# Modeling the Environment and Robots of an Amazon Warehouse

Elif Erbil
*Department of Informatics*
*Technical University of Munich*
Munich, Germany
elif.erbil@tum.de

Eesha Kumar
*Department of Informatics*
*Technical University of Munich*
Munich, Germany
eesha.kumar@tum.de

***Abstract*—With increasing focus and continuous efforts to integrate robots into daily life, there is a growing requirement to quick prototyping and building of test or simulation environments to evaluate robot presence in virtual worlds prior to deployment onto the real world. Building virtual worlds to model real worlds has multiple advantages, some of them being - the interaction closely mimics one of the real world, thereby providing a close-to-reality experience. Another reason would be the cost of building virtual worlds are less expensive both by monetary value as well as safety, a mistake in the virtual world can be replayed or undone and, even provide data to analyse collisions. An industrial setting, such as manufacturing or warehouse logistics is an excellent avenue to model, observe and evaluate operations. In accordance, this paper discusses the modeling and functionality of a robot simulation in a warehouse environment similar to Amazon's warehouses. Subsequently, we propose virtual warehouse models to deploy robots and storage amongst humans, all in a simulation. A simulation is practical way to test functionality with external applications, presumably clients. In our warehouse, robots aid in increasing shipping efficiency, by automating the movement of items between locations. The warehouse consists of multiple robots and storage units to store items from various orders. The warehouse is a semi-autonomous setting, with assumed human presence to account for uncertainty in real-world environments.**

***Index Terms*—gazebo, ros, simulation, kiva, robot, warehouse, software engineering, robotics**

## I. Introduction

Currently, robots are being used in many different areas to facilitate certain tasks which may require high effort from people or which can be automated to increase efficiency of a system. However, running experiments for such cases with physical robots can be both an expensive and a time consuming process, especially without the required hardware knowledge to build physical robots. In such cases, it is more efficient and less effort to use a simulation environment such as Gazebo [1] to build virtual robots and test them in real world conditions. This paper provides a simulation model for one the use cases of such robots, which is the automated warehouse model of Amazon with the Kiva robots that facilitate carriage of items to be shipped in an efficient manner.

Multiple Kiva robots that can carry storage units simultaneously and therefore the time to retrieve the items and to bring them to the shipping units where they are sent outside the warehouse is significantly reduced. This paper gives an insight into using the simulation model for warehouses with the necessary functionality to observe the warehouse functionality, limitations and run experiments as necessary. The remainder of the paper is organized as following: Section 2 discusses the related research and background information on warehouse models and the softwares Gazebo and ROS which are needed to create our model, Section 3 discusses the software architecture of the virtual warehouse as well as our implementation strategies. The evaluation of the simulation model with the current outcome is explained in Section 4 and, future work and concluding remarks will be given in Section 5 and 6.

## II. Related Works/Background

Since we draw inspiration for this simulation from the Amazon Models of Warehouses and Warehouse Management, this section will focus on building this idea to apply for a warehouse simulation. An Amazon Warehouse traditionally was built to automate storage and retrieval process of various items shipped to/from the warehouse. An in depth study of such a construct reveals atomic and easy-to-grasp concepts and objects. These are namely robots, shelves, items and orders, a conveyor belt and the storage unit themselves. These components in various scenarios can be analysed to generate requirements for our project. For e.g. a common use-case can be gathered as (at a very low-level stage) Move Robot 'x' steps so in the grid layout, physical force or velocity to propel the robot forward will be applied in a certain direction. Intuitively, this is trivial. However, replicating such motion in the real world can be time-consuming and costly when compared to designing a simulation environment to first carry out such tasks. Thus software engineering principles and available software are highly relied to design the simulation components as well as functionalities.

Well known and recommended Robot Programming and Simulation softwares in use are Gazebo and ROS. In short Gazebo provides a simulation environment and ROS handles communication. A high level understanding of such a system is that Gazebo objects and components represent the real-world system. In order to communicate with this real-world system, we are required to establish a medium of interaction such as ROS. This following section introduces and describes

- Gazebo Simulation

- ROS robot communication and control
- ROS Gazebo Communication
- Robot Dynamics and Kinematics

in detail so as to provide a complete section of useful resources.

### *A. Gazebo Simulation*

Gazebo is a simulation environment software aimed at providing a fast and robust interface to test expensive real-world scenarios. In Gazebo, to fit our use case, we model the warehouse with storage units, shelves, robots and items. The constructs are put together with the SDF prescribed by Gazebo.

*1) SDF:* Simulation Description Format is a markup prescribed by Gazebo where a user can build models and connect functionality [2]. It is specifically designed to adhere to Gazebo's model based approach in running simulations. Intuitively, it can be looked at as "in order to visualise a warehouse scenario, we first provide models and then add functionalities to it".

