PASTABOT:

FORCE SENSING FOR PASTA PACKAGING AUTOMATION

Alberto Compagnoni
Luca Inghilterra
Alessandro Monteleone

Smart Robotics - Academic Year 2023/2024

Project Description & Objectives

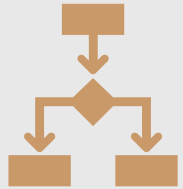
- Manipulating object by pushing them, the system sorts pasta boxes based on their mass classification
- Designed for use cases where the packaging of different products is standardized

1. Box Localization

Identify the correct push point based on the collected image

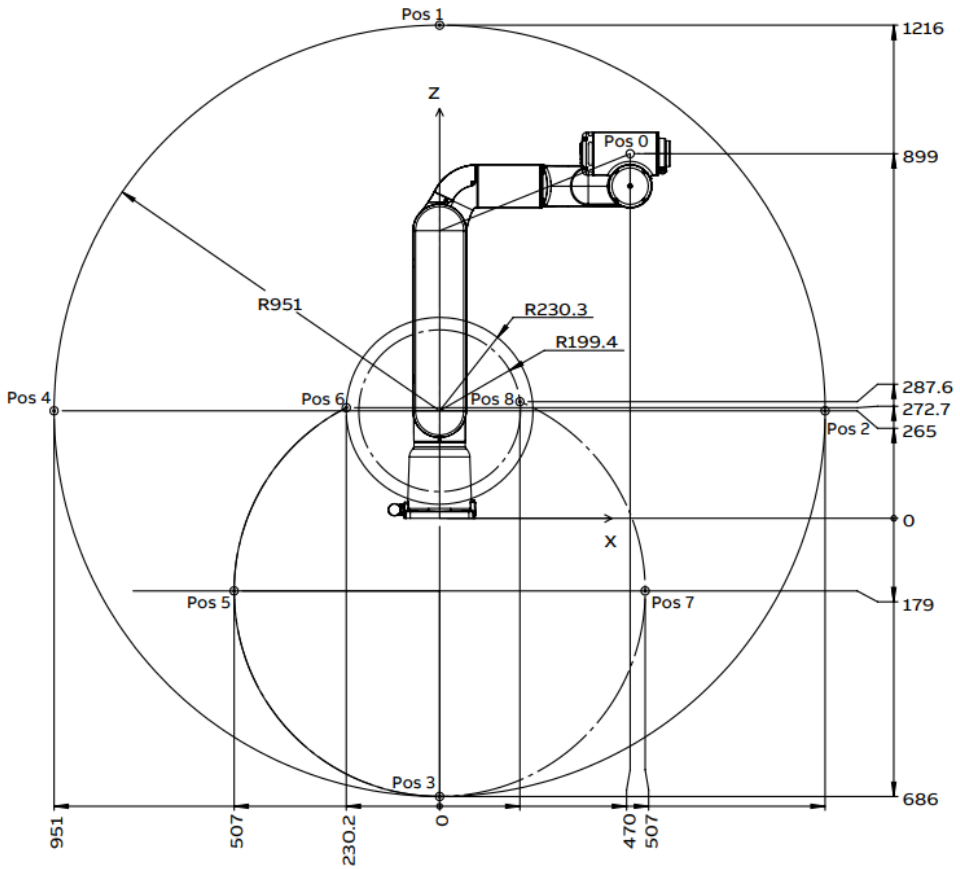
2. Sensing

Sense force through controlled pushing interactions.



3. Sorting

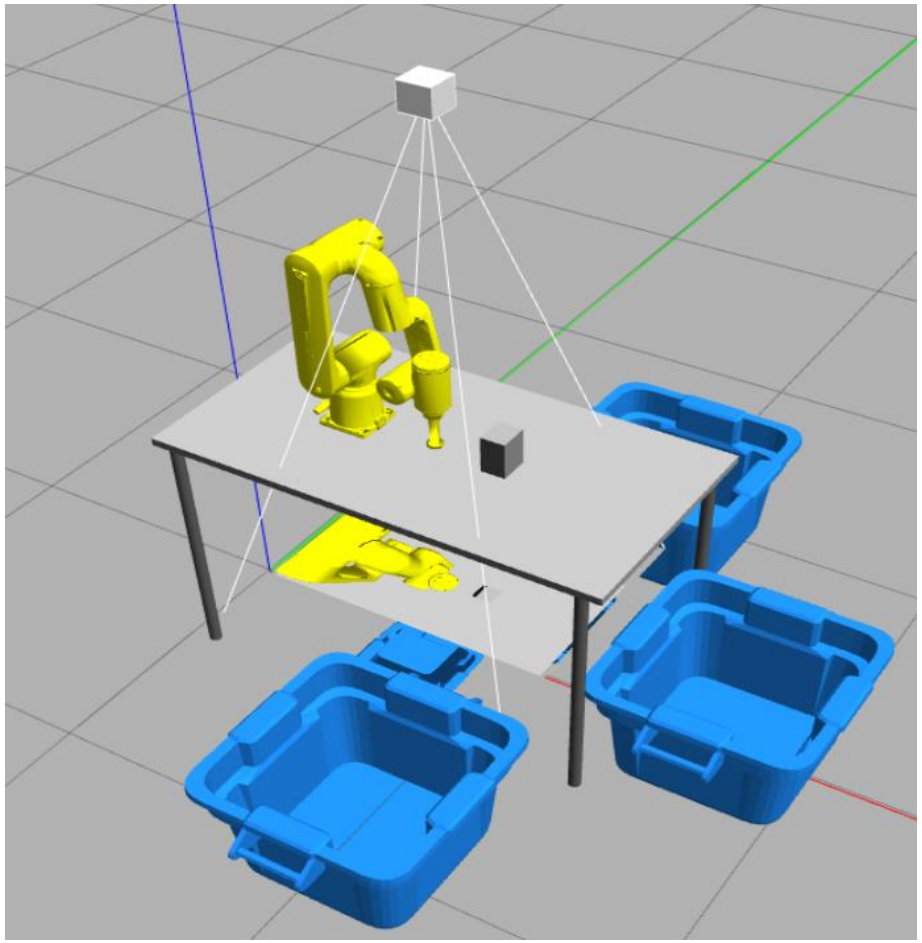
Push the box into the correct disposal bin.



