Table 4: Scene-01: Theater

Category	Scene	Code	OD	WP	Path& Exec
Generic	Drive to the helmet	Pass	Pass	Pass	Pass
Generic	Navigate to the mop	Pass	Pass	Pass	Pass
Generic	Go to the vacuum	Pass	Fail	Fail	Fail
Generic	Walk to the table	Pass	Pass	Pass	Pass
Generic	Go to the fire extinguisher	Pass	Pass	Pass	Pass
Generic	Walk to the saw	Pass	Fail	Fail	Fail
Specific	Go to the red backpack	Pass	Pass	Pass	Pass
Specific	Move to the orange bucket	Pass	Pass	Pass	Pass
Specific	Drive to the chair on the right	Pass	Pass	Pass	Fail
Specific	Walk to the white fan	Pass	Pass	Pass	Pass
Specific	Navigate to the black fan	Pass	Pass	Pass	Pass
Specific	Navigate to the black speaker	Pass	Pass	Pass	Pass
Specific	Run to the red bag	Pass	Pass	Fail	Fail
Specific	Move to the black chair infront of you	Pass	Pass	Pass	Pass
Specific	Walk to the blue chair	Pass	Pass	Pass	Pass
Relational	Go to the chair with the black backpack on it	Pass	Pass	Pass	Pass
Relational	Go to the chair with the helmet on it	Pass	Pass	Pass	Pass
Relational	Go to the backpack next to the chair	Pass	Fail	Fail	Fail
Relational	Go to the backpack infront of the ladder	Pass	Pass	Pass	Pass
Relational	Run to the person on the ladder	Pass	Pass	Pass	Pass
Relational	Drive to the man sitting on the table	Pass	Pass	Pass	Pass
Relational	Go to the third chair from the left	Pass	Pass	Pass	Pass
Contextual	Go to something that can carry water	Pass	Pass	Pass	Pass
Contextual	Navigate to the somethings that can wash the floor	Fail	Fail	Fail	Fail
Contextual	My floor is dirty. Go to something that can fix it	Fail	Fail	Fail	Fail
Contextual	Go somewhere that I throw something in	Pass	Pass	Pass	Pass
Contextual	Walk to something that put out the fire	Pass	Fail	Fail	Fail
Contextual	Go to something that can cool me down	Pass	Fail	Fail	Fail
Contextual	Go to something white that can cool me down	Fail	Fail	Fail	Fail
Contextual	I am in the mood to listen to music. Go to something that can do that	Pass	Fail	Fail	Fail

6 Appendix

6.1 Navigation Prompts

The framework is tested on 4 different real-world environments. Further details about each of the environments are listed below:

- **Theater**, an indoor environment simulating a stage-like environment during set construction.
- **Lobby**, an indoor environment, mostly occupied with chairs and tables, posing more challenges in distinguishing specific objects and planning
- **Outdoor**, a larger environment with objects such as trees, cars, bikes, and sporting areas.
- **Courtyard**, an outdoor environment tested at night to challenge the framework with low-light images.

Along with the general objects present in the scene, we placed some additional items like backpacks, shoes, and cones into the scene. All the prompts tested in these 4 different environments are presented in Tables 4, 5, 6, and 7. For each of the prompts, we also highlighted whether each step succeeded or not.