### *B. ROS Robot Communication and Control*

ROS, or robot operating system is a robot communication interface which provides services, clients and PubSub models to engage with the robot [3]. ROS provides easy-to-use APIs along with its description and launch formats for initializing with Gazebo and controlling with ROS.

*1) URDF:* Uniform robot description format is the ROS specified format for describing the robot in the ROS paradigm. In our use case, URDF does not hold equal importance to SDF since the simulation is paramount to the topic.

*2) Launch:* ROS allows launch files(.launch extensions) to launch worlds in Gazebo with some prerequisites in the simulation. This is a handy tool since all simulations are built in Gazebo, however, the communication is administered and controlled via ROS.

*3) Catkin:* Catkin is the built system prescribed by ROS to build modular world functions and plugins [4]. Catkin resolve package dependencies and helps in maintaining the code written for experiments. Since ROS provides C++ as well as Python for experiment control, Catkin is useful for resolving the implicit build dependencies when the two languages are used in the simulation and experiments.

### *C. ROS Gazebo Communication*

Gazebo and ROS use topics for inter as well as intra communication. More so, in ROS, there are two primary channels of communication when the warehouse is launched. These are Asynchronous and Event based PubSub model as well as Synchronous Client/Service model.

*1) Publisher Subscriber Model:* ROS PubSub model is used to communicate robot commands over topics to Gazebo. It is also used to retrieve simulation information since some topics in Gazebo have an in-built ROS counterpart to relay information. This hinges on abstraction as a software principle thereby allowing the same functionality to be relayed to High Level Communication.

*2) Client-Service Model:* The ROS Client service model also uses topics but to provide certain specific functionality to control create/modify/delete parameters of the simulation.

ROS and Gazebo communication is facilitated with a pub/sub model for publishing commands to move robot as well as rotate robot. ROS and Gazebo have implicit command and information relaying mechanisms because of which it is easy to publish from ROS to control robots and the simulation in Gazebo. This also would provide the necessary abstraction for when the API would like to communicate information to the simulation.

### *D. Robot Dynamics and Kinematics*

A robot performs translation and rotational functions based on certain velocities and forces applied on it. These usually indicate applying velocities on the wheels for translation motion and angular velocities on the entire body to impart torques in order to rotate the robot. Kinematics is the physics behind applying appropriate velocities and angular velocities while dynamics is analysis and expression of forces and/or torques. In our case, Gazebo implicitly applies the dynamics, considering the tedious and iterate process to calculate joint torques(forward kinematics) as well as determining pos while given joint velocities(inverse kinematics). This makes implementation easier by simplifying equations.

## III. Methodology

For the automated warehouse system the following use cases are created to follow how the user can interact with the system. The expected functionalities in the warehouse are the following:

- Spawning Kiva robots,
- Spawning storage units with items,
- Kiva robots moving in the x and y axis,
- Kiva robots rotating along z axis to align to x and y axis,
- Kiva robots lifting storage units, carrying them and bringing to the storage units to the drop points near conveyor belts.

The user is allowed to use the warehouse model in two different levels: the high level user can command the selected Kiva robot to move towards a given coordinate point or a user who prefers to access to lower level functionalities (for example to train a model for path planning using Kiva robots) can command the Kiva robot to moves in x or y axis with the given amount of distance and rotate the robot clockwise and counter-clockwise for 90 degrees. A user can command the robot to lift or drop a shelf. In order to create the simulation, the implementation work is divided in 3 parts: creating the software architecture of the warehouse, creating the robots and warehouse items in Gazebo and starting and controlling the Gazebo simulation using ROS.

### *A. Software Architecture*

*1) Warehouse Model:* In order to create the simulation, the warehouse was modeled as an object oriented structure where a warehouse has multiple Kiva robots, storage units

