

Faculty of Engineering  
MCG 5353 – Robotics  
Spring/Summer 2023  
Dr. Amirhossein Monjazeb, P.Eng

Design and Development of a Robotic Sorting System for A Production Line

|  |  |  |  |
| --- | --- | --- | --- |
| # | Group member’s full last and first name | UOIT student ID number | Signature |
| 1 | Mariam Hussien | 300389400 |  |
| 2 | Ibrahim Elshenhapy | 300389386 |  |
| 3 | Ahmed Asem | 300389894 |  |
| 4 | Ghada Salah | 300389364 |  |
| 5 | Shady Osama | 300389887 |  |

Contents

[Introduction 3](#_Toc141059404)

[Objective 3](#_Toc141059405)

[Requirements 3](#_Toc141059406)

[1- Modeling the Robot and Embedding Motors and Sensors: 4](#_Toc141059407)

[a. Create or find a URDF or xacro file for your desired robot model, ensuring that the dimensions in the simulator match the real robot. 4](#_Toc141059408)

[b. Attach motors to the joints and sensors to the links for the simulation of robot in Gazebo. 4](#_Toc141059409)

[2- Workspace Design: 6](#_Toc141059410)

[a. Design the workspace in Gazebo, including a conveyor and any other necessary components for the sorting station. 6](#_Toc141059411)

[b. Justify your design choices and the selection of components in your final report. 7](#_Toc141059412)

[In our final report, we justified the design choices and component selections for the conveyor system based on several factors that are essential to the success of our project. Here are the key justifications: 7](#_Toc141059413)

[ Availability of Gazebo Conveyor Plugin: The selection of the conveyor system with the Gazebo Conveyor Plugin was primarily driven by its availability and compatibility with our simulation environment. Gazebo is a widely used robotics simulation platform, and the existence of a conveyor plugin allowed us to easily integrate the conveyor system into our simulation. 7](#_Toc141059414)

[ Convenient Cube Sizes: The choice of cube sizes was based on their convenience to our robot arm's end effector size. By selecting cube sizes that match or are close to the dimensions of the robot's gripper, we can ensure that the robot can effectively pick up and place the cubes without any collision or gripping issues. 7](#_Toc141059415)

[ Boxes are clear Visibility for Item Placement: The decision to use boxes for the conveyor system was made because boxes provide a clear and well-defined space for placing items. The open-top design of the boxes allows us to easily position items inside them, making it convenient for the robot to handle and manipulate the objects. 7](#_Toc141059416)

[ Suitable Robot Workspace: The selection of the robot arm with its specific workspace was crucial to ensure that the robot could reach all the desired places on the conveyor system. By analyzing the robot's workspace and comparing it with the layout of the conveyor system, we ensured that the robot can efficiently handle the objects on the conveyor. 7](#_Toc141059417)

[3- Robot Control 8](#_Toc141059418)

[a. Implement ROS controllers to control the robot's joints. 8](#_Toc141059419)

[b. Save the PID parameters in a YAML file for future runs. 8](#_Toc141059420)

[c. Create launch files that run all nodes and parameters, enabling a complete project launch. 9](#_Toc141059421)

[4- Object Position Detection and Command Generation: 9](#_Toc141059422)

[a. Develop a node to obtain the positions of bricks from the Gazebo simulator. 9](#_Toc141059423)

[b. Generate commands for each controller to enable the robot to pick and place bricks in the simulation. 10](#_Toc141059424)

[5- Extra Credit 12](#_Toc141059425)

[a. Install a camera on the ceiling to capture top-down photos in the simulation. Implement image processing techniques to detect object positions and types instead of relying solely on available link positions published with Gazebo. 12](#_Toc141059426)

[Future Work 14](#_Toc141059427)

[Conclusion 14](#_Toc141059428)

[References 15](#_Toc141059429)

# Introduction

ROS (Robot Operating System) is an open-source framework for building robotic systems. It enables communication between software components, offers modularity, and supports inter-process communication and sensor integration. ROS organizes functionality into packages containing nodes that communicate through topics or services. Gazebo is a 3D physics-based simulator integrated with ROS, ideal for testing and developing robots in a virtual environment before real-world deployment.

