097244 - Cognitive Robotics

Final project report

Submitted by:

Vitaly Pankratov 961152147

Benjamin Barnes 941180010

Noah Bernten 941180002

Supervisor:

Prof. Erez Karpas
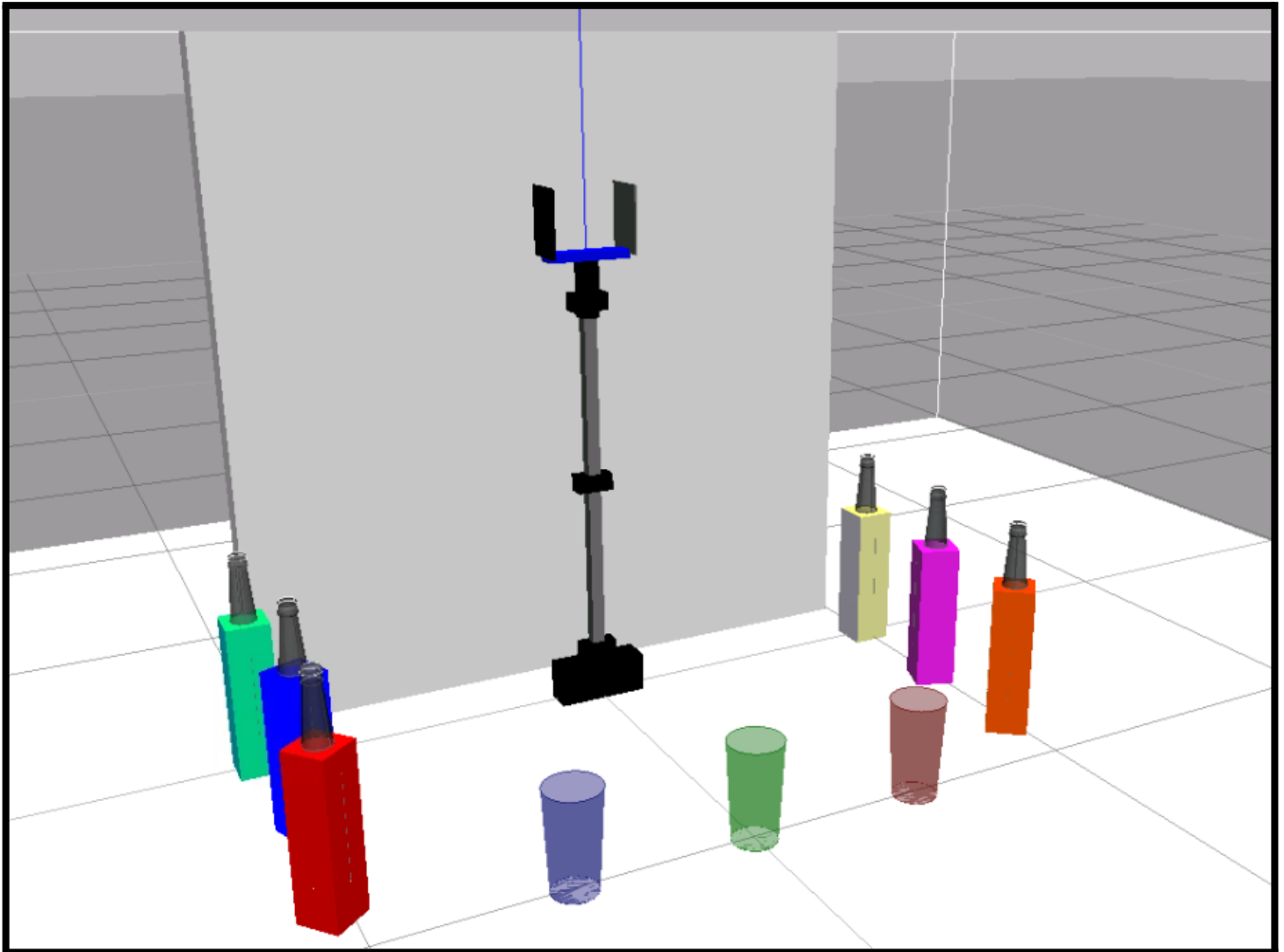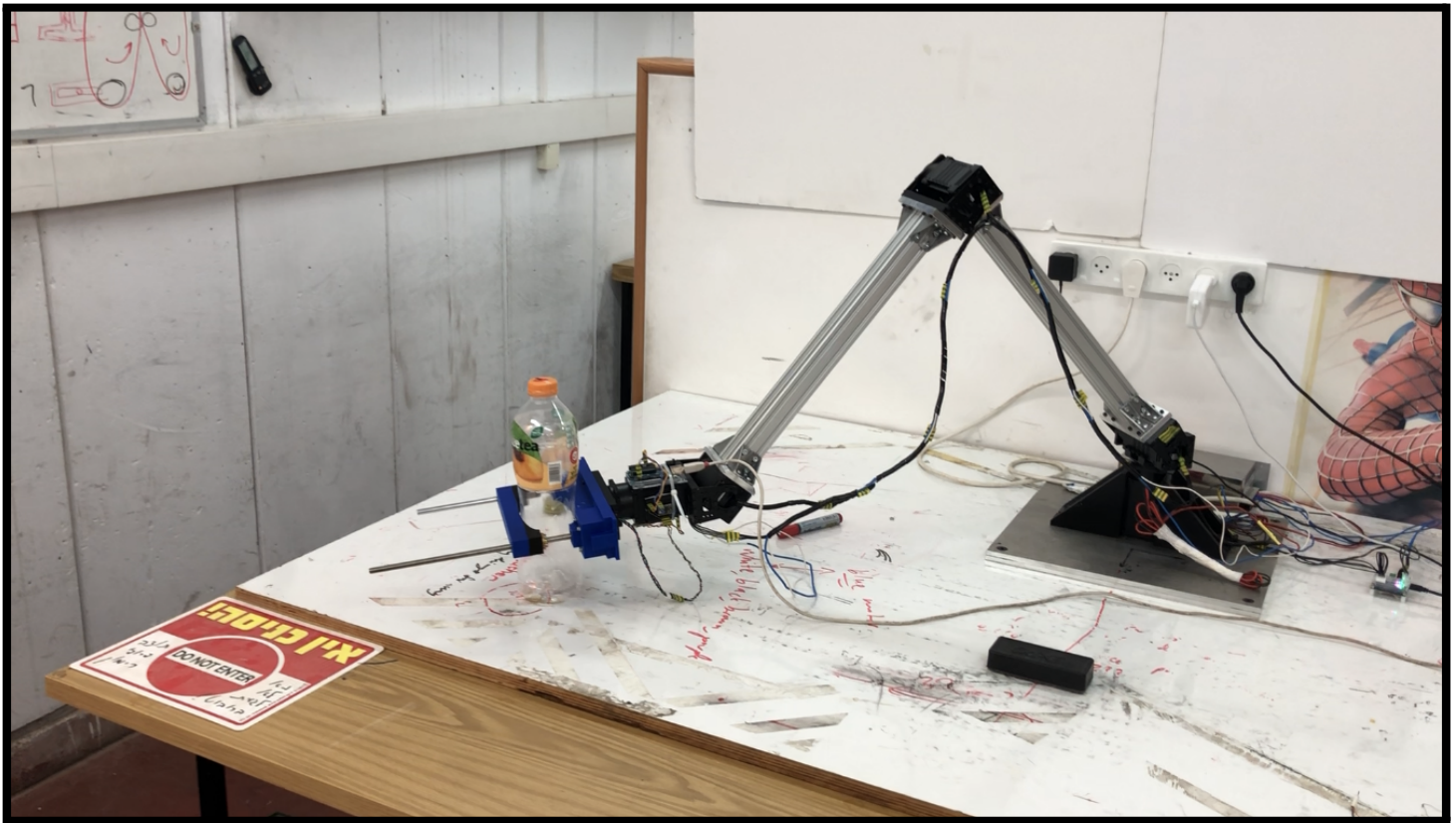
October, 2021