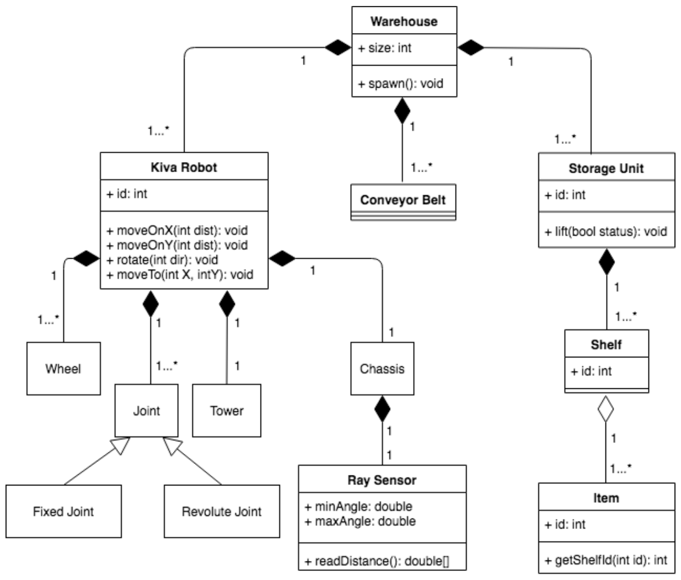


Fig. 1. UML class diagram of the warehouse model

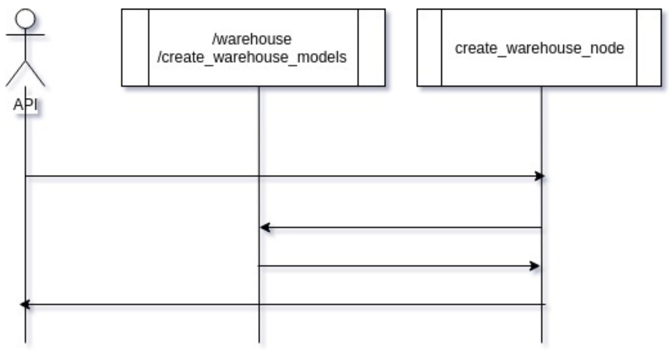


Fig. 2. Sequence Diagram of Create Warehouse

and conveyor belts. Each Kiva robot consists of a chassis, wheels, and a tower to lift the storage units. Each storage unit consists of shelves and items that are stored in shelves. Conveyor belts are simple objects as the user cannot interact with them and they act as a marker for the drop point of storage units that contain shipped items in the simulation. The warehouse model is shown as a UML Class diagram with the necessary functionality in Figure 1.

As such, the warehouse simulation at the start is responsible for the creation and instantiation of the simulation as shown in the sequence diagram in Figure 2. The representation expresses a simple call to start a simulation with some number of robots, items and storage units.

In order to create a simplified simulation, certain assumptions and limitations are made on the warehouse model for ease of use. Firstly, the warehouses are modeled as grids so that the Kiva robots can only move in between each point inside the grid where the storage units can be placed. So the movement of Kiva robot is limited to the Manhattan distances and rotations are limited to multiples of 90 degrees and the robots can be moved in a holonomic fashion. Secondly, Kiva robots can move beneath the storage units to move but they are blocked by other Kiva robots or a conveyor belt. Thirdly, the initial warehouses are created with default sizes for the user to choose, which spawns a predefined number of storage units and Kiva robots. These commands can be published to the warehouse upon launch so as to provide external control with minimum launch specifications.

*2) Kiva Robots:* After creating the object relations, it is also possible to create a flow for the functionality of the Kiva robots as they are the main actors of the simulation which the user can interact with. The Kiva robots have three functionalities to follow which calls for basic functions to run a successful simulation. Kiva robots can be moved in x and y-axis by the user or can be moved towards given coordinates. In both cases, the Kiva robots follow a certain flow to orient themselves in the warehouse and move the required amount. When the Kiva robot gets a movement command, it first checks if it needs to move by checking the displacement required and if there is a movement needed, Kiva starts its movement by orienting itself. Kiva orients itself by comparing the initial orientation with the final direction to move by using the yaw information. Then, it calculates the distance it needs to move to reach to its final destination and starts the movement. Throughout its movement, it checks the distance travelled and stops when it reaches its destination. For the movement towards an end point, the robot goes through the given steps two times, one for the x coordinate and one for y coordinate that is specified by the user. This flow is visualized in the UML activity diagram in Figure 3.

A further in depth look at sequence of operations gives a better view of move and rotate operations executed in a sequential manner on the Kiva robot. Since we expose even the low-level operations of the robot, namely move on x, move on y and rotate as well as a high-level move on x and y, it is possible to see the flow along with the published content. A sequential diagram for these set of operations can be found in Figure 4.

### B. Gazebo

In order to create and visualize the simulation Gazebo is used. Subsequently to create the warehouse and its objects, SDF is used. Kiva robots, shelfs and storage units are created as models with collisions to create a realistic simulation environment. Figure 5 shows the models that are created in Gazebo. After creating the model files, the world file is created to initialize the locations of the Kiva robots, conveyor belts and storage units. After creating the models and the world, the simulation environment is ready and the functionality can be implemented.

