Grocery Robot

CS 225 Fall 2020 Project

Gabriela Bravo-Illanes Max Farr Jeffrey Zhang

I. Introduction

The "Grocery Robot" is a robot capable of navigating through a grocery store, completing a shopping list, and taking the items to the store's checkout aisle. This type of robot could be integrated into a delivery system, as a way to provide a contact-free grocery shopping experience.

We are living in unprecedented times. COVID-19 is a virus that spreads easily, and has killed more than 250,000 US citizens. Social distancing is the best solution while we don't have a vaccine, but there are certain activities that nowadays can't be avoided, including grocery shopping. Although it is possible to shop online, which avoids much of the interaction required, there must still be another person picking up the items, potentially spreading the disease or getting sick themselves. This is the type of context in which contact-free shopping technology becomes very useful.

The Grocery Robot consists of a Panda arm attached to a mobile base, with a 2-finger gripper. Its actions are controlled by two state machines: a higher level one that controls where the robot should go and what object to pick/place, and a lower level one that controls the pick and place actions. It makes use of 3 different controllers:

- *Navigation controller*: controls the position of the base of the robot while keeping the arm on at home position.
- *Arm control in world coordinates*: controls the whole robot. Designed to reach objects placed in the world
- *Arm control in base coordinates*: Moves only the robot arm while keeping the same position in the base. Designed to interact with objects static with respect to the robot frame, in our case the basket.

Our environment consists primarily of four clusters of shelves, each populated with one of three types of products: cartons of milk, jars of jam, and boxes of pasta. We also modeled a basket to transport the items, and a checkout conveyor belt on the opposite end of the room to place the basket on after shopping is complete.

II. Final implementation.

A. State Machine

The higher level behaviour of our system is described by the state machine shown in Fig. 1. The robot has a grocery shopping list: it will go to the shelf where an object is located, pick up the object, and place it in the basket located on the robot. If there are more objects in the shopping list, the robot will move to the appropriate shelf and repeat the same operation. Once the robot has finished the entire list, it will move to the checkout and place the basket on the conveyor belt.

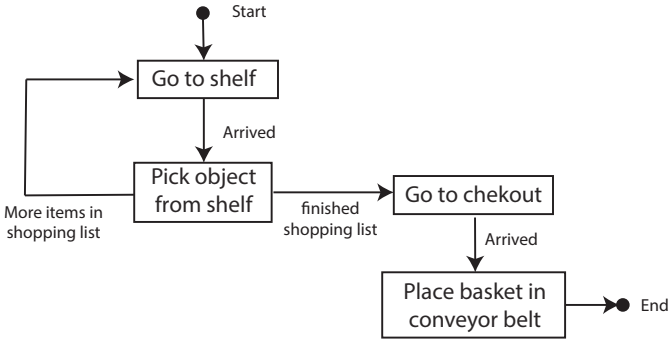

Fig. 1: High level state machine

To pick object from shelf and place basket in conveyor (Fig. 2), the robot uses a lower level state machine called "Pick and Place". In this state machine first the robot approach with the arm a position near the object outside the shelf, then it orients the end effector and start approaching slowly to the object. Once the end effector makes contact with the object (detected by a force sensor) the fingers will close and hold the object. Then the robot will lift the object a few centimeters, retreat the arm to get away from the shelf, and place the object in the basket.

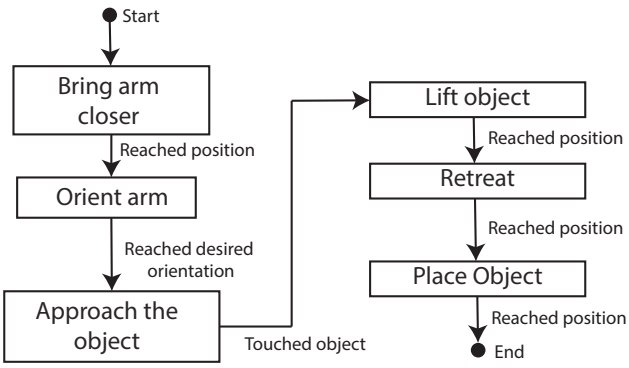

Fig. 2: High level state machine

To place the basket in the conveyor the steps are similar. First the robot place it self above the basket, orient the end