01 GoFa 5 Working range, Side view

Manipulator

- ABB GoFa™ CRB 15000
- 6 DOF
- Revolute Joints
- Pushing Tool
- 6D Force/Torque Sensor attached to the joint 6



Setting

- **Table**: center of the scene
- **Box**: placed in a default position (hypothetically taken by a conveyor/roller belt), with variable mass
- **Disposal bins**: three ones, each one as destination for a different mass item
- **Fixed Camera**: FoV over the table
- **ABB GoFa™ CRB 15000**

Tools

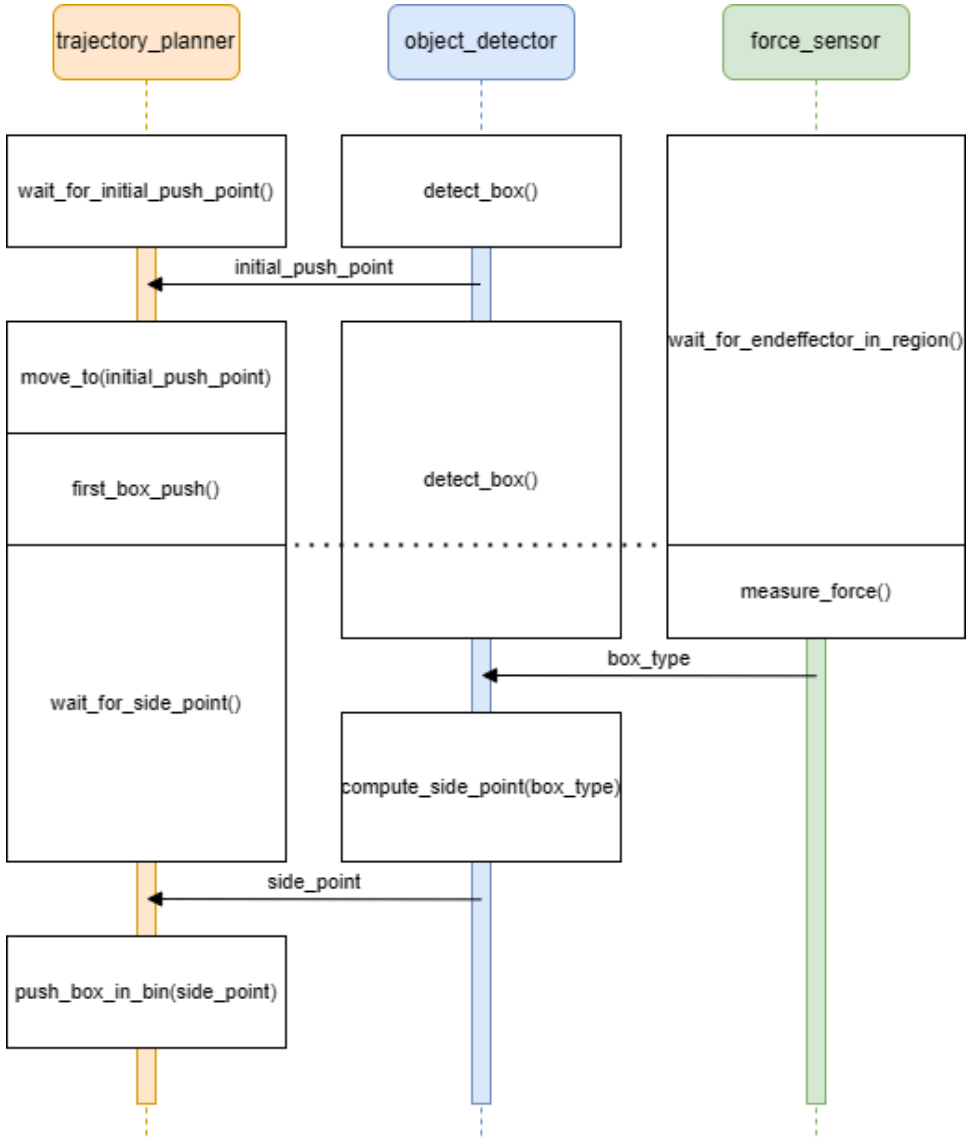
Ros

Gazebo

OpenCV

Python

MoveIt



System Design: Sequence Diagram

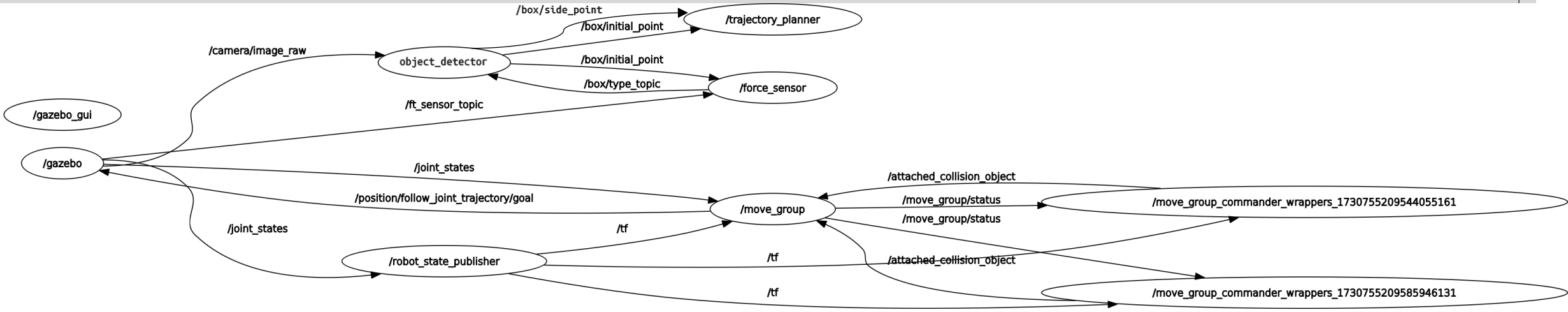
System Design – ROS Nodes and Topics

object_detector		
Topic	**Type**	**Meaning**
/camera/image_raw (S)	Image	Image read from top-view camera
/box/type_topic (S)	String	Classified box type “LIGHT_BOX” “MEDIUM_BOX” “HEAVY_BOX”
/box/initial_point (P)	Point	Initial point where the box is located
/box/side_point (P)	Point	Point at which the robot must apply the second push to the box in order to drop it into the disposal container

force_sensor		
Topic	**Type**	**Meaning**
ft_sensor_topic (S)	WrenchStamped	Force and Torque read by sensor
box/initial_point (S)	Point	Initial point where the box is located
box/type_topic (P)	String	Classified box type “LIGHT_BOX” “MEDIUM_BOX” “HEAVY_BOX”

trajectory_planner		
Topic	**Type**	**Meaning**
/box/initial_point (S)	Point	Initial point where the box is located
/box/side_point (S)	Point	Point at which the robot must apply the second push