*1) Model and World Plugins:* Movement and rotation functionality of Kiva robots are implemented as a model plugin whereas the lifting functionality of the storage units are controlled through the world since the lifting functionality is not implemented as an actual lifting movement but implemented in a way that the storage unit that is carried by a Kiva robot

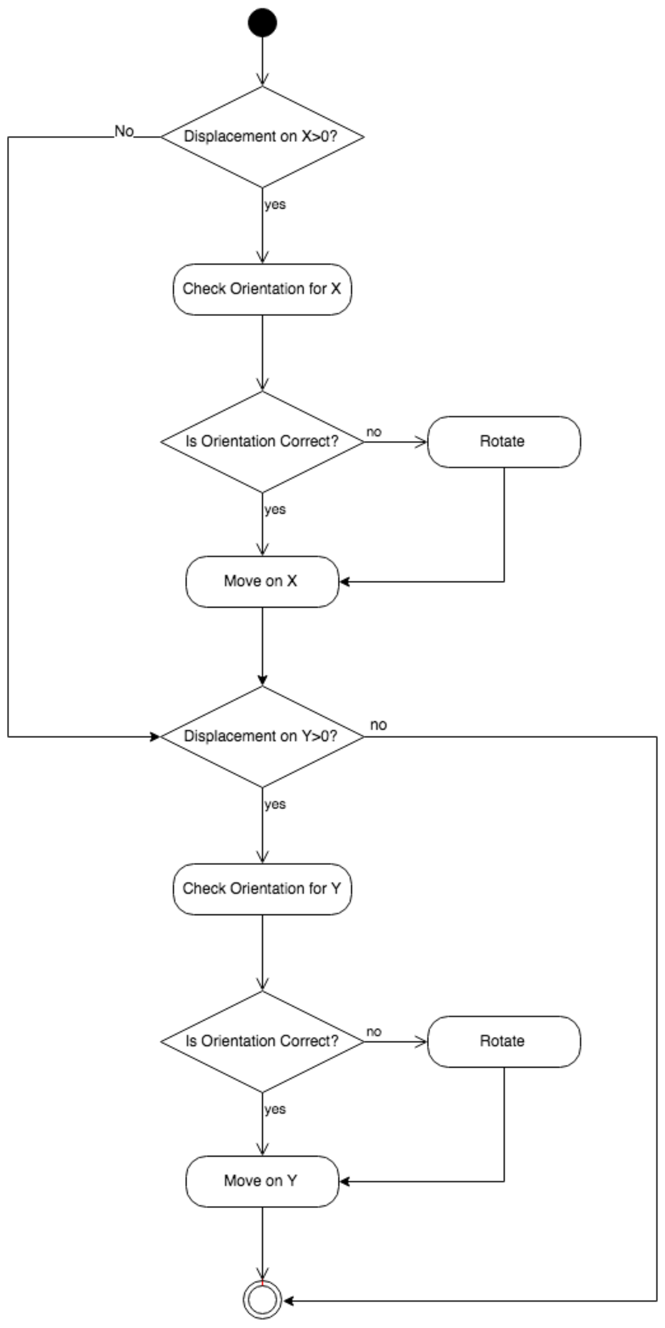


Fig. 3. UML activity diagram of the kiva robot movement

follows the robot when it is in a certain proximity to be lifted by changing the world pose of the storage unit to the world pose of the Kiva robot. The world poses allows the lifting movement since it is possible to observe the location of both the robots and the storage units. Kiva robots and shelf ids are mapped to allow the program to achieve simultaneous movement of storage units.

The Kiva plugin is implemented by using state transitions. Each Kiva robot can have 5 different states: idle, moving on X, moving on Y, rotating and blocked (which are sub-states of the state busy). These states allow Kiva robots to accomplish