Table of Contents

1.   Introduction

The idea of the project is to create a cognitive robot bartender that makes use of 'planning' in order to take a list of customers' cocktail orders and determine how to pour the drinks. We use a planning domain definition language to provide the robot operating system with the actions necessary for mixing cocktails. The project was implemented both in simulation on ROS and with a real robot we built from scratch, which made the work on the project especially interesting. The challenge was created to fit with the constraints we would encounter when implementing it on our single armed robot. We settled on a menu of 7 cocktails which can each be produced by different combinations of the six bottles. The menus and constraints can be scaled up for other applications.

2.   Body of the project

2.1.   Project setup

6 bottles:	3 cups:
1. Red bottle - rum 2. Blue bottle- vodka 3. Green - absinthe 4. Orange - juice 5. Pink - soda 6. Yellow - whiskey	1. Red 2. Green 3. Blue

2.2. Project stages

- <u>Choose cocktails</u>

First of all, the user chooses 3 of the 7 cocktails that he wants to order.

Cocktails options:

  - RandS - Rum and Soda
  - VandJ - Vodka and Juice
  - WandJ - Whiskey and Juice
  - RVAJ - Rum, Vodka, Absinthe and Juice
  - RVAJSW - Rum, Vodka, Absinthe, Juice, Soda and Whiskey
  - RASW - Rum, Absinthe, Soda and Whiskey
  - WAJ - Whiskey, Absinthe and Juice

- ## Transfer cocktails to PDDL

  When cocktails are chosen we write their names into the PDDL problem file. Domain PDDL file is obviously equal for all sets of cocktails.

- ## Transfer PDDL to IPC plan

  When the PDDL problem file is ready we send it with the domain file to the online PDDL solver at http://solver.planning.domains/solve . Since our planning task isn't very hard this online pddl service returns a solution in about 1 second. Solution is returned in the form of an ipc file[1].

- ## IPC plan to Python actions

  Now when we have a ready action plan we only have to transfer it to python commands. In order to do so we translate ipc plan into a list of steps, where each step represented by:

  - Action that robot performs (grab, pour, release, serve and finish)
  - Bottle number that robot operates at current step
  - Cup number that robot operates at current step

  In case the action doesn't need a bottle or cup, the field for it remains empty. Here is the example of such plan:

```
{'action': 'grab', 'bottle': 'bottle_1', 'cup': ''}
{'action': 'pour', 'bottle': 'bottle_1', 'cup': 'cup_1'}
{'action': 'pour', 'bottle': 'bottle_1', 'cup': 'cup_2'}
{'action': 'release', 'bottle': 'bottle_1', 'cup': ''}
{'action': 'grab', 'bottle': 'bottle_4', 'cup': ''}
{'action': 'pour', 'bottle': 'bottle_4', 'cup': 'cup_1'}
{'action': 'release', 'bottle': 'bottle_4', 'cup': ''}
{'action': 'serve', 'bottle': '', 'cup': 'cup_1'}
{'action': 'finished', 'bottle': '', 'cup': ''}
```

- ## Perform actions

  The last step of the process is the robot performing this sequence of actions. Simulated robot takes from 5 to 20 minutes to complete an order of 3 cocktails depending on the number of ingredients. This slow pace of work is due to two factors:

  - A site with a simulation does not provide an opportunity to get a high frame rate even on powerful computers, and the simulation itself does not work at full speed, so the robot moves as if in slow motion.
  - In addition, the robot itself is limited in movement speed, since the bottles it holds are quite bulky and it is important to hold them firmly all the time.