System Design – Full ROS Graph



1. Box Localization

Image Acquisition & Preprocessing

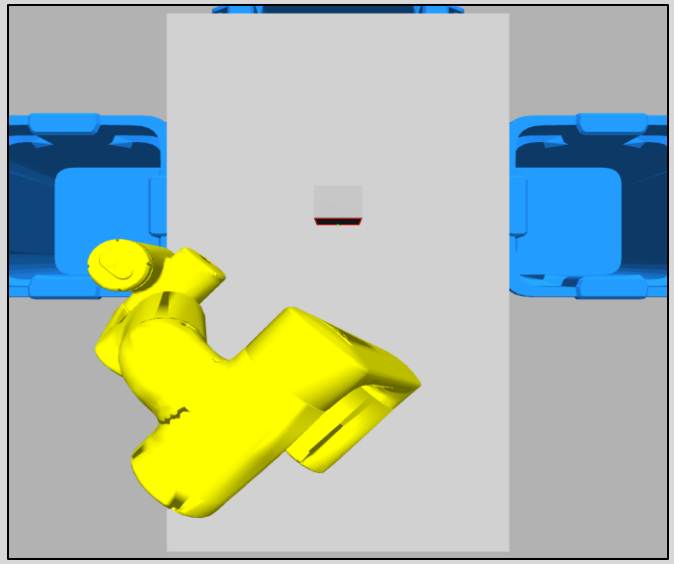
Object Detection by Contour Recognition

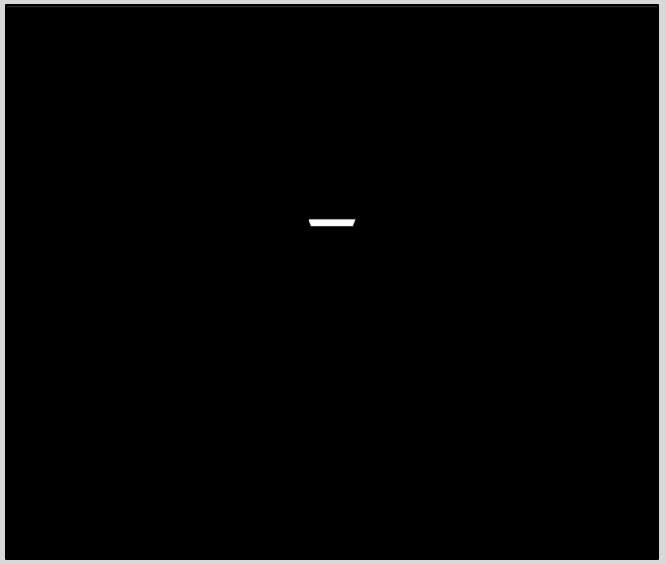
Push Point Extraction

Homography

Image Acquisition & Preprocessing

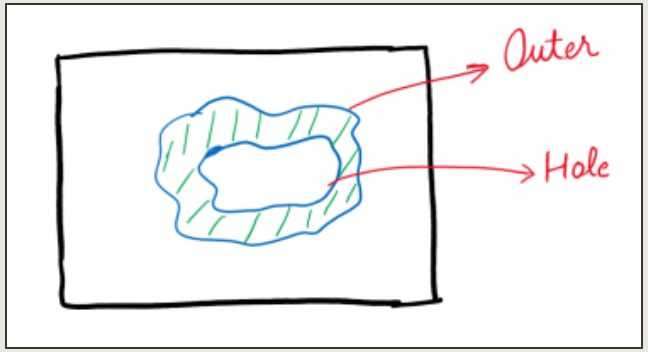
- **Image Capture**: Collect the image and convert it from ROS format to an OpenCV image
- **Cropping**: Performed on the image to focus on RoI in which boxes are located
- **Color Filtering**: Using color thresholding, segment the dark gray regions that correspond to the box's front face (dim light)





Contour Recognition & Object Detection

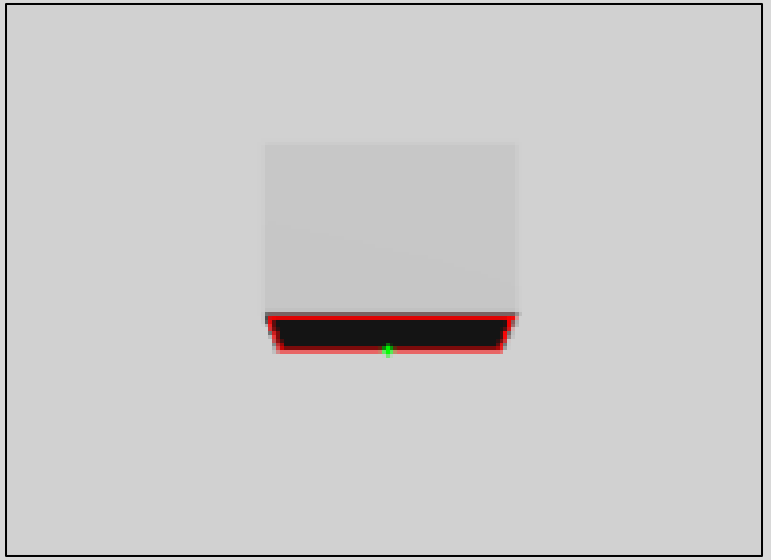
- **Suzuki-Abe algorithm** (`cv2.findContours`)
  - **Pixel Search**:
    - Starts by looking for pixels that are part of a **contour** (usually white pixels in a binary image) and, using neighbors, traces the perimeter of the object, following changes in intensity.
  - **Hierarchy of Contours**:
    - The algorithm can also build a contour hierarchy, allowing it to distinguish between outer and inner contours (e.g., holes in an object).