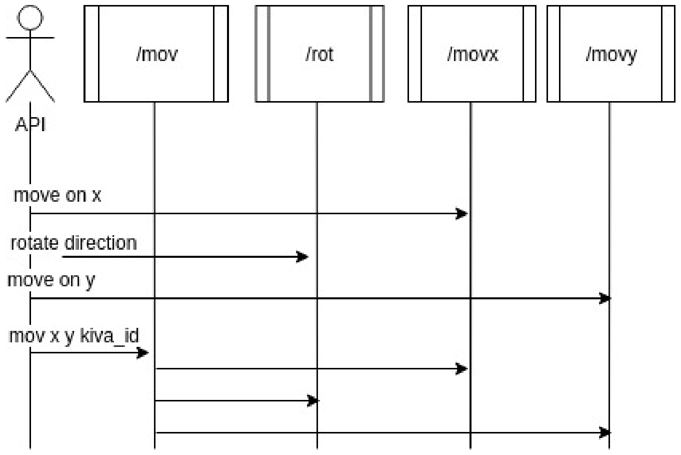


Fig. 4. Sequential Diagram for move and rotate operations

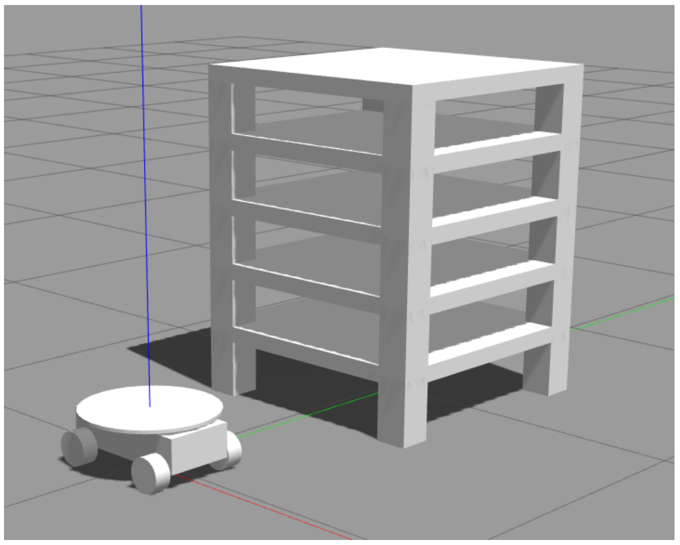


Fig. 5. Gazebo Models of Kiva Robots and Storage Units

higher level tasks such as moving between two coordinates, which might require multiple movement and rotations. Each Kiva robot uses state flags to achieve certain functionality since different commands requires different code pieces to be run and since different Kiva robots can be moved simultaneously.

In order to move the Kiva robots in the Gazebo simulation, the velocities of different joints are changed to create the movement. For the movement in x and y axes, the velocity is set instantaneously on the wheel joints since this movement is relatively simple movement and it allows to keep the velocity constant throughout the movement which makes the distance calculation towards the end point more accurate [5]. When the Kiva robot reaches its destination or is blocked, the velocity of the joints are set back to 0. This also allows to instantaneously stop the robot and keep them in the grid points.

For the rotation movement, the angular velocity is changed and the yaw value of the robot is used to compare with the final yaw to observe the rotation movement. When a rotation command is given, the start rotate function is called to calculate the final yaw value by using the direction given and the current yaw value of the Kiva robot and then starts the rotation by setting the angular velocity to an initial value towards the given direction. Since the angular velocity behaves differently than the linear velocity, after the initial change in the angular velocity, the velocity is decreased to make sure the rotation is kept in the desired range. Then for the rest of the rotation, if the current yaw is less than 0.1 degrees from the final yaw, rotation continues counterclockwise and if the difference is greater than 0.1 degrees, rotation continues clockwise. This allows the Kiva robot to check if it has reached its final yaw value or if it has passed the necessary way, it rotates towards the opposite direction to reach to its desired value. However, since the yaw value is calculated in radians, the rotation values cannot be calculated this way since 180 degrees have both the values -1.54 and 1.54. In order to solve this problem, the yaw values are calculated to degrees and to eliminate the problem of rotation from 0 degrees, the final yaw values are changed according to the movement direction to keep the robot in its necessary range and still have the correct rotation. Since the yaw value doesnt have an exact value, to keep the value close to the end value and also to successfully end the rotation, each rotation movement has a maximum number of iterations that is allowed per rotation. The rotation ends after the maximum rotations are reached. This allows Kiva robot to continue its rotation to have a more accurate result by rotating in both directions until the value is within the desired range of yaw values ($\pm$ 0.01 degrees). However, the rotation is still completed even if the exact value is not reached to have a more fluent simulation. Number of iterations are changed according to the necessary number of rotations since the orientation function might need two 90 degree rotations which will be done consecutively.

*2) Publisher Subscriber Model:* In order to control the Kiva functionality, a publisher-subscriber system is used. Kiva robots subscribe to different publishers to fulfill different commands. Movement functionality is controlled by 4 different publishers:

- ./mov x y kivaid - allows to move Kiva robot with given id to move to the given coordinates
- ./movx dist kivaid - allows to move Kiva robot with given id on x-axis for the given distance
- ./movy dist kivaid - allows to move Kiva robot with given id on y-axis for the given distance
- ./rot dir kivaid - allows to rotate Kiva robot with given id to rotate 90 degrees clockwise or counter clockwise

All publishers can take new commands at any given time; however, the command will only be executed if the status of the selected Kiva robot is idle. Another publisher is created for lifting the storage units and to map which storage unit is carried by which Kiva robot:

- ./lift status kivaid shelfid - allows Kiva robot with given id to carry or drop the shelf with given id

The lift method can again get commands at any given time; however, it can only be executed if the robot is in close proximity of the shelf and the shelf is not mapped to another Kiva robot already. Status is set as a flag where 1 means that the shelf will be carried whereas 0 means that it will be dropped.