Table 5: Scene-02: Lobby

Category	Scene	Code	OD	WP	Path& Exec
Generic	Run to the trash bag	Pass	Pass	Pass	Pass
Generic	Navigate to the water fountain	Pass	Fail	Fail	Fail
Generic	Drive to the broom	Pass	Pass	Pass	Pass
Generic	Walk to monitor	Pass	Pass	Fail	Fail
Specific	Walk to the tiny monitor	Fail	Fail	Fail	Fail
Specific	Walk to the smallest monitor	Pass	Pass	Pass	Pass
Relational	Walk to the right most cone	Pass	Pass	Pass	Pass
Relational	Walk to the table with the can on it	Pass	Fail	Fail	Fail
Relational	Walk to the table next to the red backpack	Pass	Pass	Pass	Pass
Relational	Go the blue chair with the backpack on it	Pass	Pass	Pass	Pass
Relational	Walk to the leftmost table	Pass	Fail	Fail	Fail
Relational	Go to backpack closest to the shoe	Pass	Pass	Pass	Pass
Relational	Walk to shoe next to the red backpack	Pass	Pass	Pass	Pass
Relational	Remove the trash from the floor	Fail	Fail	Fail	Fail
Relational	Walk to the table with the monitor	Fail	Fail	Fail	Fail
Relational	Drive to the closest monitor to the table	Pass	Pass	Pass	Pass
Relational	Go to the table with the bottle on it	Fail	Fail	Fail	Fail
Relational	Go to the bottle on top of the table	Pass	Pass	Pass	Pass
Relational	Go to the TV closest to the person	Pass	Pass	Pass	Pass
Relational	Go to the person wearing a blue shirt	Pass	Fail	Fail	Fail
Relational	Run to to the chair with the blue coat	Pass	Fail	Fail	Fail
Relational	Move to the backpack on the chair	Fail	Fail	Fail	Fail
Relational	Move to the black backpack on the chair	Fail	Fail	Fail	Fail
Relational	Drive to the chair with the backpack on it that is not red	Fail	Fail	Fail	Fail
Contextual	Find something that can help firefighters	Pass	Pass	Pass	Pass
Contextual	Go to something that can clean the dirty floor	Fail	Fail	Fail	Fail
Contextual	I am thirsty. Walk to somewhere this can be fixed	Fail	Fail	Fail	Fail
Contextual	Go to a place where I can watch a movie	Pass	Pass	Pass	Pass
Contextual	Drive to a place where I can watch a video	Fail	Fail	Fail	Fail

All the prompts used in classroom environment for testing the two different input representations A & B are presented in Table 8.

Table 6: Scene-03: Outdoor

Category	Scene	Code	OD	WP	Path& Exec
Generic	Wander to fire hydrant	Pass	Pass	Pass	Pass
Generic	Step towards the grill	Pass	Fail	Fail	Fail
Generic	Walk to the skateboard	Pass	Pass	Pass	Pass
Generic	Walk to the bike	Pass	Pass	Fail	Fail
Generic	Walk to the bike	Pass	Pass	Pass	Pass
Generic	Go to bike rack	Pass	Pass	Pass	Pass
Generic	Go to the sign	Pass	Pass	Fail	Fail
Generic	Go to the bench	Pass	Pass	Pass	Pass
Specific	Sashay to the stop sign	Pass	Pass	Pass	Pass
Specific	Go to the basketball hoop	Pass	Pass	Pass	Pass
Specific	Roam towards the blue car	Pass	Pass	Pass	Pass
Specific	Trot towards the red bag	Pass	Pass	Pass	Pass
Specific	Go to the red object generically	Fail	Fail	Fail	Fail
Relational	Proceed to the middle cone	Pass	Pass	Pass	Pass
Relational	Journey to the tree next to the backpack	Fail	Fail	Fail	Fail
Relational	Trek to the backpack by the tree	Pass	Pass	Pass	Pass
Contextual	A firefighter needs water. Walk to a source of water	Pass	Fail	Fail	Fail
Contextual	Head towards something that can help firefighters	Pass	Pass	Pass	Pass
Contextual	You are a dog that needs to mark its territory. Go to a place that can do this	Pass	Pass	Pass	Pass
Contextual	You are carrying trash, Find somewhere to dump	Pass	Pass	Pass	Pass
Contextual	Find me something to do a kick flip on	Pass	Pass	Pass	Pass
Contextual	I want to shoot some hoops. Take me there	Fail	Fail	Fail	Fail
Contextual	Move to a faster mode of transportation	Pass	Pass	Pass	Pass
Contextual	Head to the fastest mode of transportation	Pass	Pass	Pass	Pass

