University of Bremen

German Research Center for Artificial Intelligence

Worksheet 05: Robot Perception

Deadline: 13.01.2023
Robot Design Lab
03-IBGA-FI-RDL
WiSe 22-23

Universität Bremen
FB 3 – Mathematik und Informatik
Arbeitsgruppe Robotik
Prof. Dr. Dr. h.c. Frank Kirchner
Dr. rer. nat. Teena Hassan
M. Sc. Mihaela Popescu

1 Pinhole Camera Model 15%

Let's assume that your robot has a camera that contains a lens with a focal length $f = 150$mm.
Now, we place the robot in a scene with three objects:

- Object A has a height of 30 cm and is at a distance of 20 meters from the optical center of the camera.
- Object B is at a distance of 35 meters from the optical center of the camera and makes a projection of 15 pixels on the focal plane.
- Object C is at an unknown distance and has an unknown height, but it generates a projection of 5 pixels on the focal plane.

Consider a conversion of one pixel equals to one millimeter, and assume that all objects are standing on the same 2D plane which is parallel to the optical axis. Hence, only distances to the camera matter.

1. What is the height of the projection (in millimeters) of object A on the focal plane? (5%)
2. What is the height (in meters) of object B in the real world? (5%)
3. What is the height of object C and what is its distance from the optical center of the camera? If you think this cannot be computed, what kind of information do you need to make this computation? (5%)

2 Quiz: Camera Calibration 15%

Camera calibration is an important task in computer vision. Please answer the following questions on camera calibration:

1. What is the role of camera calibration? (3%)
2. What is the procedure to calibrate the robot's camera? (3%)
3. What are the intrinsic camera parameters and what do they represent? (3%)
4. What are the extrinsic camera parameters and what do they represent? (3%)
5. What is the calibration matrix composed of? (3%)

3 Image Transformations 10%

In this task, you will familiarize yourself with some of the basic algorithms implemented in the OpenCV library. To install OpenCV for Python3, run the following command in a terminal:

```
sudo apt-get install python3-opencv
```

We have provided you a sample image named `turtlebot.jpg` in the `images` directory and a skeleton code named `image_transformations.py` in the `perception_scrips.zip` file. Create an empty directory called `results` where the output images will be saved. Please complete the skeleton code to perform the following image transformations:

1. Rotate the original image by 180°. (2%)
2. Change the original image to grayscale. (2%)

3. Extract edges from the original image using the Canny Edge Detection algorithm in OpenCV. Tune the filter parameters `minVal` and `maxVal` until you get clear edges. (2%)
4. Blur the original image with a convolution kernel size of (30, 30). (2%)
5. Add (blend) the original image and the rotated image with a weight of 0.5 each. (2%)

Hint: Feel free to use the official OpenCV documentation and look up the definition of the functions you need using the top right corner search bar: https://docs.opencv.org/4.x/d2/d96/tutorial_py_table_of_contents_imgproc.html.

For each of the above subtasks, **insert** the output images in your PDF solution. In addition, **submit** your completed source code file `image_transformations.py` with your PDF solution.

Note: For this task, you **do not** have to include or explain your code in your LaTeX file.

4 Circle Detection 15%

In this task, you will use the Hough Circle Transform [1] from the OpenCV library to detect circles. Implement your solution in the skeleton code file `circle_detection.py` from `perception_scrips.zip` file and use the provided image `shapes.png` to show your results. If needed, reuse the method you implemented in the first subtask for the next subtasks.

Please solve the following subtasks:

1. Implement the `detect_all_circles()` function to return the list of all circles in the image containing their center coordinates and radius. (5%)
   Hint: Use the Hough Circle Transform `cv2.HoughCircles()`.
2. Implement the `mark_circles_on_image()` function to draw a black cross marker in the center of each circle, draw the contour of the circle and save the image. (5%)
   Hint: Use `cv2.drawMarker()`, `cv2.MARKER_CROSS`, `thickness=3`, `cv2.circle()` and `cv2.imwrite()`.
3. Implement the `find_largest_circle()` function to return the center coordinates and the radius of the largest circle in the image. (5%)

Note: For this task, please **include** the relevant lines of code in your LaTeX file and briefly **explain** your implementation. In addition, please **submit** your completed source code file `circle_detection.py`, a screenshot of the terminal window showing the results of subtasks 1 and 3, as well as the image from subtask 2 along with your PDF solution.

5 Polygon Detection 15%

In this task, you will use a polygon approximation function [2] from the OpenCV library to detect polygons in the image, namely squares, rectangles and triangles. Implement your solution in the skeleton code file `polygon_detection.py` from the `perception_scrips.zip` file by following the instructions in the FIXME tags and use the provided image `shapes.png` to show your results.

Please submit the following:

1. Your completed source code file `polygon_detection.py`. Moreoever, please **include** the relevant lines of code in your LaTeX file and briefly **explain** your implementation. (5%)
2. The image with all marked polygons (squares, rectangles and triangles). (5%)
3. A screenshot of the terminal window showing the list of squares, rectangles and triangles. (5%)

[1] Hough Circle Transform tutorial: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghcircles/py_houghcircles.html

[2] Douglas-Peucker algorithm for polygon approximation *approxPolyDP()*: https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html#ga0012a5fdaea70b8a9970165d98722b4c

6 Shape Detection in ROS 15%

In this task, you will integrate the shape detection algorithms from Tasks 4 and 5 into ROS2 nodes. This will enable your TurtleBot to detect shapes in the environment using its RGB camera that publishes images in ROS.