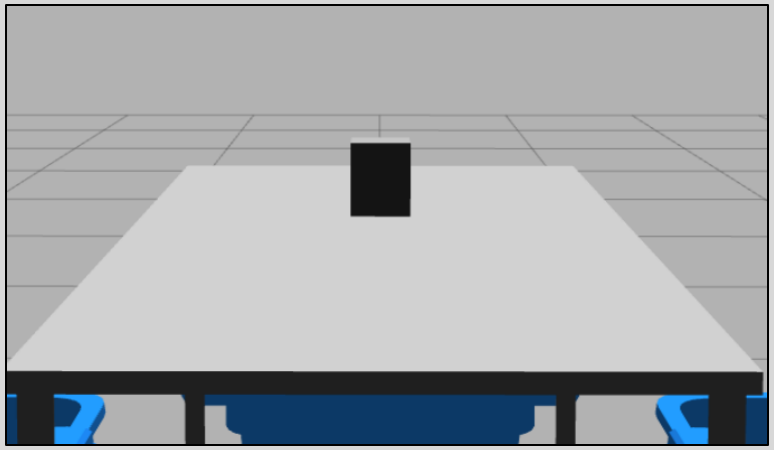
- **Object selection**
  - Discard polygons which area is not reasonable

Push Point Extraction

- **Identifying the Push Point**
  - Located at the center of the box's front face bottom edge
  - Initially expressed in pixel coordinates (x, y)



- **Coordinate Transformation**
  - Homography matrix converts pixel coordinates to real-world space
  - Enables accurate spatial positioning for robotic manipulation



Homography Estimation

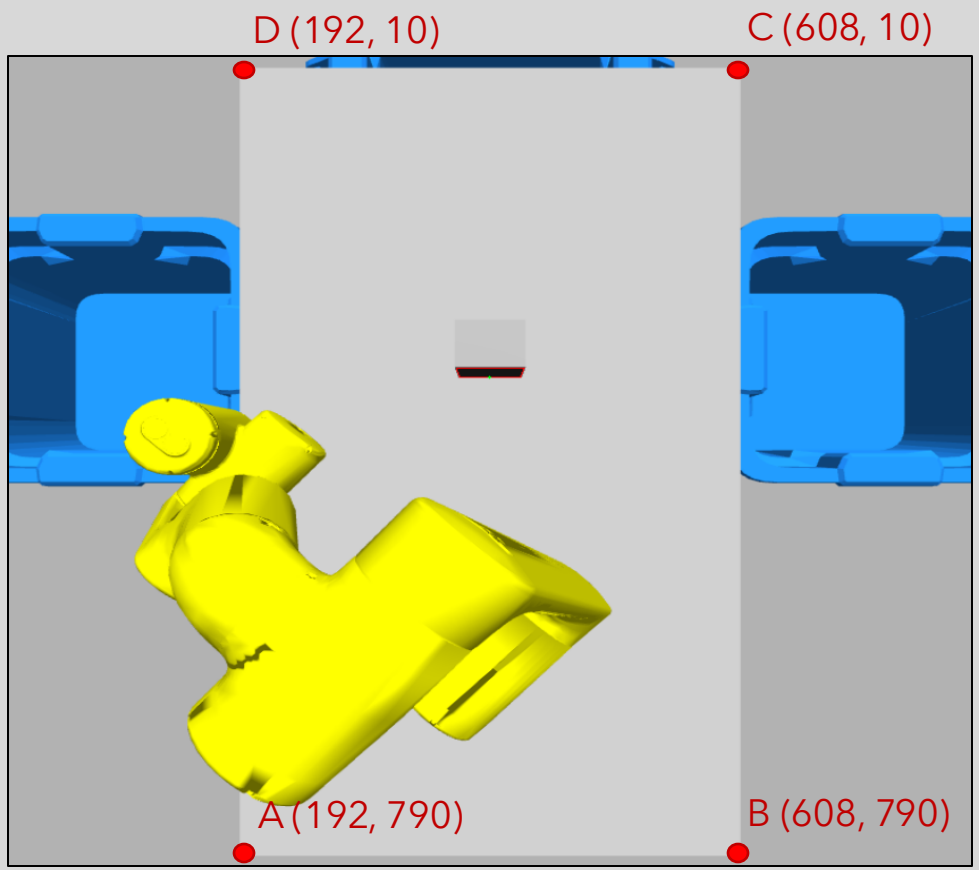
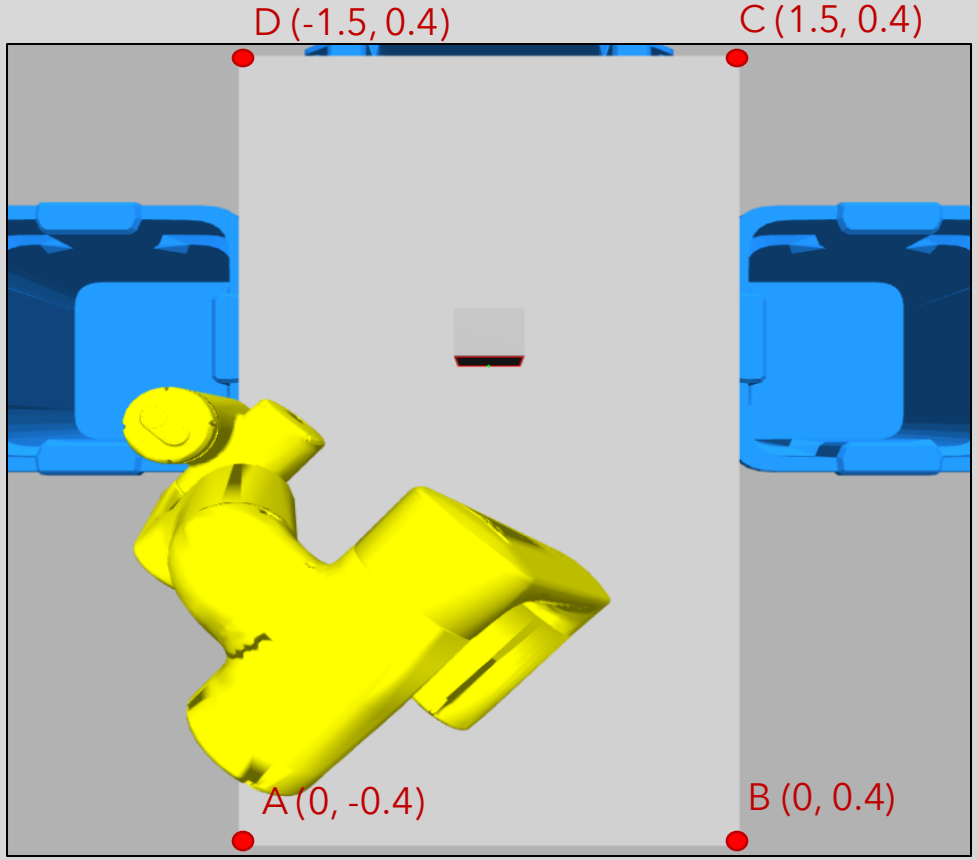