Table 7: Scene-04: Courtyard

Category	Scene	Code	OD	WP	Path& Exec
Generic	Run to the door	Pass	Pass	Pass	Pass
Generic	Run towards the backpack	Pass	Pass	Pass	Pass
Generic	Drive to the wagon	Pass	Pass	Pass	Pass
Generic	Navigate to the stairs	Pass	Pass	Pass	Pass
Specific	Proceed towards the garbage can on the right	Pass	Pass	Pass	Pass
Specific	Stroll to the recycle bin on the left	Pass	Pass	Fail	Fail
Specific	Sprint to the picnic table	Pass	Pass	Pass	Pass
Relational	Go to the bench with water container	Pass	Pass	Pass	Pass
Relational	Walk to the bench with nothing on it	Pass	Pass	Pass	Pass
Relational	Proceed to the bench with most objects on it	Fail	Fail	Fail	Fail
Relational	Move towards the backpack farthest from the bench	Pass	Fail	Fail	Fail
Relational	Head towards the middle cone	Fail	Fail	Fail	Fail
Relational	Head towards to middle cone in the row of cones	Pass	Pass	Pass	Pass
Relational	Go to the table with only 1 chair	Fail	Fail	Fail	Fail
Relational	Go to the table with only 1 chair. There are multiple groups of chairs multiple tables	Pass	Pass	Pass	Pass
Relational	Step towards the column closest to the cart	Pass	Pass	Pass	Pass
Relational	Move to the largest group of benches	Pass	Pass	Pass	Pass
Relational	Walk towards the table with the umbrella	Fail	Fail	Fail	Fail
Relational	Walk towards the table with the umbrella	Pass	Pass	Pass	Pass
Relational	Drive to the table without any chairs	Pass	Pass	Pass	Pass
Relational	Walk to black table with 6 chairs	Fail	Fail	Fail	Fail
Relational	Walk to the table with most chairs	Pass	Pass	Pass	Pass
Relational	Navigate to the table with the backpack	Fail	Fail	Fail	Fail
Contextual	Go to nearest entrance to the building	Fail	Fail	Fail	Fail
Contextual	Go to something that you would hold open for someone elderly	Pass	Fail	Fail	Fail
Contextual	Go to something that will make it easy to carry heavy luggage	Pass	Fail	Fail	Fail
Contextual	Go to somewhere I can eat my lunch	Pass	Fail	Fail	Fail
Contextual	Go up to the second floor	Fail	Fail	Fail	Fail
Contextual	Go find something to climb	Fail	Fail	Fail	Fail
Contextual	Run upstairs	Pass	Pass	Pass	Pass
Contextual	Find me somewhere to park my bike	Pass	Pass	Pass	Pass

Table 8: Scene-05: Classroom

Category	Sentence	A	B
Generic	Go to the backpack	Pass	Pass
Generic	Move towards the backpack	Pass	Pass
Generic	Drive to the backpack	Pass	Pass
Generic	Run towards the backpack	Pass	Pass
Generic	Go to the cone	Pass	Pass
Generic	Go to conical traffic delineator	Pass	Pass
Generic	Go to the trash can	Pass	Pass
Generic	Walk to the whiteboard	Pass	Pass
Generic	Proceed to the broom	Pass	Pass
Generic	Trek towards the wagon	Pass	Pass
Generic	Find paper towels	Pass	Pass
Generic	Go to the outlet	Pass	Pass
Specific	Go to the red backpack	Pass	Pass
Specific	Go to the black backpack	Pass	Pass
Specific	Navigate to the backpack on the left	Pass	Pass
Specific	Drive to the backpack on the right	Pass	Pass
Specific	Go to the whiteboard in front of you	Pass	Pass
Specific	Move to the whiteboard on your right	Pass	Pass
Specific	Move to the whiteboard on the right	Pass	Fail
Specific	Go to the backpack on your right	Pass	Pass
Specific	Walk to the backpack on the left	Pass	Fail
Specific	Go to the leftmost backpack on the right	Pass	Fail
Specific	Go to the orange cone on your right	Fail	Pass
Specific	Go to the middle outlet	Pass	Fail
Relational	Go to backpack to the right of the red backpack	Pass	Pass
Relational	Drive to the backpack that is to the left of the black backpack	Pass	Fail
Relational	Walk to the bag that is next to the black bag	Fail	Fail
Relational	Move towards the backpack under the whiteboard	Pass	Pass
Relational	Walk to the backpack on the chair	Pass	Pass
Relational	Go to the chair with the backpack	Pass	Fail
Relational	Walk to the backpack on top of the chair	Pass	Fail
Relational	Run to the rightmost backpack	Pass	Pass
Relational	Walk to the leftmost backpack	Pass	Pass
Relational	Go to the middle chair	Pass	Fail
Relational	Go to the leftmost backpack on your right	Fail	Pass
Relational	Go to the middle chair in the row of chairs	Pass	Pass
Relational	Go to the backpack to the left of the cone	Pass	Fail
Relational	Go the cone to the left of the backpack	Pass	Pass
Relational	Go to the second chair from the left	Pass	Fail
Contextual	Go to somewhere I can sit down	Pass	Pass
Contextual	Find a place for me to rest	Pass	Fail
Contextual	Go to somewhere I can speak from	Pass	Fail
Contextual	Find a place to store cleaning supplies	Pass	Fail
Contextual	Find me something to write on	Pass	Pass
Contextual	My friend has question. Go to somewhere you can explain the answer to him	Fail	Fail
Contextual	I spilled a lot of sand. Find me something to pick up my mess	Pass	Pass
Contextual	Walk to something I can put my laptop in	Fail	Fail
Contextual	I spilled water. Find me something to clean this up	Pass	Fail
Contextual	Go to somewhere I can google something	Pass	Pass
Contextual	Go to somewhere I can charge my phone	Pass	Pass