For this task, two skeleton files named `circle_detection.py` and `polygon_detection.py` have been provided in the ROS package `shape_detection`. Unzip the `shape_detection.zip` package into your `rdl_ws`, carry out the required changes marked by FIXME tags and build the package using `colcon`. If you want to build one ROS node at a time, comment out the desired node entry from the `setup.py` file.

In StudIP, we provided three bagfiles (`rosbag2_01_bright`, `rosbag2_02_dark`, and `rosbag2_03_driving`) that contain recorded camera images published on the topic `/image`. In order to test your nodes, first start the nodes in separate terminal windows, then play the bagfile in another terminal window:

```
Terminal 1: ros2 run shape_detection circle_detection
Terminal 2: ros2 run shape_detection polygon_detection
Terminal 3: ros2 bag play name-of-bag-file.db3
```

Remarks:

- Please tune the parameters of the shape detection functions (e.g. Gaussian noise, gray image threshold, Hough circle detector parameters) such that all shapes are detected as good as possible with the provided bag files. You are allowed to filter out polygons based on their size, in order to eliminate false detections due to noise.

  Hint: In order to find the area of a triangle, you can use the following function in OpenCV:
  `[area, _] = cv2.minEnclosingTriangle(np.array(shape, dtype=np.float32))`

- Optionally, test your shape detection nodes using camera images directly from your TurtleBot instead of bag files and observe the detection quality for different distances, robot motions and light conditions.

Please submit the following:

1. The completed skeleton files `circle_detection.py` and `polygon_detection.py` from the ROS package `shape_detection`, as well as the first image from the `rosbag2_01_bright` bagfile with marked circles and polygons. (5%)
2. The number of circles, the radius of the largest circle, and the number of red circles from the first image in the `rosbag2_01_dark` bagfile. (5%)
3. The number of squares, rectangles and triangles from the first image in the `rosbag2_01_driving` bagfile. (5%)

Note: For this task, you **do not** have to explain your code in the LaTeXfile.

7 Task Planning with PyTrees 15%

In this task, you will create a small behaviour tree containing three nodes as shown in Figure 1.
The **sequence node** named `find_next_waypoint` has two child nodes:

- A **condition node** named `robot_stationary` that checks whether the robot is stationary.
- An **action node** named `detect_max_num_shape` that analyzes the most recent output from the shape detection nodes in Task 6 and finds out which shape(s) have been detected the maximum number of times.

Your task is to implement this behaviour tree using the `py_trees` library. For this, we have provided a ROS2 package named `maze_game` with skeleton code containing #FIXME hashtags. Please follow the next steps:

1. In you home directory, create a folder named `repo`:
   `mkdir ~/repo`

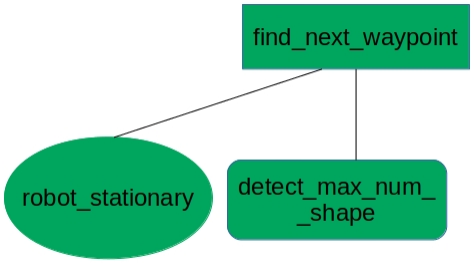

Abbildung 1: Behaviour tree for Task 7

2. Clone the `py_trees` repository into the `repo` folder:
   ```
   cd ~/repo
   git clone https://github.com/splintered-reality/py_trees.git
   ```
   Note: You might need to use the "eduroam" network to clone `py_trees`.

3. Build and install `py_trees`:
   ```
   cd py_trees
   python3 setup.py install --user
   ```

4. Unzip the `maze_game` package we have provided in Stud.IP and copy the folder to `~/rdl_ws/src`.

5. The `maze_game` package contains two files named `behaviour_tree.py` and `waypoint_finder.py`, which contain several #FIXME tags. Follow the instructions provided and complete those parts of the code. [15%]

6. Build `maze_game` package using `colcon build`:
   ```
   cd ~/rdl_ws
   colcon build --packages-select maze_game
   ```

7. In four different terminals, start the following nodes:
   ```
   ros2 run turtlebot3_teleop teleop_keyboard
   ros2 run shape_detection circle_detection
   ros2 run shape_detection polygon_detection
   ros2 run maze_game waypoint_finder_tree
   ```
   Note: For convenience, we recommend you to write a launch file to launch the above four nodes.

8. Play any bag file from Task 6 to test how the tree works. If everything is working well, you should see that after some time the `waypoint_finder_tree` node prints that the tree completed successfully. You would also be able to see that the IDs of shapes that have been detected the most no. of times are being published on the topic `/most_detected_shape`. Note that you might have to play the bag file 2 or 3 times until the tree succeeds.

9. **Please submit the following:**
   - The zipped version of your modified `maze_game` ROS 2 package which contains the completed files `behaviour_tree.py` and `waypoint_finder.py`.
   - A screenshot of the output from `waypoint_finder_tree` node showing that the tree has completed successfully.
   - A screenshot of the output being published on the topic `/most_detected_shape`.

10. What comes next: This behaviour tree is part of a larger behaviour tree, which will be used for the competition to determine the next waypoint based on the shape that is present in maximum number. We will release the outline of the larger behaviour tree in the next worksheet.

8 Feedback

Your feedback is very important to us. Please briefly answer the following questions:

1. How much time did you spend on doing this sheet per person? Anonymize your answer!
2. Was it too easy, easy, ok, hard, too hard?
3. Please tell us what you liked in this exercise sheet.
4. Did you face any difficulties? What should be improved?
5. Any other general remarks?

9 Submission Procedure

- Please use the LaTeX template provided in *StudIP/Wiki* to write your solutions. Upload the PDF file together with source code and other additional materials as a .zip file in StudIP.
- The naming style of your submission should follow the pattern **Gxx_0y_lastname1_lastname2_lastname3.zip**, where xx stands for the group number and y stands for the exercise sheet number.