---

[1] Link to the plan example file: https://github.com/Stayermax/5dof-bartender-robot/blob/main/pddl/plan.ipc

3. Project details
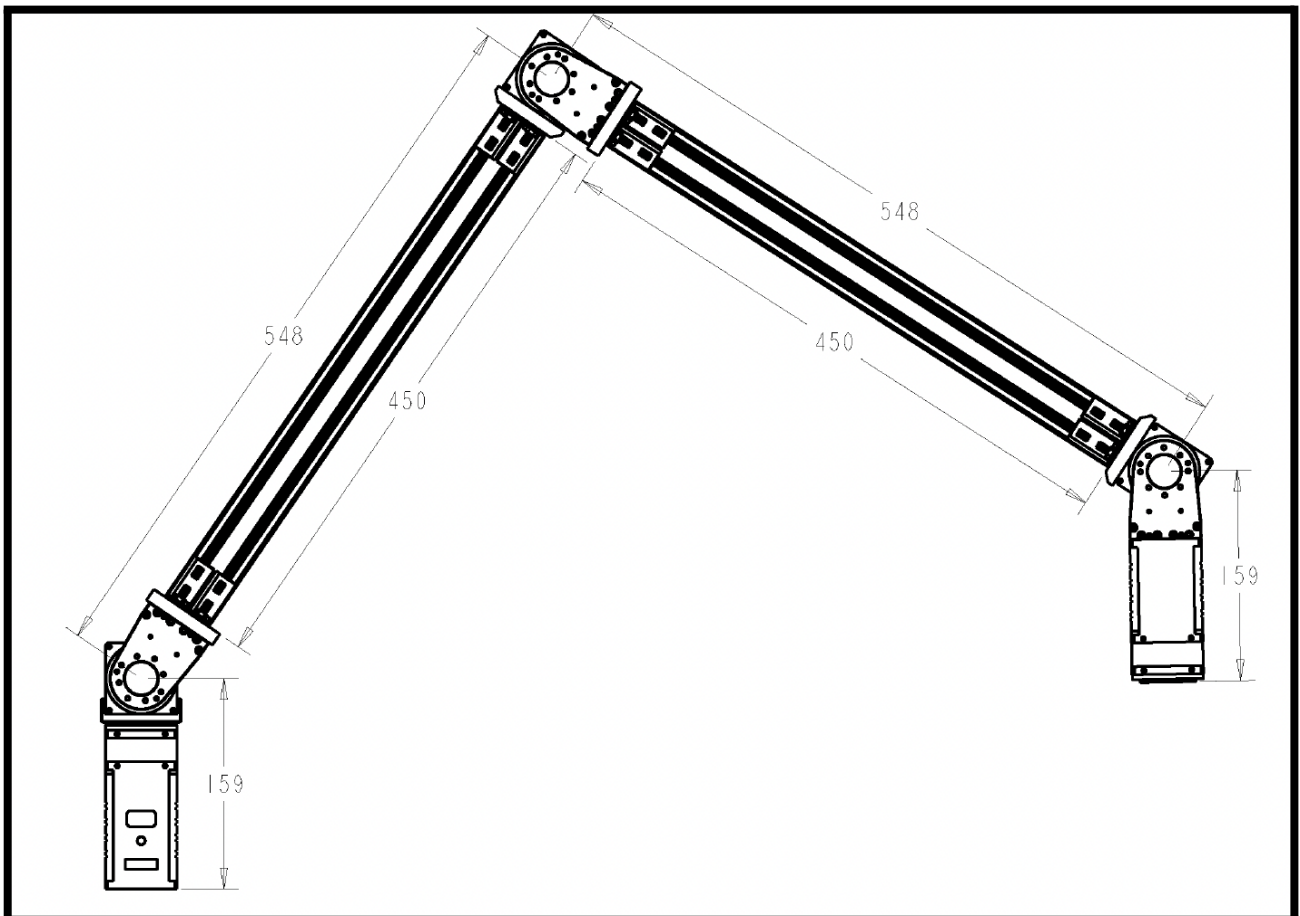
3.1. Software details

The software used for this project is as follows: ROS kinetic was run on an Ubuntu 16.04. Although this is an outdated distribution it is the most stable distribution for the actual robot used. The main program used on ROS was MoveIt, which was used for motion planning. Further an online PDDL solver was used to solve the robots planning problems in real time. This seemed like a very elegant solution and for this reason it was chosen as opposed to ROS plan. Gazebo was also used for simulations as well as RVIZ for robot visualization.

The entire project was created on the Constructsim online ROS Development platform. This allowed for easy sharing of the project between group members and it also allowed us to develop in ROS without the need for a local ubuntu installation. There is a possibility of running the software on the online platform and connecting it to the real robot however this is only possible for projects created in ROS Noetic. During the conversion of our project to Noetic something was corrupted and we were unable to run our project in Noetic. For this reason it was decided to replicate our project on a local version of ROS in order to run it on the real robot.