Image Points (Pixel)

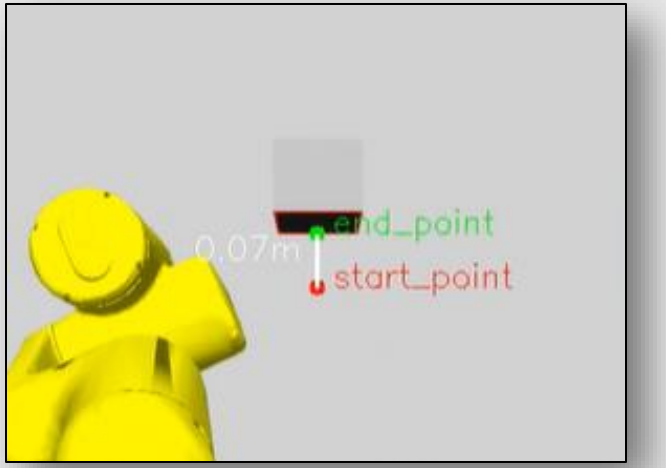
Mapping



World Points (m)

Homography Estimation

- We use four pairs of points `(image_point, world_point)` to find the homography matrix H (`cv2.findHomography`)
- H is then normalized by dividing each element by h_{33}
- We can then obtain the push point in real-world coordinates:
- $$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

