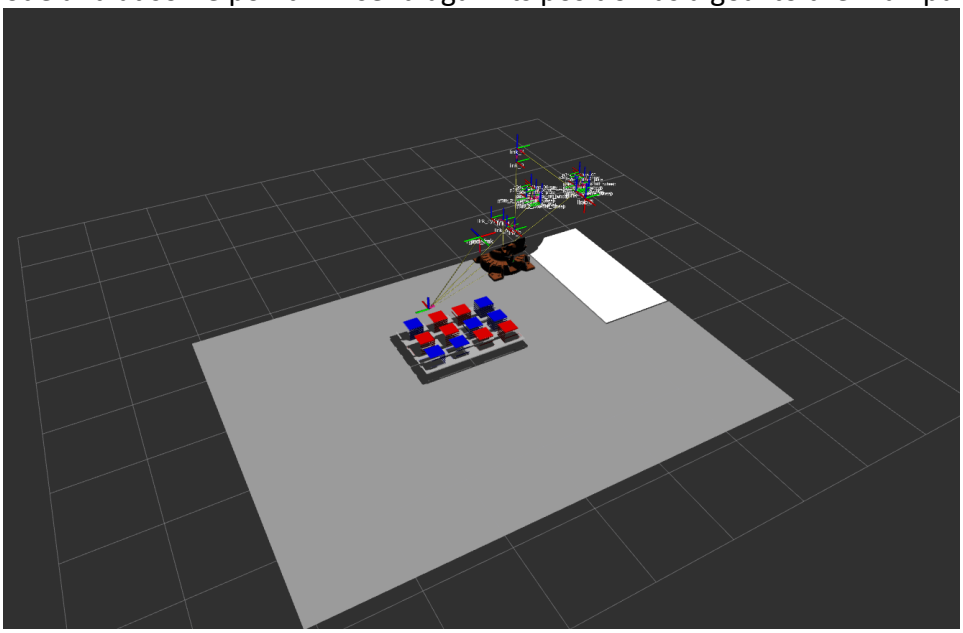


Figure 5 example of point cloud derived from camera sensor

An easy solution to this kind of problem could be just moving a little bit the camera and in this particular case it's sufficient to just rotate the camera around the z axis(the "z" we are talking about is the one with respect to the ground frame), as showed in Figure 6.

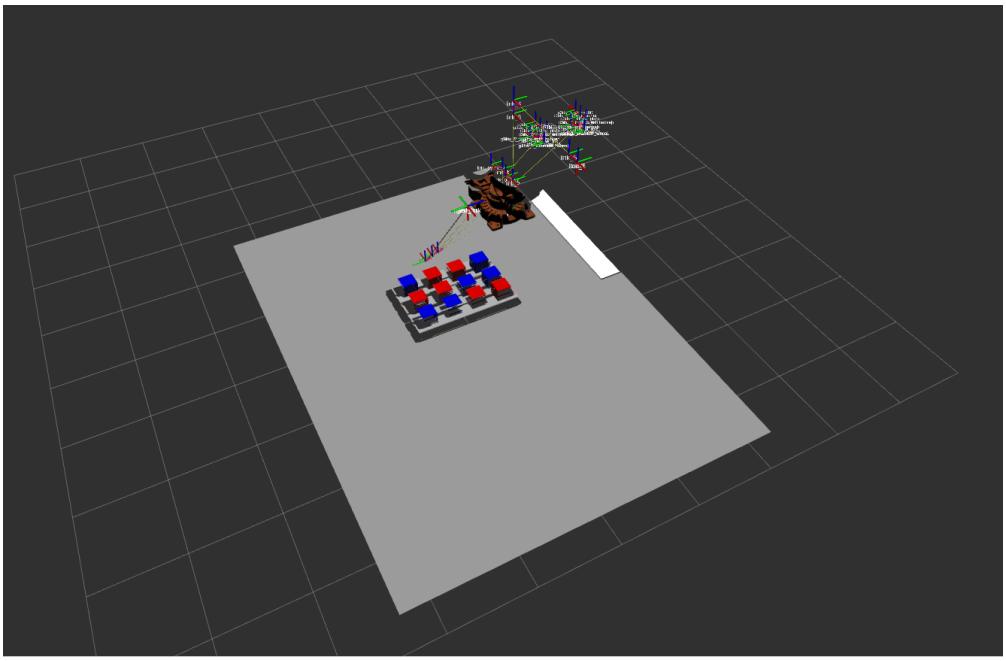


Figure 6 The new point cloud after the rotation.