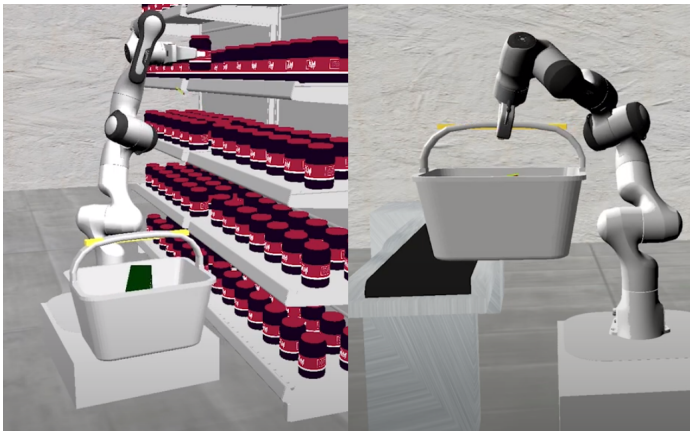

Fig. 3: Grocery Shopping robot. The robot consisted on a mobile platform and a panda arm with a 2 fingers gripper. It has a surface where the basket can be placed.

effector, approach the basket until contact with the handle. Lift the basket and place it on the conveyor belt.

B. Environment

We designed the aforementioned three products (milk, jam, pasta) for our robot to pick up, as shown in Fig. 4.

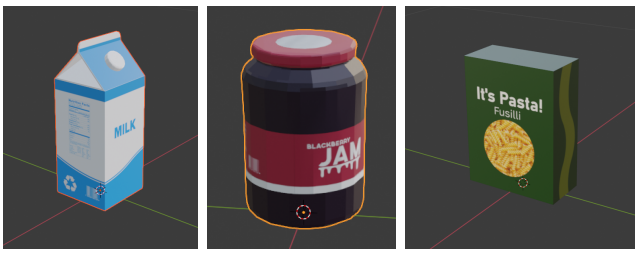

Fig. 4: Products

Our simulation takes place in a "grocery store" comprised of four walls, a floor, sixteen shelves, and a checkout conveyor belt. The shelves are arranged into clusters of four, as shown in Fig. 5.

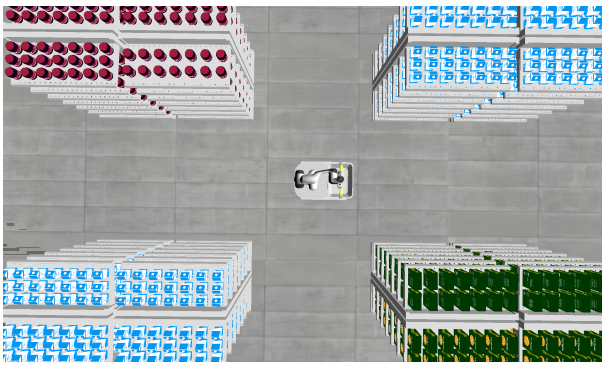

Fig. 5: Top-down view of the store.

Each shelf consists of an empty shelf model and one of two meshes to populate the shelf: either a mesh with all possible locations filled, or one with a single product removed, as shown in Fig. 6. These meshes are static, and the one with a product removed allows us to place a physics-simulated copy of the product in the empty space, which then allows us to display a densely populated shelf of products without having to simulate each and every object.

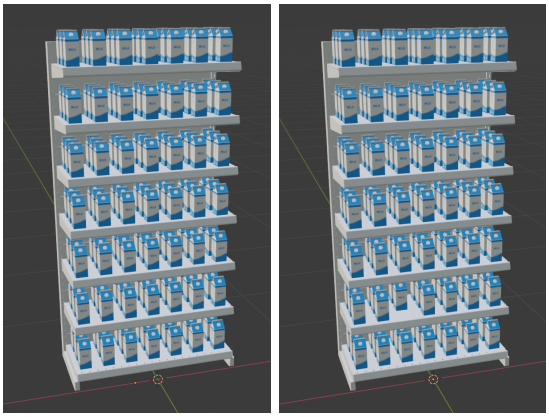

Fig. 6: Left: Full shelves. Right: Full shelf with a object removed on the second level.