In this project, our group will create a Robotic Sorting System for a big production line. The system will use a robotic arm in a computer simulator called Gazebo. We'll be dealing with different types of bricks with various shapes and colors that come on a conveyor belt. Our task is to make the robot identify at least three types of bricks, pick them up from the conveyor, and place them in specific areas for packing.

# Objective

* Create a Sorting Station: Make a robot-based sorting station in the Gazebo simulator for the production line.
* Detect Different Bricks: Teach the robot to recognize at least three types of bricks as they move on the conveyor.
* Pick Up Bricks Carefully: Program the robot to pick up the recognized bricks carefully from the conveyor.
* Place Bricks in Designated Areas: Make the robot put the picked bricks in their assigned places for packing.
* Test and Improve: Test the robot's sorting abilities and make any needed improvements to ensure it works well.

# Requirements

1. Modeling the Robot and Embedding Motors and Sensors: We'll design or find a suitable file (URDF or Xacro) for the robot with 6 or 7 degrees of freedom. Making sure the simulator's dimensions match the real robot is crucial. We'll also add motors and sensors to the robot's joints and links for accurate simulation in Gazebo.
2. Workspace Design: Creating a well-designed workspace in Gazebo with a conveyor and essential components is vital for the sorting station's success.
3. Robot Control with ROS: We'll use ROS controllers to control the robot's joints, ensuring precise and efficient movement.
4. Object Position Detection and Command Generation: To detect brick positions in Gazebo.

A computer screen shot of a computer screen

Description automatically generated

Figure 1 Our Workspace as starting.

## 1- Modeling the Robot and Embedding Motors and Sensors:

### Create or find a URDF or xacro file for your desired robot model, ensuring that the dimensions in the simulator match the real robot.

We found a robot online that was designed and transformed into a URDF file and started working from there. The robot is shown in Figure (2).

A white robotic arm on a grey tile floor

Description automatically generated

Figure 2 Robot arm

### Attach motors to the joints and sensors to the links for the simulation of robot in Gazebo.

First, we edited the package.xml and cmake files of the robot URDF to add the dependencies for the robot to work correctly.

A screen shot of a computer code

Description automatically generated

Figure 3 Attach Motor to joint 1 (robot\_arm\_urdf.urdf)

A diagram of a program

Description automatically generated

Figure 4 Robot links in a time instant.

## 2- Workspace Design:

### Design the workspace in Gazebo, including a conveyor and any other necessary components for the sorting station.

A screen shot of a computer code

Description automatically generatedThe simulation is on Gazebo where I can control the virtual environment of the robot. We start by creating a new empty world. Every empty world has orthogonal coordinates (x, y, z) where we can refer to every object in that environment to its origin point. In the empty world some attributes such as a light source and gravity are implemented to mimic the physical world. We used the gazebo ROS library as a reference for the empty world used.

Figure 5 Launching Gazebo empty world (full\_robot\_arm\_sim.launch)

A screen shot of a computer code

Description automatically generatedThen, in this world we launch each object using different ways such as urdf files. The objected launched is in reference to the world origin point. We can control the position and the orientation using 6 factors: x , y , z , raw , pitch , yaw. We launched a gazebo conveyor plugin that helped us control the conveyor and spawn cubes on it. The file uses a reference urdf files that represent the conveyor, red cube , green cube , and blue cube. The demo.py file in the demo world package.

Figure 6 Spawning function initialization (demo.py)

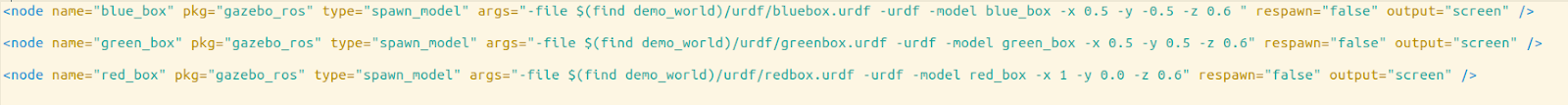
Then we added the boxes of the environment where each box is used for a certain cube color.

