**Assignment-4   
Yashaswi Ranga Sivaraju**

**Part-C**

**Flow of the Program**

1. **Initialization (\_\_init\_\_)**:
   * Sets up color ranges for detecting different colors
   * Initializes ROS topic subscribers and publishers for camera images and detected objects and obtain camera parameters for spatial computations along with setting up corresponding transformations.
2. **Receiving an Image Message (image\_callback)**:
   * Triggered when a new image is received from the **/camera/color/image\_raw** topic. Converts the ROS image message to an OpenCV format for processing.
   * Applies color thresholding to segment objects based on predefined color ranges. Detects contours, computes object properties, and identifies object positions. Updates detected objects and publishes their information.
3. **Visualization and Output**:
   * Draws bounding boxes and labels on the objects detected in the image for visualization. Displays the processed image with annotations.
   * Publishes detected objects with their detailed information to another ROS topic.

**Description of Functions**

* **\_\_init\_\_**:
  + Initializes the object detector, setting up ROS subscribers and publishers, and loads the camera model. It configures basic parameters like color ranges and transformations used for image processing.
* **get\_image\_dimensions**:
  + Fetches the dimensions of the latest image captured by the camera. This is useful for operations that require knowing image size, such as pixel access and image processing limits.
* **get\_camera\_homogeneous\_tranforms**:
  + Computes the transformation matrices between the camera and the world frames. Essential for converting coordinates from camera-centric to world-centric, which is crucial for positioning and robotics tasks.
* **get\_depth\_image** and **get\_color\_image**:
  + These methods fetch the latest depth and color images from the camera. Depth images are vital for calculating the real-world distance of detected objects, while color images are used for object detection and visualization.
* **get\_pinhole\_camera\_model**:
  + Sets up a camera model based on intrinsic parameters. This model is used to project points from the image plane into 3D space and vice versa, facilitating accurate spatial analysis.
* **get\_model\_position\_from\_gazebo**:
  + Retrieves the position of a specified model from Gazebo, aiding in understanding the scene's layout and the relative positions of objects within it.
* **get\_mask**:
  + Creates a binary mask for a specified color. This mask is used to isolate parts of the image that match the color, which is the first step in object segmentation.
* **get\_workbench\_depth**:
  + Determines the depth of the workbench, which is used as a reference plane for measuring object heights and for placing objects accurately in a spatial context.
* **get\_pixel\_depth**:
  + Retrieves the depth value from the depth image at specified pixel coordinates, essential for depth measurements of detected objects.
* **compute\_mass\_center**:
  + Calculates the center of mass of detected objects based on their contour data. This center is used for positioning and further analysis.
* **get\_detected\_objects**:
  + Processes image contours to extract meaningful object data such as dimensions, positions, and estimated physical characteristics.
* **get\_3D\_point\_from\_pixel** and **get\_pixel\_from\_3D\_point**:
  + Convert between 2D image coordinates and 3D world coordinates. These functions are pivotal for tasks that require understanding the spatial arrangement and interaction of objects.
* **convert\_point\_from\_camera\_to\_world** and **convert\_point\_from\_world\_to\_camera**:
  + Perform coordinate transformations between the camera frame and the world frame. These are critical for accurate localization and mapping in robotics and augmented reality applications.
* **image\_callback**:
  + The main function triggered by incoming image messages. It handles the conversion of images, object detection, visualization, and data publication.
* **publish\_detected\_objects**:
  + Gathers and sends information about detected objects to other components or systems, facilitating integrated robotic or automated responses.

**Part-D**

**States Defined in the State Machine:**

* **Home** : The initial state where the robot starts and to which it returns after handling an object.
* **Selecting Object** : The state where the robot selects an object from the workbench.
* **Picking and Placing**: The state where the robot physically moves the object from the workbench to the appropriate bin.
* **Done (done)**: A final state indicating that no more objects are left to process or the task is completed.

**Transitions Between States:**

* **select\_object**: Transition from **home** to **selecting\_object** if objects are detected, otherwise to **done**.
* **pick\_object**: Transition from **selecting\_object** to **picking\_and\_placing** once an object is selected.
* **get\_ready**: Transition from **picking\_and\_placing** back to **home** after placing the object, indicating readiness for the next cycle or completion.

**Methods and Their Functions:**

* **\_\_init\_\_ (Constructor)**: Initializes the state machine, sets up the controller for robot operations, and logs the start of the process. It also triggers the state machine's operation.
* **are\_objects\_detected**: A guard function that checks if there are objects on the workbench to determine the next state transition from **home**.
* **on\_enter\_home**: Actions performed when entering the **home** state, including moving the robot to a neutral/home position and deciding the next action based on object availability.
* **on\_enter\_selecting\_object**: Actions performed when entering the **selecting\_object** state, such as selecting an object to pick. This method updates the state to reflect that an object has been selected.
* **on\_enter\_picking\_and\_placing**: Actions performed when entering the **picking\_and\_placing** state, involving the actual movement of the selected object to its designated bin.
* **on\_enter\_done**: Actions performed when entering the **done** state, indicating task completion and shutting down the operation.

**Utility Function:**

* **create\_state\_machine\_graph**: Generates and saves a visual representation of the state machine's structure, useful for documentation or debugging.

**Execution:**

* The script initializes the ROS node and potentially creates a state machine graph for visualization. It then sets up the controller for the robot, initiates the state machine, and enters the ROS event loop with **rospy.spin()** to keep the node active.