As discussed further in the Challenges section, the basket's collision mesh was hard for us to import as an .obj file. In the final version of our project it is, in reality, four walls and a single rectangular bar "floating" where the handle is to approximate its shape. Similarly, the shelf's collision mesh is made up of a number of thin cubes which act as the different levels of each shelf, and add an extra level of difficulty in that we must pick up products without bumping adjacent levels of a given shelf.

C. Controllers

The grocery robot can be represented by 12 links. The first 3 are links without length that represent the 3 degrees of freedom (DOF) of the base $q_B = [q_0, q_1, q_2] = [x_B, y_B, \theta_B]$. The next 7 links the links of the panda robot $q_{arm} = [q_3, .., q_9]$. And the last 2 links the gripper links $q_{grip} = [q_{10}, q_{11}]$.

The total joint forces is the stack of the joint forces of the base F_b, the arm F_{arm} and gripper F_{grip}:

$$F = [F_b, F_{arm}, F_{grip}]^T$$

To compute this force, we have 3 different controller explained below [1]:

1) Navigation: This controller controls the position of the base of the robot $q_{B,d} = [x, y, \theta]$ while keeping the robot arm in "home" position.

$$q_d = \begin{bmatrix} q_{B,d} \\ q_{arm,home} \end{bmatrix}$$

[1]Controllers do not include gravity compensation since this was performed by the simulator

The torques of the joints are computed using the following equations

$$\begin{bmatrix} F_b \\ F_{arm} \end{bmatrix} = M(-k_{vj}\dot{q} - k_{pj}(q - q_d))$$

where M is the Kinetic energy matrix of the first 10 links.

2) Arm control in World coordinates: This controller is needed to reach an object in the real world. This will move the arm robot and the base when it is needed. The base movement is necessary in order to reach an object in real world, otherwise it could be possible that the position and orientation needed to successfully grasp the object is not in the task space of the arm. We have two options for our controller depending on the goal: controlling only the position, or controlling the position and orientation simultaneously.

- *End effector position control*

$$\begin{bmatrix} F_b \\ F_{arm} \end{bmatrix} = J_v^T \Lambda_v \left(-k_v(\dot{x} - \nu\dot{x}_d)\right) + NM(-k_{vj}\dot{q}) \quad (1)$$

where ν is a factor to limit the maximum speed to go to the desired position. The desired speed $\dot{x}_d$ and ν are computed as:

$$\dot{x}_d = -\frac{k_p}{k_v}(x - x_d)$$
$$\nu = \text{sat}\left(\frac{V_{\max}}{\dot{x}_d}\right)$$

- *End effector position and orientation control*

$$\begin{bmatrix} F_b \\ F_{arm} \end{bmatrix} = J_0 \Lambda_0 \begin{bmatrix} -k_v(\dot{x} - \nu\dot{x}_d) \\ -k_p\delta\phi - k_v\omega \end{bmatrix} + N_0 M(-k_{vj}\dot{q}) \quad (2)$$

Where $\delta\phi$ is the angle error between the desired and current orientation computed as

$$\delta\phi = -0.5 \sum_{n=1}^{3} R_i \times (R_d)_i$$

where R_i and $(R_d)_i$ are the i-th columns of R and R_d

3) Arm control in base coordinates: This controller is needed to interact with the basket above the robot, since it is easier to describe this behaviour from the point of view of the robot.

This controller was divided in two, one that computes the needed toque in the base of the robot to maintain the same position and one that moves only the arm. The controller that moves the arm is given the desired position (x_d^B) and orientation in base frame (R_d^B) (frame attached to the base of the panda robot) and transformed to world coordinates using the following equation:

$$x_d = R_z(\theta_b)x_d^B + x_B$$
$$R_d = R_z(\theta_b)R_d^B$$

To compute the force of the arm joints (links 3 to 9) (F_{arm}) we apply similar controller than the ones expressed on equations (1) and (2) but changing the following terms:

$$J_{arm} = J[0:3, 3:10]$$
$$M_{arm} = M[3:10, 3:10]$$
$$L_{arm} = (J_{arm}M_{arm}^{-1}J_{arm}^T)^{-1}$$
$$N_{arm} = (I - J_{arm}^T(M_{arm}^{-1}J_{arm}^T L_{arm})^T)$$
$$\dot{q}_{arm} = \dot{q}[3:10]$$