6.2 Code Prompt

Navigation Prompt for the concatenated single image input. We leverage the original prompts presented in [14].

```
import math

class ImagePatch:
    """A Python class containing a crop of an image centered around a
        particular object, as well as relevant information.
    Attributes
    ----------
    cropped_image : array_like
        An array-like of the cropped image taken from the original image.
    left, lower, right, upper : int
        An int describing the position of the (left/lower/right/upper)
            border of the crop's bounding box in the original image.
    frame: name of camera frame

    Methods
    -------
    find(object_name: str)->List[ImagePatch]
        Returns a list of new ImagePatch objects containing crops of the
            image centered around any objects found in the
        image matching the object_name.
    exists(object_name: str)->bool
        Returns True if the object specified by object_name is found in the
            image, and False otherwise.
    verify_property(property: str)->bool
        Returns True if the property is met, and False otherwise.
    best_text_match(option_list: List[str], prefix: str)->str
        Returns the string that best matches the image.
    simple_query(question: str=None)->str
        Returns the answer to a basic question asked about the image. If no
            question is provided, returns the answer to "What is this?".
    llm_query(question: str, long_answer: bool)->str
        References a large language model (e.g., GPT) to produce a response
            to the given question. Default is short-form answers, can be
            made long-form responses with the long_answer flag.
    compute_depth()->float
        Returns the median depth of the image crop.
    crop(left: int, lower: int, right: int, upper: int)->ImagePatch
        Returns a new ImagePatch object containing a crop of the image at
            the given coordinates.
    """

    def __init__(self, image, left: int = None, lower: int = None, right:
        int = None, upper: int = None, frame = None):
        """Initializes an ImagePatch object by cropping the image at the
            given coordinates and stores the coordinates as
        attributes. If no coordinates are provided, the image is left
            unmodified, and the coordinates are set to the
        dimensions of the image.
        Parameters
        -------
        image : array_like
            An array-like of the original image.
        left, lower, right, upper : int
```

```
            An int describing the position of the (left/lower/right/upper)
                border of the crop's bounding box in the original image.
        """
        if left is None and right is None and upper is None and lower is
            None:
            self.cropped_image = image
            self.left = 0
            self.lower = 0
            self.right = image.shape[2] # width
            self.upper = image.shape[1] # height
        else:
            self.cropped_image = image[:, lower:upper, left:right]
            self.left = left
            self.upper = upper
            self.right = right
            self.lower = lower

        self.width = self.cropped_image.shape[2]
        self.height = self.cropped_image.shape[1]

        self.horizontal_center = (self.left + self.right) / 2
        self.vertical_center = (self.lower + self.upper) / 2

        self.frame = frame

    def find(self, object_name: str) -> List[ImagePatch]:
        """Returns a list of ImagePatch objects matching object_name
            contained in the crop if any are found.
        Otherwise, returns an empty list.
        Parameters
        ----------
        object_name : str
            the name of the object to be found

        Returns
        -------
        List[ImagePatch]
            a list of ImagePatch objects matching object_name contained in
                the crop

        Examples
        --------
        >>> # return the foo
        >>> def execute_command(image) -> List[ImagePatch]:
        >>> image_patch = ImagePatch(image)
        >>> foo_patches = image_patch.find("foo")
        >>> return foo_patches
        """
        return find_in_image(self.cropped_image, object_name)

    def exists(self, object_name: str) -> bool:
        """Returns True if the object specified by object_name is found in
            the image, and False otherwise.
        Parameters
        -------
        object_name : str
            A string describing the name of the object to be found in the
                image.