3.2. Hardware details

Dimensions:

Specifications:

Items	Robot arm
DOF	5
Payload	3 Kg
Operating voltage	24 V
Joints	$Joint1: H54-200-S500-R$ $Joint2: H54-200-S500-R$ $Joint3: H54-200-S500-R$ $Joint4: H54-100-S500-R$ $Joint5: H54-100-S500-R$
Operating Range	$Joint1: -\pi(rad) \sim \pi(rad)^{1}$ $Joint2: -100(deg) \sim 100(deg)^{2}$ $Joint3: -\frac{2\pi}{3}(rad) \sim \frac{2\pi}{3}(rad)^{2}$ $Joint4: -\frac{2\pi}{3}(rad) \sim \frac{2\pi}{3}(rad)^{2}$ $Joint5: -\pi(rad) \sim \pi(rad)$
Default ID	$Joint1: ID001$ $Joint2: ID002$ $Joint3: ID003$ $Joint4: ID004$ $Joint5: ID005$

The robot used was a robot based on the Robotis Manipulator P. Robotis is a South Korean company that specializes in smart server motors which work especially well in ROS. There was a need for the robot to be larger than the original manipulator P and cheaper , and for this reason a custom robot was built using the same motors but different linkages. The main difference however is that the robot used is a 5 degree of freedom manipulator. Since the robot is a 5R robot the inverse kinematics were easily solved. This was important since the MoveIt inverse kinematics solver does not work well for less than 6 degrees of freedom robots. All this was built for a final project in mechanical engineering.

The hardware built for this course's project is as follows: a well suited gripper was designed to allow for positioning errors when trying to pick up a bottle. It also securely grips the bottle and it is very simple to operate. First the gripper was designed in Solidworks and then it was 3D printed , using a slicing software called Cura and an Artillery sidewinder X1 3D printer was used. The gripper was then added to the simulation, however after many attempts we were unable to successfully grab objects with our gripper in the simulation. It was then decided to create a different gripper for the simulation in order to prove that our algorithm worked and that the robot operated safely before attempting to run our program on the real robot. Once we were satisfied with the simulation we began testing on our real robot. This included making use of Rosseriel in order to control our gripper by an Arduino.