This way

- *Arm position control*

$$F_{arm} = J_{v,arm}^T \Lambda_{v,arm} \left(-k_v(\dot{x} - \nu\dot{x}_d)\right) + N_{v,arm}M_{arm}(-k_{vj}\dot{q}_{arm})$$

- *Arm position and orientation control*

$$F_{arm} = J_{0,arm}\Lambda_{0,arm} \begin{bmatrix} -k_v(\dot{x} - \nu\dot{x}_d) \\ -k_p\delta\phi - k_v\omega \end{bmatrix} + N_{0,arm}M_{arm}(-k_{vj}\dot{q}_{arm})$$

To compute the joint forces of the base (F_b) required to maintain its position we used the following equation:

$$F_b = M_b(-k_{vj}\dot{q}_b - k_{pj}(q_b - q_{b,d}))$$

where $q_{b,d}$ is the position of the base and $M_b = M[0:3, 0:3]$ is the mass matrix of the base considering the inertial effect of the rest of the robot.

D. Gripper controller

This controller only controls the gripper joints; it will always run in cooperation with the previous controllers.

- *Open gripper*:

$$F_{grip} = -k_{vj}\dot{q}_g - k_{pj}(q_g - q_{g,d})$$

where $q_{g,d}$ is the open position of the gripper.

- *Close gripper*:

$$F_{grip} = F_{g,d} + (-k_{vj}\dot{q}_g - k_{pj}(q_g - q_{g,d}))$$

where $q_{g,d}$ is the position equivalent to the object width and $F_{g,d}$ the desired holding force.

E. Robot Arm Waypoints

Waypoints are used to move the robot arm such that the trajectory is collision-free and is most advantageous for grasping objects. When we pick up an object from the shelf, we first move the gripper to the position that has an offset normal to the object towards the middle of the aisle. This enables the gripper to approach the object and subsequently plunge into the shelf to retrieve it. After successfully grasping the object, it then returns to the aforementioned waypoint. To drop the objects into the basket, we have two additional waypoints: the first one is defined at an elbow-up configuration and the second one is defined directly above the basket. As the gripper moves from to the first then second waypoint, it will ensure that the none of the robot joints will collide with the basket.

F. Navigation Waypoints Generation

To move to the correct location for the pick-up operation, our robot must be able to plan a path between any location in the store to the position of the shelf object. Fig. 7 below shows a representation of the problem. The green triangle represents the location that the robot (red circle) needs to reach to perform the picking of the object from the shelf.

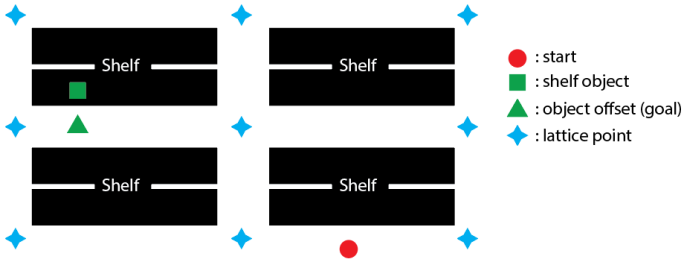

Fig. 7: Typical Setup of the Navigation Problem

Instead of using a traditional planning algorithm like A*, we make one key observation to drastically simplify the problem: any two points in a grid world like this where only horizontal or vertical movements are allowed can be connected with at most two additional "lattice points" to give an optimal path. "Lattice points" are defined as the set of points that are the intersections of the rows and columns of the map. The observation is rather self-explanatory and we will not provide a proof here.

Our algorithm separates this problem into three scenarios that account for when zero, one or two are needed to complete the optimal path. Algorithm[1] below defines our path planning process. The function *can_see* checks whether there is a line-of-sight between two points.

We first check for the trivial case where the start and goal are in direct line-of-sight, which will need no waypoints, as shown in Fig. 8.