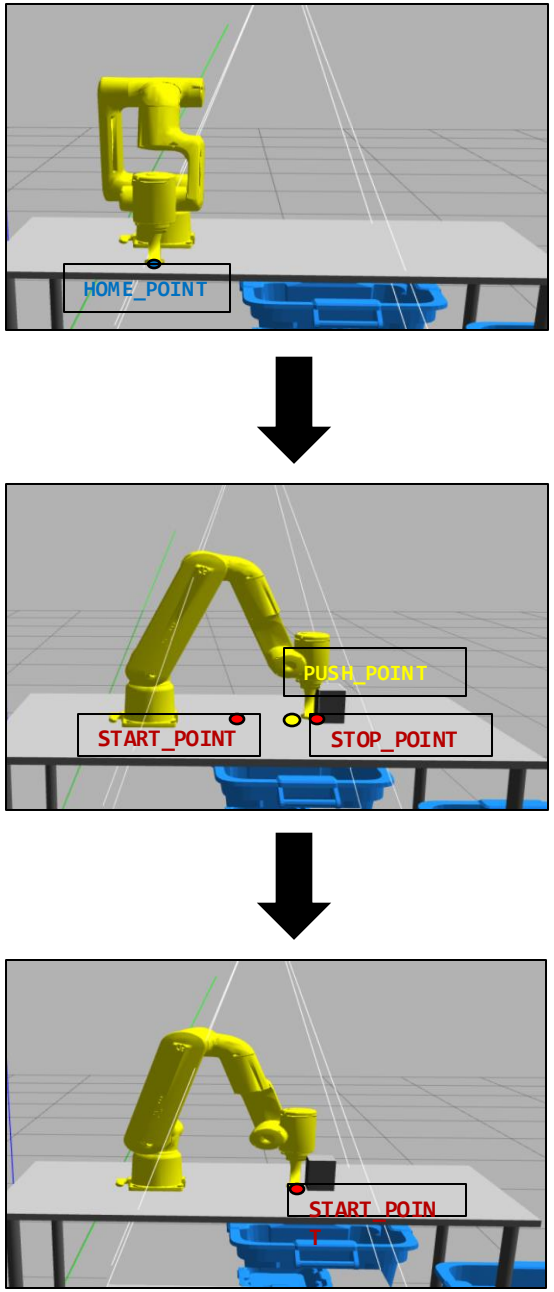


2. Sensing

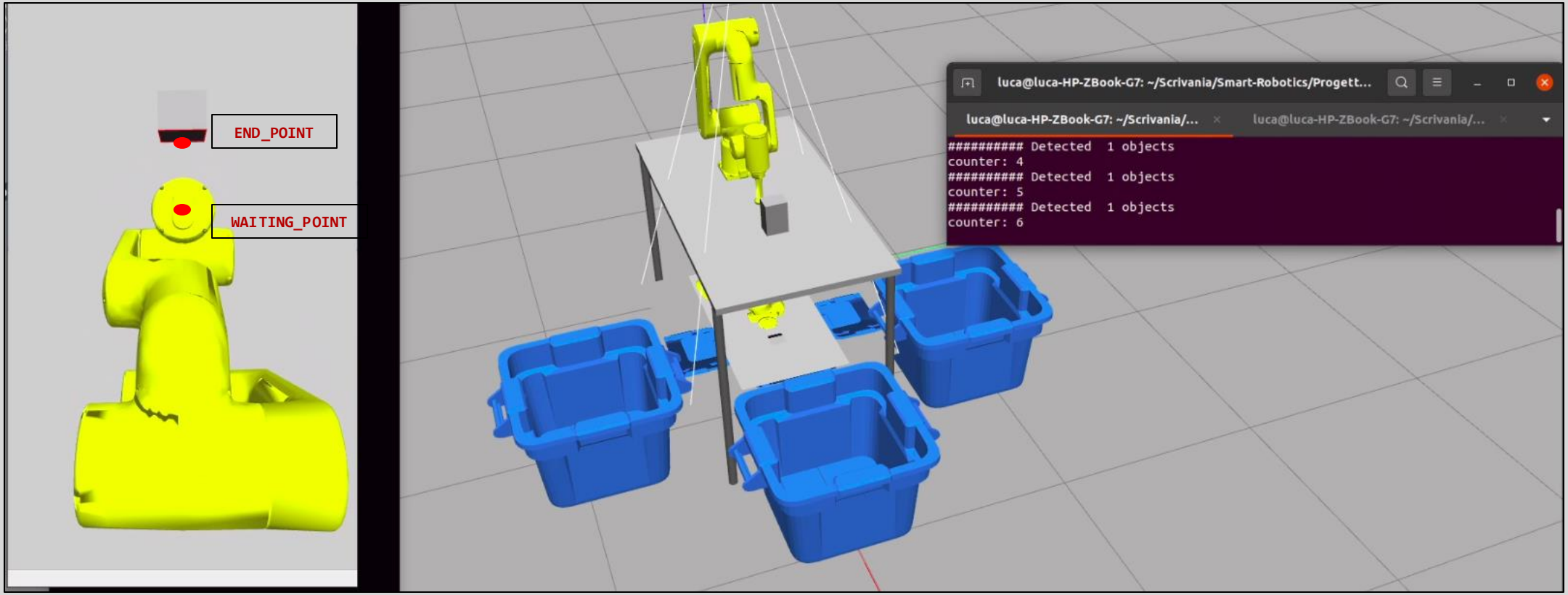

Initial Push Interaction
Force Measurement
Mass Classification

Initial Push Interaction

- Perform inertia measurement for each object
- Position Control Scheme (`position_controllers/JointTrajectoryController`)
- Waypoints:
  - `HOME_POINT` as default home position
  - `START_POINT` located 16 cm along the x-axis before reaching the `PUSH_POINT`
  - `PUSH_POINT` determined by the object detection module
  - `END_POINT` located 20 cm along the x-axis from `START_POINT` for a consistent and controlled final displacement



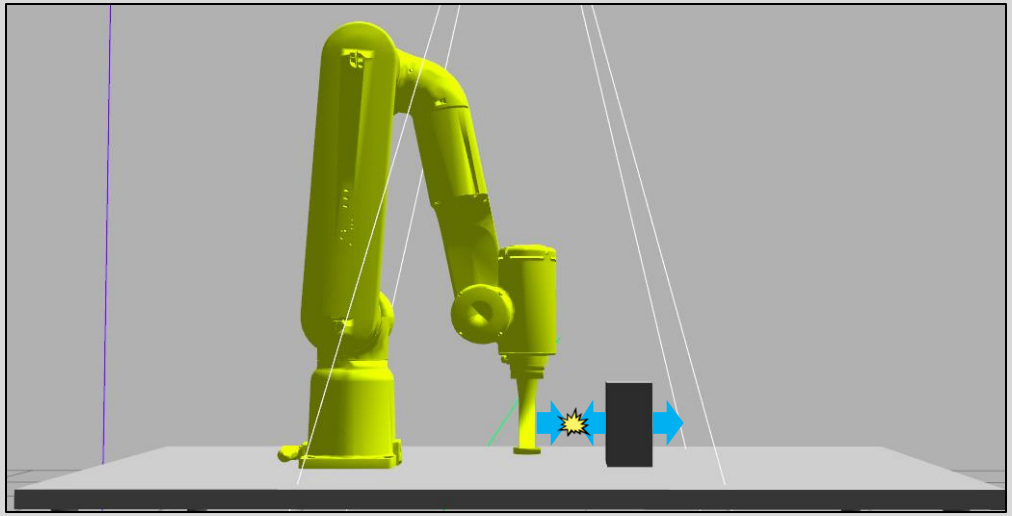
Waiting Point



Force Measurement

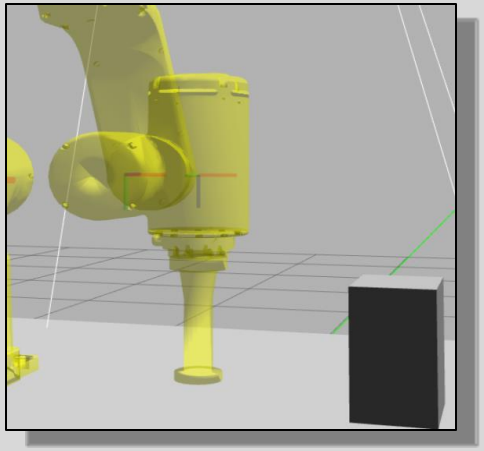
A 6D force-torque sensor on the wrist detects the forces applied on the end effector.

- **Newton's Second Law** (**$F = m \bullet a$**): proportional to the mass of the object
- **Newton's Third Law** (**$F_{End\ effector\ -\ Box} = F_{Box\ -\ End\ effector}$**): while performing the Position Control, the contact between the End effector and the box return a thrust to the sensor equal to the thrust needed for moving the box
- **Friction force** (**$f = \mu \bullet P_{Box}$**): proportional to the mass of the object



Mass/Force Classification

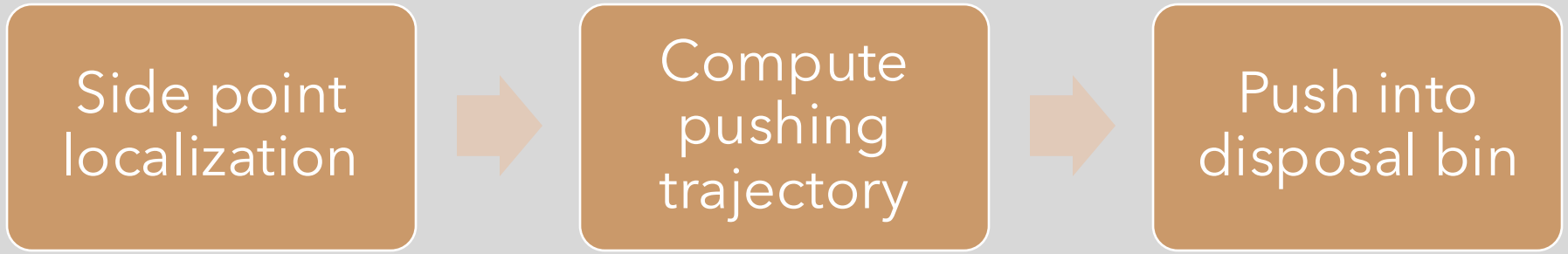
- Since the pushing trajectory for weight measurement occurs under **position control** and **involves a very small displacement** (approximately 20 cm), we assume **constant average acceleration** during the contact.
- Therefore we can **directly classify the force**.
- `force_sensor` node monitors the current position of the end effector, collecting **force measurements** along the x-axis from the sensor **when it enters the force measurement area**
- As soon as the end effector exits the measurement area, the node calculates the **average of the five highest force readings** to ensure robustness against potential outliers.