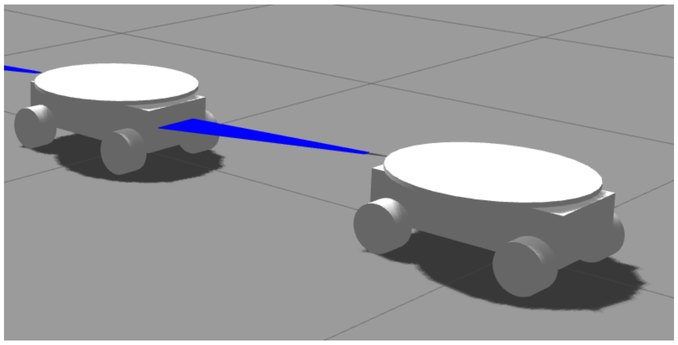


Fig. 6. Kiva Robots with Ray Sensors

Lastly, a publisher is created to share the status of each Kiva robot to the user:

- ./"kiva"+kivaid/status - returns the current status of the Kiva robot

The status of the Kiva robot is returned to either idle, busy or blocked, so that it is possible to observe which Kiva robots are idle and which are working, which may be beneficial for the users to optimize their queries.

*3) Sensors:* In order to prevent collisions, each Kiva robot has a laser sensor at the front to check if they are being blocked [6]. Ray sensors scan a given range periodically to calculate its distance with other objects and to check if there will be a collision. The range of the sensors are selected to cover the angle that can check if the Kiva is blocked by another Kiva or a conveyor belt, but not by a storage unit. The sensors are added to the Kiva models in the sdf file and controlled by the built-in sensor plugin for ray sensors where the scanning period and the angle is defined. Since each Kiva robot can only move by unit distances, each robot must stop moving if there is another Kiva robot in the next grid point it will be moving to. The ray sensors publish their measurement information through the following publisher:

- /kiva+id/chassis/laser/scan

since the sensor is a part of the chassis and the model plugin of the Kiva robots subscribe to this information to check the distance.

The model plugin checks the distance information of each ray in the given angle range and compares it to the defined limit and even if one ray is blocked, the blocked flag of that robot is set true, it ends the movement command, changes the status of the Kiva robot to blocked and set it back to idle. This allows the user to know that Kiva hasn't completed its movement due to an obstacle and is ready for a new command.

## *C. ROS*

ROS, as previously introduced, helps facilitate and relay commands to the objects and models in our warehouse. In our scenario, the warehouse is a Gazebo simulation. ROS is used to help API communicate with the simulation for control experiments. In this case, our use cases from API can be along with lines of initializing the Warehouse (empty/ predefined size), fetching objects (locate robot + relay command to move + receive status), etc. This sub-section describes standalone and atomic programs that could be of use for the API to exert such control.

*1) Launch and Plugins:* ROS provides launch files as a means of triggering a simulation launch in Gazebo. Warehouse world written as a Gazebo simulation is launched using the include tag in URDF markup. The launch is modeled in a way that a user can launch an empty world and then insert a Gazebo world file to run their simulation. Thus, a ROS launch is performed to start a simulation. A ros launch can be used by the front end as well as the API when a user makes a request from the web page.

*2) ROS Nodes:* ROS Nodes are atomic programs that are run as part of a launch. The run independently to modify and update individual components of the simulation. Since our simulation has control over creation, updation and tracking of objects, we make use of ROS nodes to execute these independent tasks.

- create_warehouse_service_node - This node manages the creation of warehouse models like robots, items and shelves.
- create_warehouse_topics - This node initializes topics required to facilitate communication in the warehouse.
- track_items_node - This node shows tracking example of items in warehouse
- track_storage_units_node - This node shows tracking example for storage units in the warehouse

The nodes are created to ensure that certain functionality of the warehouse is accomplished. The functions as part of the nodes can be leveraged by the API or digital assistant in order to control the warehouse simulation.

*3) ROS Messages:* ROS Topics and Services allow creation of custom messages to control modelling of ROS Programs and commands. ROS Messages are very useful when we require a specific group of data types and replace having to create classes or models instead. With Catkin, it is possible to define a message and its type so catkin can build the appropriate message model. Our simulation uses ROS messages for mainly two purposes. When a warehouse is launched, an API/digital assistant is allowed to provide the dimensions to the warehouse as well as components by quantity. The simulation also uses messages to track objects in the warehouse in order to exert and keep simulation control in the hands of the consumer/llient. The following are the ROS Messages with content types written in our application to give an idea of the type of information required to be shared by clients.