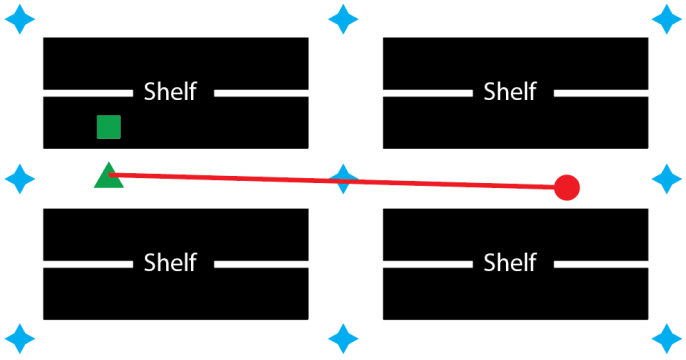

Fig. 8: Scenario 1: Zero Waypoints Needed

If there is no line of sight, we collect the set of lattice points seen by start and goal respectively. There can be at most one such point if start and goal don't already have direct line-of-sight. If there is a common lattice point that is seen, it will be the only waypoint needed, as shown in Fig. 9.

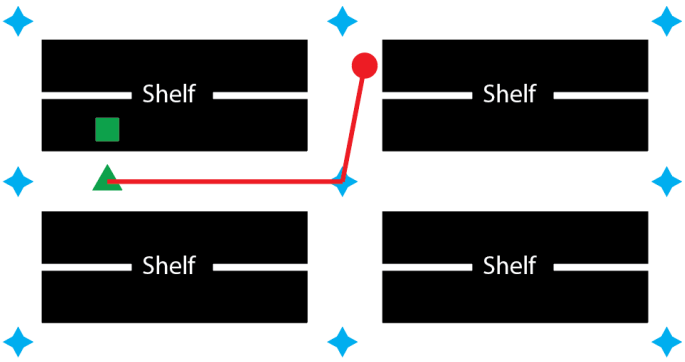

Fig. 9: Scenario 1: One Waypoint Needed

If the first two cases fail, we arrive at the general case where two waypoints are needed, as shown in Fig. 10. We then search through the possible routes from start to goal given the lattice points seen by start and goal respectively. The route that has the minimum $L1$ distance will be selected.

Note that in general, robot and self offset locations will not exactly align with the imaginary grid, and thresholds will need to be adjusted for this algorithm to perform correctly.

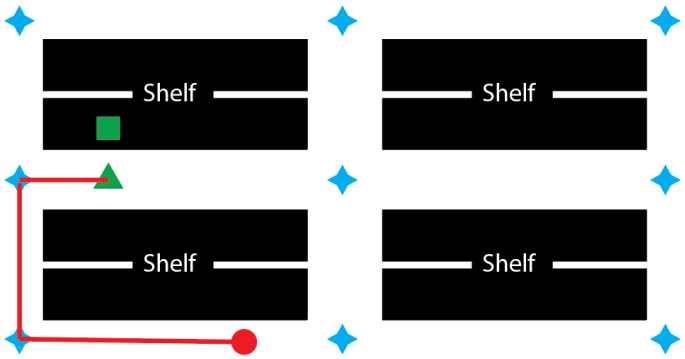

Fig. 10: Scenario 3: Two Waypoints Needed

III. Challenges

During the development of our project we faced the following challenges:

- *Confusion with the robot base frame*: For the basket controller we wanted to describe the dynamics of the robot in the base frame of the Panda Arm. For a while we tried to do this using the "Base frame" kinematics of SAI2, but we weren't getting the expected behaviour. Later we realized that this "Base frame" is really respect the "ground_link" and not the base of the Panda Arm. We opted to use a controller in world frame and transform the position and orientation from base frame to world frame.
- *Shelf and basket collision mesh*: Since the shelf is not a concave shape, we opted to create a "shelf.urdf" with each level of the shelf modeled as a fixed link with a thin cube as a collision mesh. Similarly the basket was created as a "basket.urdf" file. The wall of the basket