Figure 7 Launching the sorting boxes. (full\_robot\_arm\_sim.launch)

Launching The conveyor with random generated boxes with (blue, red and green) color in Gazebo world to be as shown in the next figure. We already have the 3 boxes with the conveyor and blocks as shown in figure (1) and will be later shown as the system working.

A white table with blue objects on it

Description automatically generated

Figure 8 Moving conveyor with randomly spawned cubes.

### Justify your design choices and the selection of components in your final report.

## In our final report, we justified the design choices and component selections for the conveyor system based on several factors that are essential to the success of our project. Here are the key justifications:

## Availability of Gazebo Conveyor Plugin: The selection of the conveyor system with the Gazebo Conveyor Plugin was primarily driven by its availability and compatibility with our simulation environment. Gazebo is a widely used robotics simulation platform, and the existence of a conveyor plugin allowed us to easily integrate the conveyor system into our simulation.

## Convenient Cube Sizes: The choice of cube sizes was based on their convenience to our robot arm's end effector size. By selecting cube sizes that match or are close to the dimensions of the robot's gripper, we can ensure that the robot can effectively pick up and place the cubes without any collision or gripping issues.

## Boxes are clear Visibility for Item Placement: The decision to use boxes for the conveyor system was made because boxes provide a clear and well-defined space for placing items. The open-top design of the boxes allows us to easily position items inside them, making it convenient for the robot to handle and manipulate the objects.

## Suitable Robot Workspace: The selection of the robot arm with its specific workspace was crucial to ensure that the robot could reach all the desired places on the conveyor system. By analyzing the robot's workspace and comparing it with the layout of the conveyor system, we ensured that the robot can efficiently handle the objects on the conveyor.

## 3- Robot Control

### Implement ROS controllers to control the robot's joints.

We added the required controllers as you can find in the next figure.

A screen shot of a computer program

Description automatically generated

Figure 9 Ros Controllers for robot joints (ros\_controllers.yaml)

### Save the PID parameters in a YAML file for future runs.

Our Pid is done for all joints you can find part from the pid gains for the robot arm in the next figure.

A screen shot of a computer program

Description automatically generated

Figure 10 PID gains for the joints 2,3 (ros\_controllers.yaml)

### Create launch files that run all nodes and parameters, enabling a complete project launch.

We collect all in one launch file for the arm, the conveyor, cubes, collecting boxes and the camera.

A screen shot of a computer program

Description automatically generated

Figure 11 Launch file for whole project (full\_robot\_arm\_sim.launch)

## 4- Object Position Detection and Command Generation:

### Develop a node to obtain the positions of bricks from the Gazebo simulator.

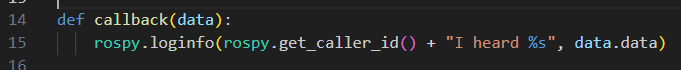
We publish the position of each brick as the cube on the conveyor.

A screen shot of a computer program

Description automatically generated

Figure 12 Publish cubes positions (demo.py)

We now will subscribe to the cube position so that we later give the robot arm it's location to be able to pick.



A screen shot of a computer screen

Description automatically generated

Figure 13 Subscribe the cube position (move\_to\_point.py)

### Generate commands for each controller to enable the robot to pick and place bricks in the simulation.

First, we define arm group and gripper group

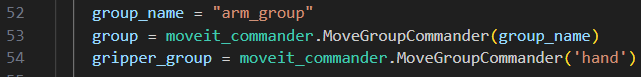


Figure 14 Define arm and gripper group (move\_to\_point.py)

Now we go to the initial position.