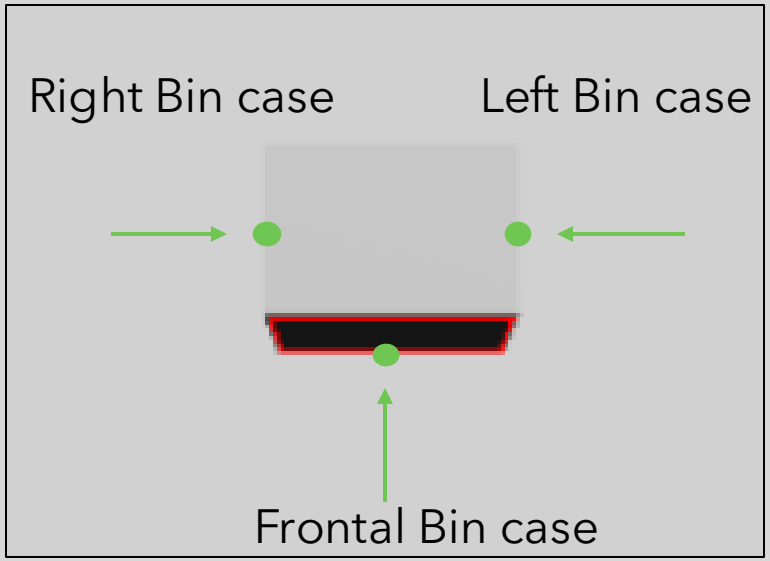
```
if average_top_5_force < THRESHOLD_EMPTY_LIGHT:
    box_type = "Empty Table"
elif THRESHOLD_EMPTY_LIGHT <= average_top_5_force < THRESHOLD_LIGHT_MEDIUM:
    box_type = "LIGHT BOX"
elif THRESHOLD_LIGHT_MEDIUM <= average_top_5_force < THRESHOLD_MEDIUM_HEAVY:
    box_type = "MEDIUM BOX"
else:
    box_type = "HEAVY BOX"
```

3. Sorting

To ensure that the box is pushed in the correct disposal bin, we need to perform the following steps:



Side point localization

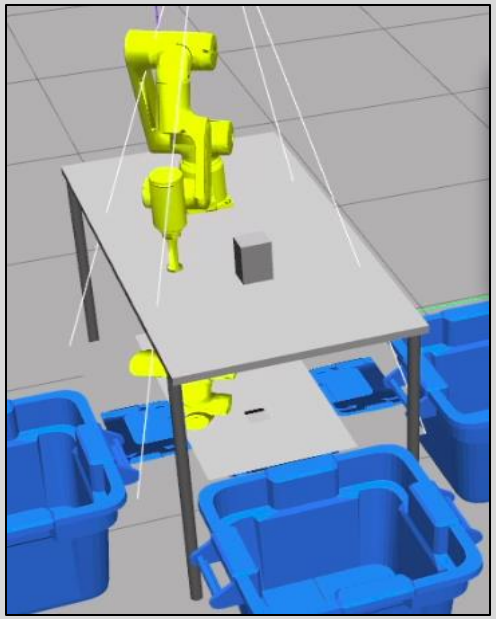


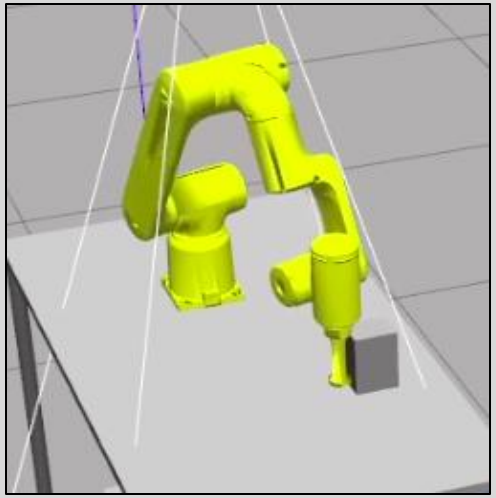
- Mapping of box type to disposal bin:
  - "LIGHT BOX" → frontal disposal bin
  - "MEDIUM BOX" → right disposal bin
  - "HEAVY BOX" → left disposal bin

- The `force_sensor` node publishes the `box_type` on the topic `box/type_topic`.
- The `object_detector` node reads this information, calculates the `side_point` (the point from which the robot should initiate the second push) based on the initial `push_point` and the distance the box has traveled due to the push, and then publishes it on the topic `box/side_point`.

Compute pushing trajectory

- When the trajectory planner node receives the `side_point`, then calculates the target disposal bin location and plans the robot's path accordingly.
- If the disposal bin is located in front, the robot simply moves in a straight line.
- However, if the disposal bin is on the left or right side, the planner must generate an additional waypoint to ensure the end effector reaches the `side_point` without colliding with the box.