3.3. IK

Since we created the robot by ourselves we also had to implement an inverse kinematics algorithm. Here is the list of formulas that we used, assuming the position of the object that we want to take is (x, y, z).

```python
L = 0.54 # Length of arm elbow
l_base = 0.16 # base height
l_45 = 0.28  # length from joint 4 to joint 5
length = sqrt(x ** 2 + y ** 2) # distance to object

h = abs(zp - l_base)  # distance in z direction between 1st and 4th joints
D = length - l_45 # distance from the base to the beginning of the gripper
S = sqrt(h ** 2 + d ** 2) / 2 # half distance from the base to the gripper

theta_0 = arctan(x / y) # angle on the target
```
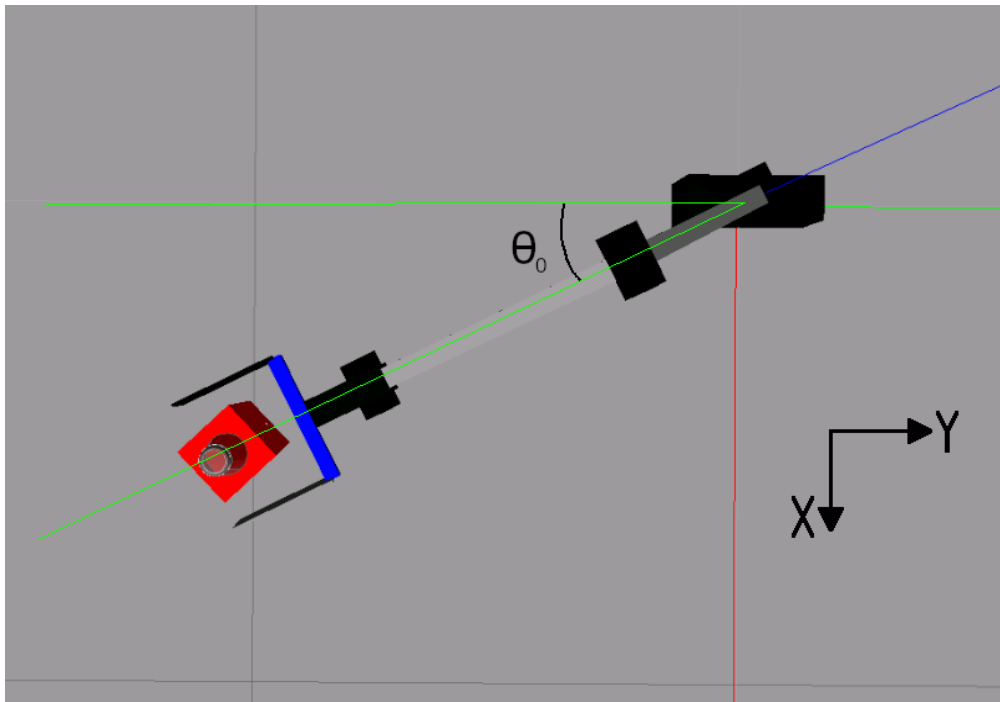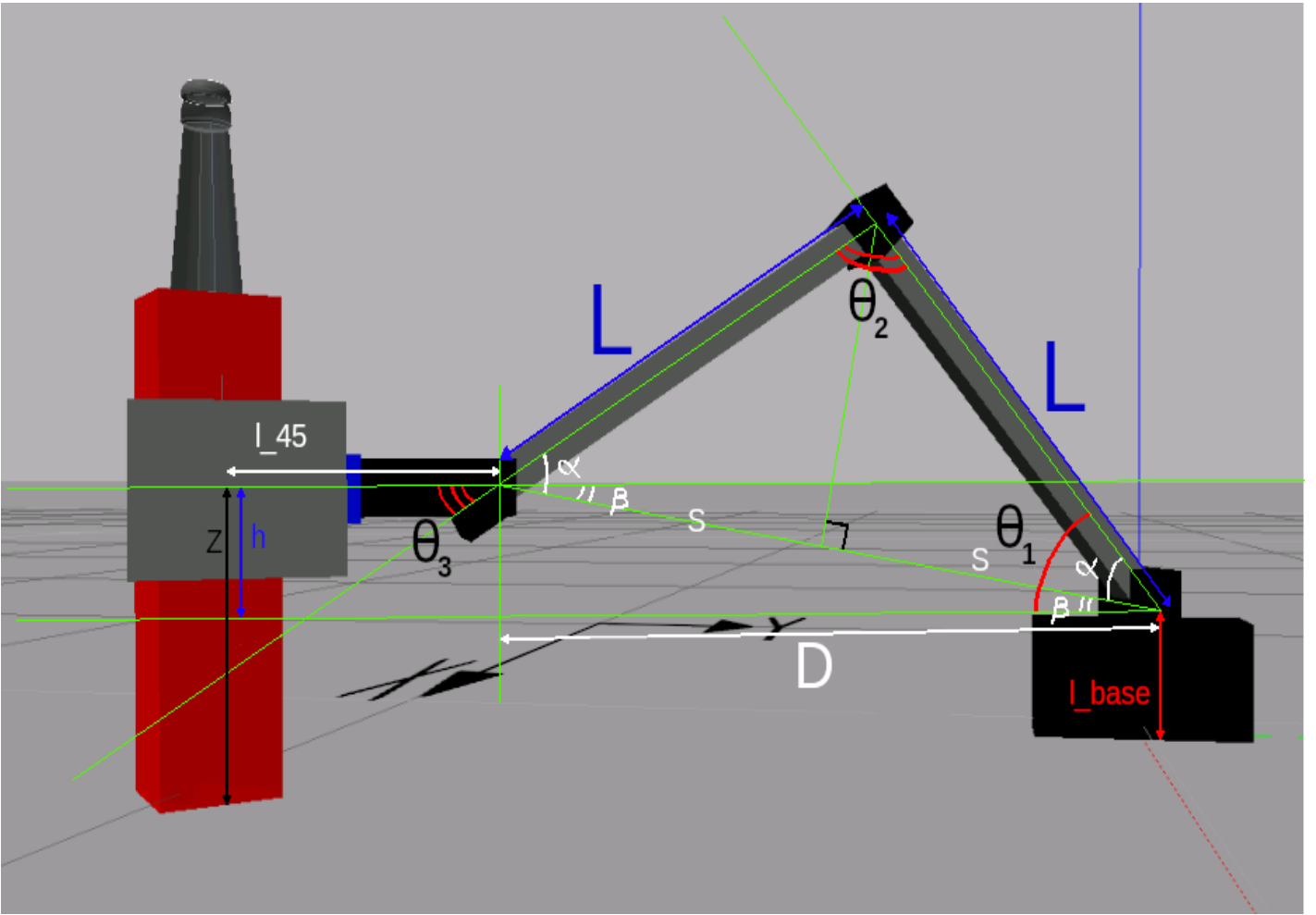


```python
alpha = acos(S / L)
beta = acos(D / (2 * S))

theta_1 = pi / 2 - (alpha + beta)
theta_2 = 2 * alpha
theta_3 = -(theta_1 + theta_2) + pi / 2
```

In case the object z coordinate is lower than l_base or higher than $l\_base + L$ calculations are almost the same.

3.4. PDDL

PDDL is a planning language which uses heuristic search algorithms like A* to solve problems without human decision. To familiarize the solver with the parameters of the challenge, we create a domain file which defines the constraints for each important action that the robot can take. The solver then takes a problem file and uses all of the possibilities to pick an efficient solution. In our case, the problem is the cocktail orders and the solution is a list of actions which the robot performs to produce the cocktails efficiently. The planning was made for the possibility of adding more robotic arms, cocktails, and bottles.

Domain file[2]:

The important components to our domain file are the actions which the robot can choose to do which will further the process of making cocktails. All of the actions have predefined cases which trigger their use and afterwards modify various true or false variables as the effects. Variables and objects were created according to the actions needed.

The action options are:

● Grab - The robot can pick up a bottle when the bottle is on the table and the gripper doesn't hold anything. When the robot grabs the bottle, the bottle is then in the gripper and the gripper isn't free.

---

[2] Link to the domain file: https://github.com/Stayermax/5dof-bartender-robot/blob/main/pddl/Domain.pddl

- Release - When the gripper is holding a bottle which is not being demanded, then the gripper puts down the bottle.
  - We initially didn't hardwire the release to occur when the bottle is not demanded because we thought that the AI would figure out on its own when it should release. However our planner wasn't able to find the optimal solution, when it could release the bottle anytime it was holding it. So we had to hardwire the precondition for the release to only trigger once the bottle has been used for each cup that demands it.
- Pour - When the bottle is in the gripper, the cocktail's recipe demands that drink and it currently does not exist in the corresponding cup then we pour the drink into the correct cup and it starts to contain the drink.
- Serve - When an arm is empty and a cup contains all of the ingredients of the ordered cocktail then it is ready to be served. Once the cup is served then it is completed which is important for finishing. Cup serving can only be seen in the console log, the robot doesn't show it by it's actions.
- Finished - once every order that is in the system is completed, then the robot is done working and goes to initial state.