A screen shot of a computer program

Description automatically generated

Figure 15 Intial position for the robot (move\_to\_point.py)

Using this function, we go to the goal position which obtained from the subscribe node.

A computer screen shot of a program

Description automatically generated

In the init function of the class we use this function to get to the goal

A screen shot of a computer program

Description automatically generated

As the robot arm receives the cube position it starts to move to pick it then place it in the box where the camera used for cube color detection to have the decision of with box to place the cube in the right box.

A machine with different colored boxes

Description automatically generated

Figure 16 Robot after placing some cubes and missing some.

## 5- Extra Credit

### Install a camera on the ceiling to capture top-down photos in the simulation. Implement image processing techniques to detect object positions and types instead of relying solely on available link positions published with Gazebo.

We installed the camera into the system. The camera represents a .sdf file that has a box as a camera sensor with the RGB format. The sdf file has a plugin that represents the camera parameters and the information about the topic the images is published to it.

A screen shot of a computer program

Description automatically generated

Figure 17 Launching the sorting boxes (mycam.sdf)

Then we added the image subscriber node that subscribes on the /camera\_model/camera/image\_topic in order to access the recent camera RGB image captured by the camera sensor. This enabled us to have a live capture of the top view image through the camera.

A screen shot of a computer code

Description automatically generated

Figure 18 Image Callback (cam\_subscriber.py)

To conclude we used those items as an environment for our station each one for a reason:

* Empty world: to have a space with an orthogonal representation.
* Conveyor: to have objects to move on, to keep the process running and automated.
* Cubes different colors: the targeted objects to be sorted.
* Robotic arm to be able to reach out for the cube in a proper orientation and move pick the object and place it in the right spot.
* Boxes: to sort the cubes in.
* Camera: to sort the cubes by color using open cv and trigger a callback once the cube is in the right spot.

Then we added a mask for all the ranges of boxes colors we might need. We converted the RGB into HSV as it represents a better color map when it comes to classification. After that, we added masks for (green and lemon green), (blue and light blue), and red. Then we published the results on a topic to use it to give the action to the robot arm to start the pick and place for that specific scenario.

A screenshot of a computer program

Description automatically generated

Figure 19 Masking the colors (Classifier.py)

As previously explained, the camera sdf model has an installed plugin that captures gazebo environment in RGB formal and publishes it as an image type message on the designed topic: /camera\_model/camera/image\_topic. Then, we subscribed to this node to both view the image as a live view and to classify the image of the cube color.

A diagram of a computer

Description automatically generated

Figure 20 Color detection node

As seen here, the camera can see the green cube at the starting of the conveyor from the top view which is shown in the camera image window. We can run this file by running the cam\_subscriber.py in the Moveit\_robot\_arm\_sim package.

# Future Work

* Hardware Selection: Choose an appropriate robotic arm with the required degrees of freedom and payload capacity for real-world application. Consider factors such as cost, availability, and compatibility with the system.
* Sensor Integration: Integrate real-world sensors (e.g., cameras, depth sensors) on the physical robot to detect and identify the different types of bricks on the conveyor.
* Real-Time Control: Develop real-time control algorithms and interfaces to ensure the robot can respond quickly and accurately to the changing environment of the production line.

# Conclusion

In this group project, we designed and developed a robotic sorting system using the Gazebo simulator. The system can identify different types of bricks on the conveyor and place them in designated areas for packing. For future work, we could explore transitioning to real hardware integration, including selecting an appropriate robotic arm, integrating sensors. The project provided valuable practical experience in robotics, control systems, and simulation, fostering collaboration and problem-solving skills. It highlighted the potential of robotics in modern manufacturing and contributed to our understanding of automation applications.

A diagram of a diagram

Description automatically generated

Figure 21 Moveit topic

A diagram of a diagram

Description automatically generated

Figure 22 rqt graph

# References

1. ROS Wiki: Documentation (<https://wiki.ros.org/Documentation>)
2. Gazebo Software and Documents (https://classic.gazebosim.org/tutorials)