Input: s (start), g (goal)
Output: $waypoints$
Function `get_navigation_waypoints(`s, g`)`**:**
  $waypoints \longleftarrow \{g\}$
  if ***can_see****(s,g)* **then**
    do_nothing
  else
    foreach $p \in lattice$ **do**
      if ***can_see****(s,p)* **then**
        $start\_sees\_list$.**append**(p)
      end
      if ***can_see****(g,p)* **then**
        $goal\_sees\_list$.**append**(p)
      end
    end
    $p \longrightarrow start\_sees\_list \cap goal\_sees\_list$
    if $p \neq \emptyset$ **then**
      # at most only one point in common
      $waypoints$.**insert**(p,0)
    else
      # need one more waypoint
      $d_{min} \longleftarrow 0$
      foreach $p_1 \in start\_sees\_list$ **do**
        foreach $p_2 \in goal\_sees\_list$ **do**
          if ***!can_see***(p_1, p_2) **then**
            continue
          end
          $d \longleftarrow$ **L1**(s, p_1, p_2, g)
          if $d < d_{min}$ **then**
            $d_{min} \longleftarrow d$
            $p_{1,min} \longleftarrow p_1$
            $p_{2,min} \longleftarrow p_2$
          end
        end
      end
      $waypoints$.**insert**($p_{2,min,0}$)
      $waypoints$.**insert**($p_{1,min,0}$)
    end
  end
  return $wapoints$
End Function

Algorithm 1: Algorithm for Navigation Path-Planning

where created using fixed links, and other 6 DOF where added to allow the movement of the basket in the world.

- *Control the base of the robot and the arm independently*: For navigation we wanted to move only the base, and for movements in base frame we wanted to move only the arm. To decouple this behaviour we created the controllers described in the previous sections.
- *SAI .obj rendering issues*: For some reason, the way we had our lights set up caused objects with textures to render without shadows, which meant that complex objects were rendered completely flat with no discernible detail. Our solution was to leave simple objects (the products, walls, and floor) as-is, and remove the textures from the more complex objects (shelves and basket). This way, products were still able to be aesthetically pleasing without a significant loss in visual depth.

IV. Results and conclusions

Simulation is an important part when developing robots in order to test algorithms, state machines and controllers. This can help us to test our ideas without the risk of damaging the robot a person working with it. We learned how to implement the controller saw in classes in a more complex context, how to simulate force sensor and collision meshes, and how to navigate using lattice points.

In the future we would like to test our algorithm using a real robot, facing the challenges of how to retrieve sensors information to estimate the position of the objects in the world, as well as navigation in a real-world store environment.

V. Links

Video: https://youtu.be/f_2yzn_A708

Repository: https://github.com/gbravoi/cs225_Grocery_Team.git

VI. Nomenclature

F	Joint Force of the 12 joints of the robot
F_b	Joint Force of the base links
F_{arm}	Joint Force of the panda arm links
F_{grip}	Joint Force of the gripper
$F_{g,d}$	Desired holding force.
J_0	Basic Jacobian associated with the end effector.
J_v	Linear Motion Jacobian associated with the end effector.
k_p, k_{pj}	Proportional constants of PD controller.
k_v, k_{vj}	Derivative constants of PD controller.
Λ_0	Operational space basic kinetic energy matrix associated with the end effector.
Λ_v	Operational space linear motion kinetic energy matrix associated with the end effector.
M	Kinetic Energy matrix
N	Nullspace of J_v
N_0	Nullapce of J_0
ν	Factor to limit maximum velocity.
$V_{\max}$	Maximum linear velocity on the end effector
ω	Angular velocity of the end effector.
$\delta\phi$	Orientation error
q	Robot's joint positions.
$\dot{q}$	Robot's joint velocity.
q_B	Joint angles that contains the base position and orientation $[x_b, y_b, \theta_b]$
$\dot{q}_B$	Speed of the base joint angles.
$q_{B,d}$	Desired position and orientation of the base
$q_{arm,home}$	Panda robot joints home position.
q_g	Gripper joints position
$q_{g,d}$	Desired gripper joints position
$\dot{q}_g$	Speed of the gripper joints
R	End effector orientation expressed as rotation matrix.
R_d	Desired end effector orientation expressed as rotation matrix.
$R_z(\theta)$	Rotation matrix in z axis by θ angle.
θ_b	Heading of the base of the robot
x	Position of the end effector.
x_d	Desired position of the end effector.
x_B	position of the base of the panda robot ($[x_b, y_b, z_b]$).
x_d	Desired position of the end effector.
$\dot{x}$	Linear velocity on the end effector.
$\dot{x}_d$	Desired velocity on the end effector.