The domain file is written to work with the problem file. Pretty much the only thing to change in order to update the domain for other setups, can be the number of arms the robot has. The number of recipes and bottles can be modified in the problem file.

Problem file[3]:

The problem file is where the starting conditions for every variable are set as well as the goal. The sections of the problem file get updated when there are new orders and together with the domain new plans are formed.

Some useful expressions in the problem file for the bartender are order and recipe.

- Order - The idea is to create a starting condition called orders which matches the desired cup with a cocktail so that the actions in the domain file knows how to appropriately fill out orders. (order numbers can also be attached from the user to the cup)
- Recipe - The recipe expression matches a drink with a cocktail which allows the actions in the domain to determine if they are using the correct drinks for the desired cocktails.

Besides for those two unique expressions, the rest of the variables are just true or false starting conditions. The last part of the problem is setting the final state which we defined as being done. The goal of the problem is to be done and being done is the resulting effect of the finished action in the domain file.

---

[3] Link to the problem example file: https://github.com/Stayermax/5dof-bartender-robot/blob/main/pddl/Problem.pddl

4. Experiments

4.1. Parameters tuning

In the beginning of the project we encountered problems with grasping bottles in simulation. This is a widely known problem for ROS Kinetic. We tried to fix it by changing robot and bottle parameters and by installing the grasp fix plugin by Jennifer Buehler.

Parameters that we had to change:

- Model and gripper friction and torsional friction
    - All parameters were set to maximum possible value
- Model weight
    - We had to reduce bottles weight to 50 grams
- Gripper effort
    - Gripper effort was chosen in such a way as to firmly hold the bottle in the hand, but not to push it out of the fingers.
- Gazebo graspin plugin
    - Unfortunately for us, the plugin still has some imperferfection since it was created by an open source developer and we couldn't make it work, it just broke the simulation.
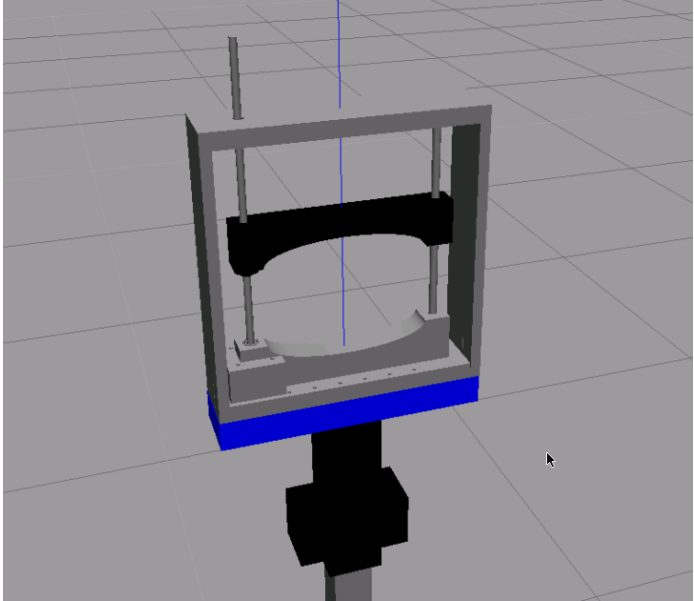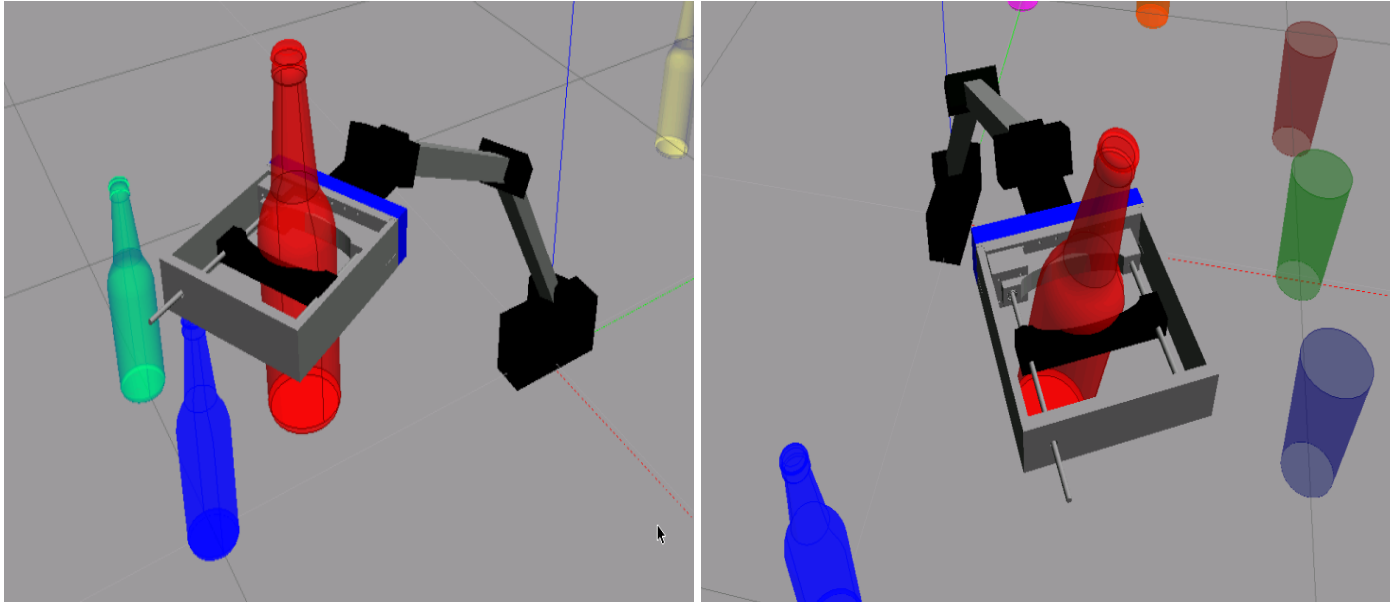
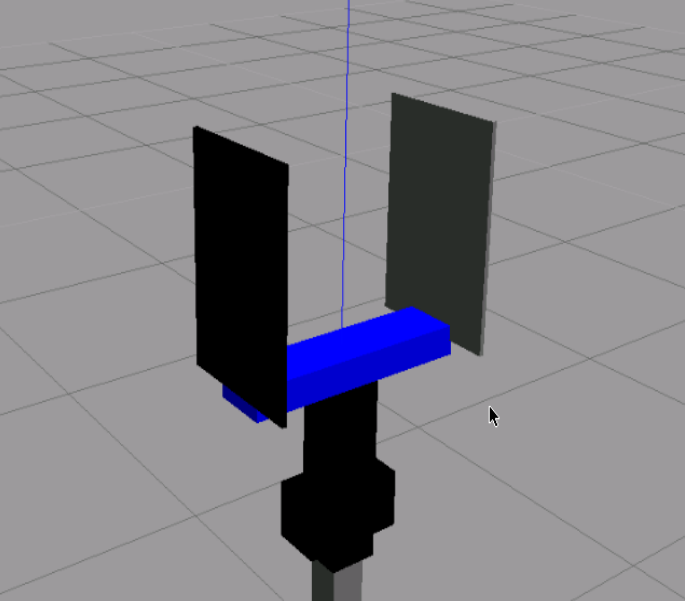## 4.2. Gripper variations