- Create Warehouse - Typically a request/response model where the request accepts a Header (with meta data and timestamp), warehouse name and id, number of robots, items and shelves for initialization. The response model returns Header and a Boolean set to true if request served successfully.
- Track Objects - Items and shelves need to be tracked in the warehouse to assign pickup and drop duties to the robot. In order to retrieve this information regarding state of the warehouse, it is constantly communicated for consumption.

*4) Publisher Subscriber Model:* ROS allows publisher subscriber communication model where our programs and plugins launched can listen on a particular topic. The Kiva robots as well as warehouse world plugins have PubSub support to subscribe to listen for published commands respectively. The simulation programs use both in built as well as custom made ROS Topics to publish and subscribe to;

- Publishers
  - /items/item_id/pose - To track information regarding particular item.
  - /storage_units/storage_unit_id/pose - To track storage unit in warehouse.
- Subscribers
  - /model_states - To obtain information about the state of the model in the Simulation.

*5) Client Server Model:* ROS allows client server request response communication which our warehouse simulation leverages to perform certain tasks. The Create Warehouse is one such service which takes a client call with the parameters mentioned in ROS Messages to instantiate the warehouse. We also use an existing ROS Service to get a particular model state at a certain point in time in the simulation. The following are the formal definitions of the Services and Clients.

- Service
  - /warehouse/create_warehouse_models - Instantiate the models in a warehouse.
  - /get_model_state - Obtain a specific model state at a point in simulation time.
- Client
  - /warehouse/create_warehouse_models - Input for warehouse models' instantiation.

## D. ROS Gazebo Communication

Upon looking at ROS and Gazebo at individual components, it is time to now consider the requirements of the application with respect to communication. As previously seen, ROS and Gazebo work well together and provide a useful set of functionalities, such as launching Gazebo worlds with ROS. This section discusses the implicit and explicit communication between ROS and Gazebo for control in simulation environment via ROS.

Implicit communication between ROS and Gazebo is when there are in-build topics for communication between ROS and Gazebo for common functionalities so the API does not have to build from scratch. Implicit communication is particularly useful when handing control to external consumers so they are not bothered with inner implementation.

For example, implementation of instantiating and creating warehouse models calls upon spawn_sdf_model which in turn calls upon /gazebo/SpawnModel. Similarly, /get_model_state invokes /gazebo/GetModelState to track robot, items and storage units.

## IV. Evaluation

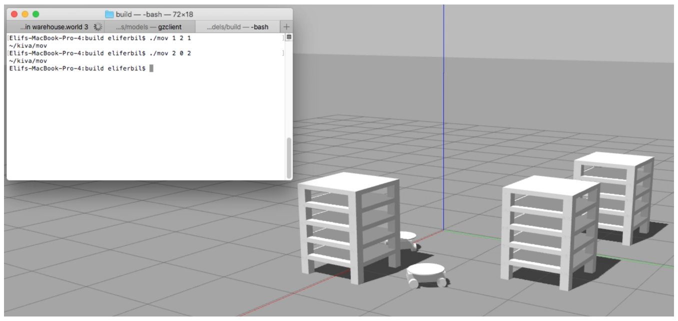
To evaluate the success of the warehouse simulation, the following functionality must be available users to create a stable environment which can be beneficial for users who intend to run experiments on the warehouse systems:

- One or more Kiva robots, storage units, items and conveyor belts can be spawned by the user to start the simulation,
- Kiva robots should follow the commands that are discussed in the earlier sections,
- Multiple Kiva robots should be able to follow the commands simultaneously and behave according to their status,
- Users should be able to monitor the state of the warehouse as well as the Kiva robots to observe the scenario in the warehouse without running the graphical user interface.

For a suitable warehouse model for the user, certain predefined models are created and the user can select among these warehouses accordingly. A user who might want to work on robot motion planning might need fewer robots to use the computational power more efficiently whereas a user trying to optimize the warehouse layout for faster deliveries might need more robots and storage units. For different purposes, the model offers different sized warehouses with one or more Kiva robots and these models are spawned using ROS.

In order to test the functionality of the Kiva robots and their behavior together, a simulation that covers different scenarios for the warehouse are created. Two different robots and multiple Kiva robots are created and they are simultaneously commanded to lift two different storage units to bring them to the conveyor belts. The simulation covered simultaneous movement, rotation, lifting and dropping of the robots as well as the scenario of collision since both robots are commanded to bring the storage units to the conveyor belt. The scenario showed that Kiva robots can receive commands at any time and behave independently and simultaneously.