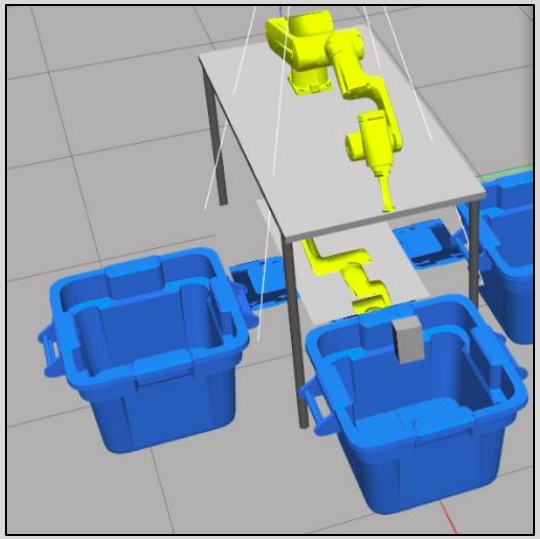


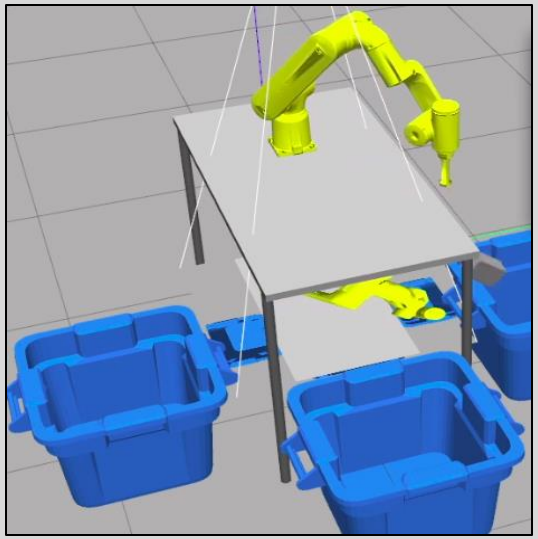


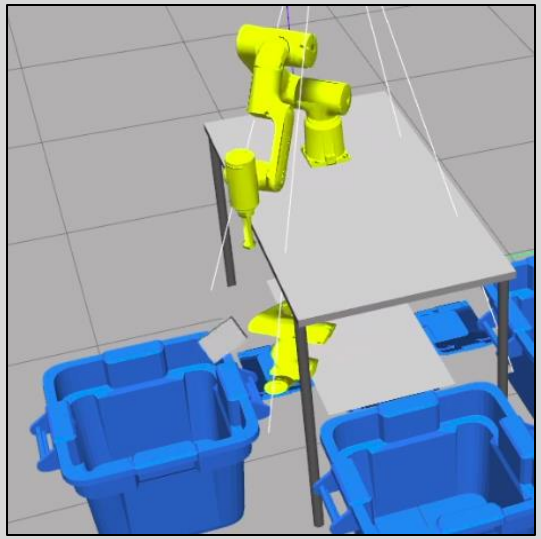
Push into disposal bin

Once the trajectory is defined, the robot pushes the box into the correct disposal bin. At this point, it is detected that there are no more boxes on the worktable, so_

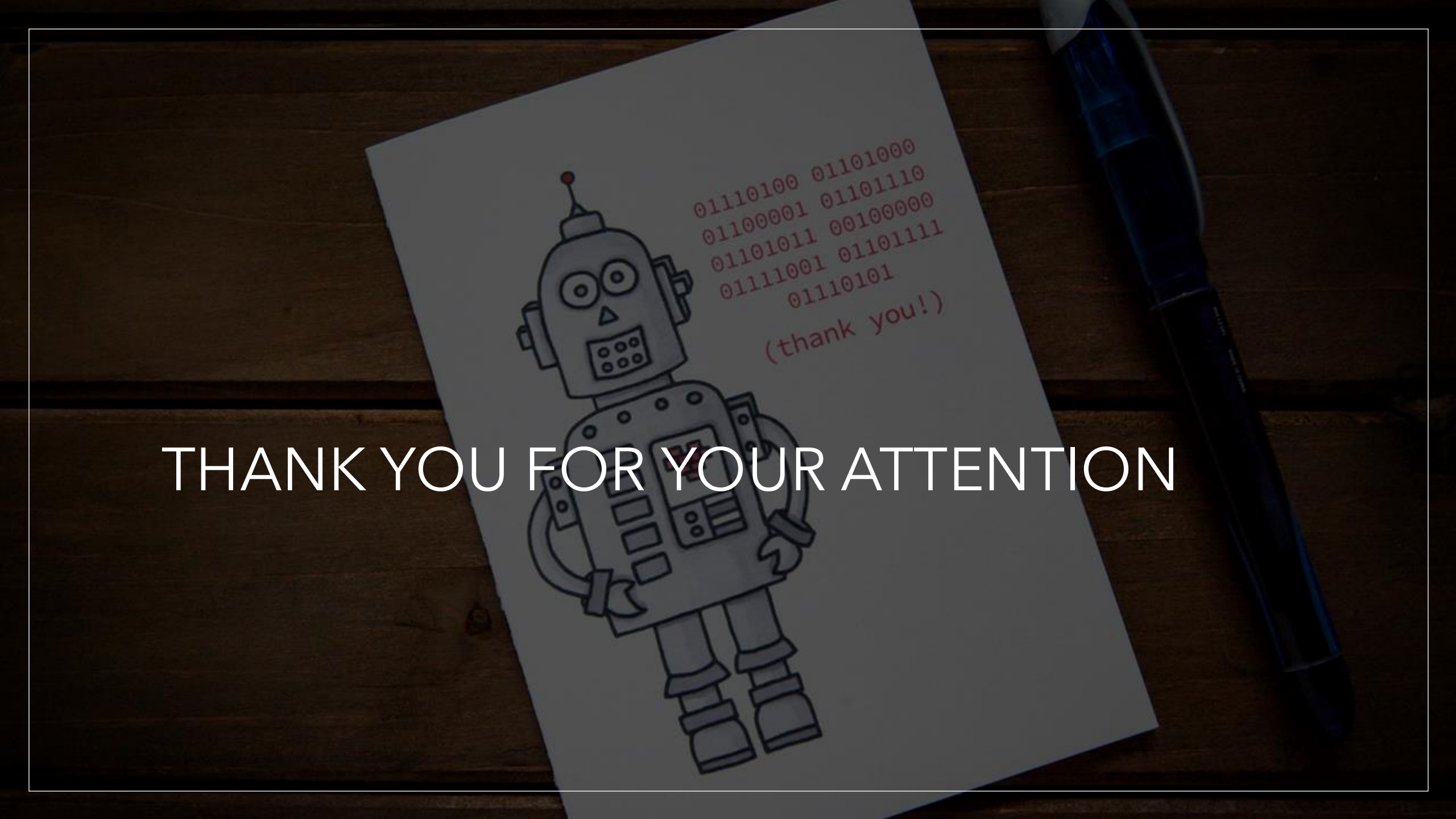
- the arm return to `HOME_POINT`;
- the `box_spawner` generates a new one.







DEMO TIME



THANK YOU FOR YOUR ATTENTION