Real world gripper:
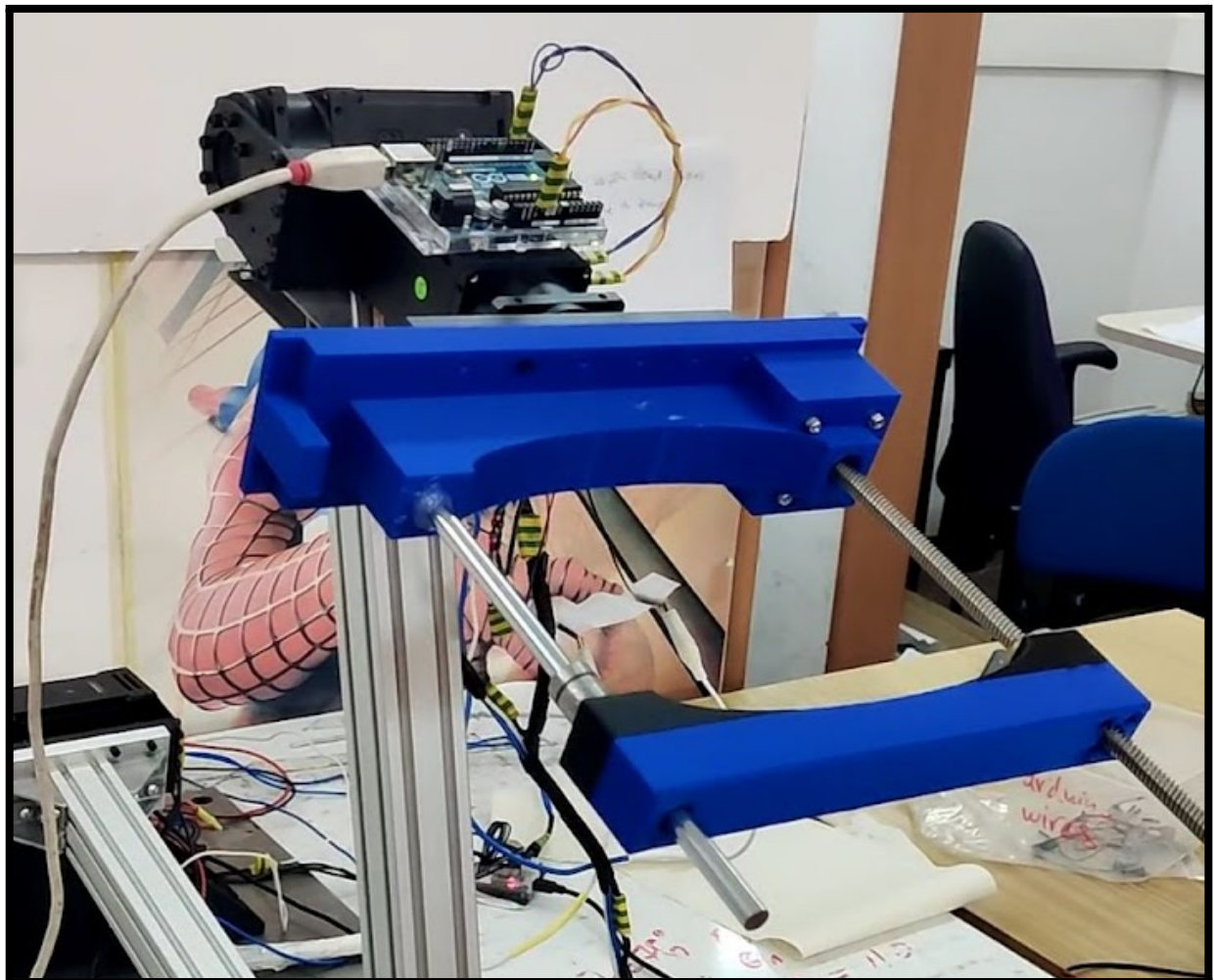
<u>Simulation grippers:</u>