```

```
        Examples
        -------
        >>> # Are there both foos and garply bars in the photo?
        >>> def execute_command(image)->str:
        >>> image_patch = ImagePatch(image)
        >>> is_foo = image_patch.exists("foo")
        >>> is_garply_bar = image_patch.exists("garply bar")
        >>> return bool_to_yesno(is_foo and is_garply_bar)
        """
        return len(self.find(object_name)) > 0

    def verify_property(self, object_name: str, visual_property: str) ->
        bool:
        """Returns True if the object possesses the visual property, and
            False otherwise.
        Differs from 'exists' in that it presupposes the existence of the
            object specified by object_name, instead checking whether the
            object possesses the property.
        Parameters
        -------
        object_name : str
            A string describing the name of the object to be found in the
                image.
        visual_property : str
            A string describing the simple visual property (e.g., color,
                shape, material) to be checked.

        Examples
        -------
        >>> # Do the letters have blue color?
        >>> def execute_command(image) -> str:
        >>> image_patch = ImagePatch(image)
        >>> letters_patches = image_patch.find("letters")
        >>> # Question assumes only one letter patch
        >>> return bool_to_yesno(letters_patches[0].verify_property("letters
            ", "blue"))
        """
        return verify_property(self.cropped_image, object_name, property)

    def best_text_match(self, option_list: List[str]) -> str:
        """Returns the string that best matches the image.
        Parameters
        -------
        option_list : str
            A list with the names of the different options
        prefix : str
            A string with the prefixes to append to the options

        Examples
        -------
        >>> # Is the foo gold or white?
        >>> def execute_command(image)->str:
        >>> image_patch = ImagePatch(image)
        >>> foo_patches = image_patch.find("foo")
        >>> # Question assumes one foo patch
        >>> return foo_patches[0].best_text_match(["gold", "white"])
        """
        return best_text_match(self.cropped_image, option_list)
```

```
    def simple_query(self, question: str = None) -> str:
        """Returns the answer to a basic question asked about the image. If
            no question is provided, returns the answer
        to "What is this?". The questions are about basic perception, and
            are not meant to be used for complex reasoning
        or external knowledge.
        Parameters
        -------
        question : str
            A string describing the question to be asked.

        Examples
        -------

        >>> # Which kind of baz is not fredding?
        >>> def execute_command(image) -> str:
        >>> image_patch = ImagePatch(image)
        >>> baz_patches = image_patch.find("baz")
        >>> for baz_patch in baz_patches:
        >>> if not baz_patch.verify_property("baz", "fredding"):
        >>> return baz_patch.simple_query("What is this baz?")

        >>> # What color is the foo?
        >>> def execute_command(image) -> str:
        >>> image_patch = ImagePatch(image)
        >>> foo_patches = image_patch.find("foo")
        >>> foo_patch = foo_patches[0]
        >>> return foo_patch.simple_query("What is the color?")

        >>> # Is the second bar from the left quuxy?
        >>> def execute_command(image) -> str:
        >>> image_patch = ImagePatch(image)
        >>> bar_patches = image_patch.find("bar")
        >>> bar_patches.sort(key=lambda x: x.horizontal_center)
        >>> bar_patch = bar_patches[1]
        >>> return bar_patch.simple_query("Is the bar quuxy?")
        """
        return simple_query(self.cropped_image, question)

    def compute_depth(self):
        """Returns the median depth of the image crop
        Parameters
        ----------
        Returns
        -------
        float
            the median depth of the image crop

        Examples
        --------
        >>> # the bar furthest away
        >>> def execute_command(image)->ImagePatch:
        >>> image_patch = ImagePatch(image)
        >>> bar_patches = image_patch.find("bar")
        >>> bar_patches.sort(key=lambda bar: bar.compute_depth())
        >>> return bar_patches[-1]
        """
        depth_map = compute_depth(self.cropped_image)
        return depth_map.median()
```

```
672
673     def crop(self, left: int, lower: int, right: int, upper: int) ->
674         ImagePatch:
675         """Returns a new ImagePatch cropped from the current ImagePatch.
676         Parameters
677         -------
678         left, lower, right, upper : int
679             The (left/lower/right/upper)most pixel of the cropped image.
680         -------
681         """
682         return ImagePatch(self.cropped_image, left, lower, right, upper,
683             self.frame)
684
685     def overlaps_with(self, left, lower, right, upper):