Finally, to create a more user friendly simulation model and decrease the hardware need to run a simulation, the model must output the state of the simulation so that the user can observe the state of the warehouse without running a graphical user interface. To achieve the state of the warehouse model, the user must know the location of each object in the simulation, which can be tracked through the built-in world pose plugin, and also the state of the Kiva robots to construct further commands. This is achieved by the status publishers of each Kiva robot which shows if a Kiva is idle or if not what its status is. Thus the system presents the necessary functionality together with a mechanism to follow up the simulation in an efficient manner.



```
Elifs-MacBook-Pro-4:build eliferbil$ ./mov 1 2 1
~/kiva/mov
Elifs-MacBook-Pro-4:build eliferbil$ ./mov 2 0 2
~/kiva/mov
Elifs-MacBook-Pro-4:build eliferbil$
```

Fig. 7. Simulation Instance Where Multiple Kiva Robots Move Simultaneously

## V. Future Work

Having created a functioning simulation, we acknowledge there is room for improvement on two fronts. The first, along improving access and control to the simulation. This would cover improvements to communication via ROS to facilitate external communication and control to the simulation. The second, improving certain components within the simulation. These improvements are aligned towards upgrading certain components within the simulation.

### A. Improving access and control to simulation

In order to allow proper control of simulation and widen reach, it is necessary to connect our simulation with external clients in the project like the digital website as well as digital assistants. Currently, there exists no communication between the API to simulation and is crucial for launch, deploy and update operations in the simulation. Another limitation is use of limited/quickest algorithms for traversal in grid by kiva robots. Currently the kiva robots move x length along x axis and y length along y axis. While this works and is acceptable with minimum robots in the warehouse, traversal is complex with the advent of collision detection and needs path-planning utilities in-order to exert better control over the warehouse environment. A suggested solution is to use MoveIt [7], a motion-planning framework provided by ROS which is compatible with a Gazebo simulation. This would also enable the API to plan the motion with tracking information provided by our implementation in the simulation.

### B. Improving components within simulation

Although the simulation achieves the required functionality, there is room for improvement when it comes to expressing certain components of the warehouse. Firstly, the lifting functionality can me implemented as a separate module by using necessary sensors and actuators to create a more realistic simulation and observe the behavior of robots when objects are actually being carried. Also, the warehouse look can be improved with adding more objects such as walls, dividers, blockers, human models etc. This would improve the look and feel of the warehouse. Meshes can be used to cover the robot, storage units, items and shelves in order to express a better-looking warehouse. Meshes [8] are 3D representations provided by Gazebo to build realistic looks and overlay onto the original model in order to give a more finished look to the simulation. Improvements can also be made to define universal warehouse coordinators in order to provide better control to API while moving models in the simulation. A database connection can also be established if necessary to maintain an inventory of the warehouse as a look-up for stocks and items in a warehouse during simulation.

Finally, minor and miscellaneous improvements such as refactoring code for improved readability, better adaptation of software principles such as software testing can also be made to increase trust and reliability on the simulation. A futuristic improvement would be deployment on physical robots to create data and analyse performance.

## VI. Conclusion

A virtual warehouse was created with adequate functionality in regards to robots, storage units, shelves and items. The robots are able to traverse the warehouse and take bare minimum steps to detect and tackle collisions. While the robots do not communicate with each other, commands to move, rotate and lift are relayed to the robot for execution. In all, an efficient simulation is provided to navigate a robot in a semi-autonomous warehouse. The simulation, powered with Gazebo and ROS proved to be a cost-effective improvement prior to testing in the real world.

## VII. References

### References

[1] N Koenig, A. Howard, ”Design and use paradigms for gazebo and open-source multi-robot simulator”, IEEE/RSJ International Conference on Intelligent Robots and Systems Sendai, pp. 2149-2154, 2004, [online] Available: https://robotics.usc.edu/publications/media/uploads/pubs/394.pdf

[2] SDF Format, 2019, [online] Available: http://sdformat.org/

[3] About ROS, 2019, [online] Available: https://www.ros.org/about-ros/

[4] Catkin Conceptual Overview, 2019 [online] Available: http://wiki.ros.org/catkin/conceptual_overview

[5] Gazebo : Tutorial : Setting Velocity on Joints and Links, 2019, [online] Available: gazebosim.org/tutorials?tut=set_velocity&cat=

[6] Gazebo: RaySensor Class Reference, 2019, [online] Available: osrf-distributions.s3.amazonaws.com/gazebo/api/dev/classgazebo_1_1sensors_1_1RaySensor.html

[7] MoveIt: Motion Planning Framework, 2019, [online] Available: Moveit.ros.org

[8] Meshes: Gazebo Meshes - Improving Appearance, 2019, [online] Available: http://gazebosim.org/tutorials?tut=guided_i2