During our work in simulation we designed two types of grippers. First one was an exact replica of the real world gripper model (but with frame). It had one moving part which is black plate that pushed bottle and hold it like this:
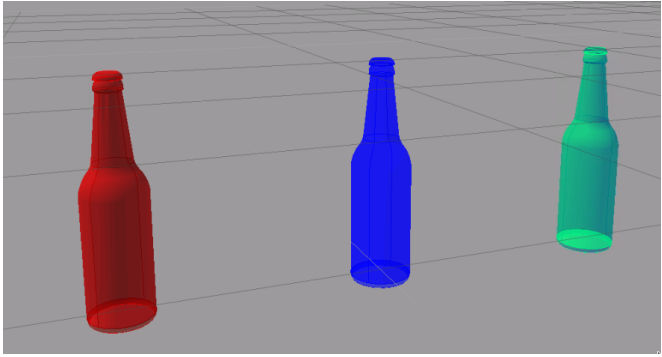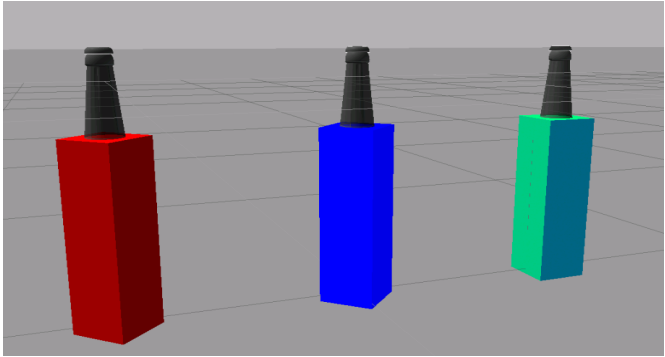

Unfortunately we couldn't make this gripper work and had to change the gripper to a much simpler model with two fingers.

Old gripper	New Gripper

4.3.   Bottles variations

Firstly we used regular beer bottle models with cylindrical shape base, but since we decided to change the form of the gripper inside of the simulation we converted to using Jack Janiels bottle shape. Square basis makes bottles more steady and stable which is important because robot actions are not always precise.

Old bottles models	New bottles models

5.   Results

5.1.   Simulation videos

Links to the simulation runs:

- Run number 1: https://www.youtube.com/watch?v=HwMrsE0jKWQ
- Run number 2: https://www.youtube.com/watch?v=V-YA655vqGA
- Run number 3: https://www.youtube.com/watch?v=CqfJ0AyVzlY

5.2.   Real world video

Real world run:

- Run number 1: https://www.youtube.com/watch?v=2sgOBlgd5fw
- Run number 2: https://www.youtube.com/watch?v=WSWCvtBugL4

6.   Conclusions and Recommendations

During the work on this project, we learned a lot, since the work included software and hardware development, working with PDDL, working in a virtual environment and interacting with ROS.

Despite the fact that the final product looks finished, it has room for development. Since the model in the simulation does not depend on the robot in the real world, you can think about creating a better model that will work on Python 3 and in newer versions of Gazebo. In addition, we were never able to connect the plugin for grasping objects and perhaps this could add stability to the robot.

From the point of view of the environment in which we worked, Constructsim.com is good for projects like ours since it allows you to work with simulation without installing Ubuntu. However, the site performance is not very high, and for a more complex project, you

should use simulation environments on a powerful computer, and not in the browser version of Gazebo.

Making use of a DIY robot as opposed to an off the shelf solution definitely has its limitations. Since the robot only had 5 degrees of freedom many MoveIt features were missing. Further, there was a lot of vibration at the base of the robot which made putting bottles down very difficult. On the other hand, we did not have to understand other people's libraries and APIs, which can significantly complicate the work with open source projects. Plus, your own code is always clearer and easier to use.

7. Conclusion

Although it may seem like the main focus of this project was the hardware and its integration with the robot operating system. The most interesting part of the project was certainly the planning aspect. It's true a large amount of time and effort was spent creating a working robotic environment. However, this was not something particularly new to us. On the other hand, integrating planning into our robot is something none of us has ever done before. Initially it was very difficult to fight the urge to tell the robot what to do in code. Instead we had to simply define the world, the rules and the goal and let the robot do the rest. Once we overcame this initial challenge the results were extremely interesting and we were all surprised and excited to observe as our bartender effortlessly created cocktails, without being told how to do so. Although, we only made use of the concept of planning for a very simple and fun application. Thanks to this project we now understand the true capabilities in planning in cognitive robotics and we are very excited to make use of this on far more complex projects in the future.

8. References

1. Link to the Github - https://github.com/Stayermax/5dof-bartender-robot
2. Link to the PDDL solver - http://solver.planning.domains/solve
3. Link to the Gazebo grasp-fix-plugin - https://bit.ly/2XH6dJc
4. Link to the simulation environment - https://www.theconstructsim.com