686         """Returns True if a crop with the given coordinates overlaps with
687             this one,
688         else False.
689         Parameters
690         ----------
691         left, lower, right, upper : int
692             the (left/lower/right/upper) border of the crop to be checked
693
694         Returns
695         -------
696         bool
697             True if a crop with the given coordinates overlaps with this one
698                 , else False
699
700         Examples
701         --------
702         >>> # black foo on top of the qux
703         >>> def execute_command(image) -> ImagePatch:
704         >>> image_patch = ImagePatch(image)
705         >>> qux_patches = image_patch.find("qux")
706         >>> qux_patch = qux_patches[0]
707         >>> foo_patches = image_patch.find("black foo")
708         >>> for foo in foo_patches:
709         >>> if foo.vertical_center > qux_patch.vertical_center
710         >>> return foo
711         """
712         return self.left <= right and self.right >= left and self.lower <=
713             upper and self.upper >= lower
714
715 def best_image_match(list_patches: List[ImagePatch], content: List[str],
716     return_index=False) -> Union[ImagePatch, int]:
717     """Returns the patch most likely to contain the content.
718     Parameters
719     ----------
720     list_patches : List[ImagePatch]
721     content : List[str]
722         the object of interest
723     return_index : bool
724         if True, returns the index of the patch most likely to contain the
725             object
726
727     Returns
728     -------
729     int
730         Patch most likely to contain the object
```

```
    """
    return best_image_match(list_patches, content, return_index)


def distance(patch_a: ImagePatch, patch_b: ImagePatch) -> float:
    """
    Returns the distance between the edges of two ImagePatches. If the
        patches overlap, it returns a negative distance
    corresponding to the negative intersection over union.

    Parameters
    ----------
    patch_a : ImagePatch
    patch_b : ImagePatch

    Examples
    --------
    # Return the qux that is closest to the foo
    >>> def execute_command(image):
    >>> image_patch = ImagePatch(image)
    >>> qux_patches = image_patch.find('qux')
    >>> foo_patches = image_patch.find('foo')
    >>> foo_patch = foo_patches[0]
    >>> qux_patches.sort(key=lambda x: distance(x, foo_patch))
    >>> return qux_patches[0]
    """
    return distance(patch_a, patch_b)


def bool_to_yesno(bool_answer: bool) -> str:
    return "yes" if bool_answer else "no"


def coerce_to_numeric(string):
    """
    This function takes a string as input and returns a float after removing
         any non-numeric characters.
    If the input string contains a range (e.g. "10-15"), it returns the
        first value in the range.
    """
    return coerce_to_numeric(string)

### Nav Client
"""
Navigates an agent to an object in an imgae given its center coordinates
Parameters
    ----------
    x : x coordinate of the center of the object
    y : y coordinate of the center of the object
"""
 Examples
       -------
       >>> # Go to the blue foo.
       >>> def execute_command(image)
       >>> image_patch = ImagePatch(image)
       >>> foo_patches = image_patch.find("foo")
       >>> # Verify visual property
       >>> blue_color_patches = []
       >>> for foo_patch in foo_patches:
```

```
        >>> if verify_property(blue, "color")
        >>> blue_color_patches.append(foo_patch)
        >>> inputs = (blue_color_patches[0].horizontal_center,
            blue_color_patches[0].vertical_center)
        >>> return {'function': 'nav_function', 'inputs': inputs, 'box': [
            blue_color_patches[0].left, blue_color_patches[0].lower,
            blue_color_patches[0].right, blue_color_patches.upper]}
        """
def navigate_to_object(double x, double y):

"""



Write a function using Python and/or the ImagePatch class (above) that
    could be executed to provide an answer to the query by calling one of
    the functions in the Nav Client (navigate_to_object).
Note using the ImagePatch is not required for all queries. If the query is
    not relevant to navigation, return None for the function and a string
    describing the problem.

Consider the following guidelines:
- Use base Python (comparison, sorting) for basic logical operations, left/
    right/up/down, math, etc.
- Use the llm_query function to access external information and answer
    informational questions not concerning the image.
- All properties such as color should be verified using the verify_property
     function. "go to the blue foo" implies "go to foo if foo is blue"
- The output of this function should be a dict {function: 'function in the
    nav client', inputs: 'inputs to the chosen nav client function', box: [
    left, lower, right, upper]}, or for an error {function: 'None', error:
    'message describing the problem'}
- If more than one object is found pick the best match

Query: INSERT_QUERY_HERE
```