Multi-robot pipe manipulation:

Grasping specification stage

Aziza Zhanabatyrova

THIS PROJECT WAS DONE UNDER THE SUPERVISION OF RAÚL MARÍN PRADES FOR THE COURSES OF "PERCEPTION AND MANIPULATION" AND "TELEROBOTICS"

UNIVERSITY JAUME I

SOURCECODE: /GITHUB.COM/AZIZAZHANABATYROVA/UNDERWATEROBJECTGRASPING

18 January 2018

Contents

1. Introduction

1.1 Motivation

Underwater robotics represents a rapidly growing research area and a promising industry. Great efforts are being made in order to develop autonomous underwater vehicles which would be capable to overcome challenging problems caused by unstructured and dangerous underwater environments. Such robots should be able to work autonomously and take important decisions when the human support is unavailable.

1.2 Objective

The objective of this project is to develop a program in which a robot must:

- identify an object underwater under any angle and from any distance. A neural network classifier was used for that purpose
- decide proper places to be able to grasp the identified object in a stable manner
- if the type of the object assumes the cooperation of two robots for a stable execution of a grasp, then decide which of the robots will grasp at which place
- execute the grasping phase

2. Materials and Methods

2.1 Software and Tools

The following software was used during the development of the project:

- ROS, a Robot Operating System, which provides libraries and tools to help software developers create robot applications
- Python programming language, which is an interpreted high-level programming language for general-purpose programming
- VirtualBox 5.1.28, a virtual machine for virtualization of operating systems
- Uwsim-UbuntuMate 14.04, an underwater simulator for marine robotics research and development, installed on a virtual machine [1]
- Scikit-learn library, which is a machine learning library for the Python programming language
- Pickle library to save a trained model
- Multiprocessing library to use all CPU cores when training a model, to decrease processing time
- OpenCv library for image processing

2.2 Methods

In this section, we describe the working environment and the steps of program development.

2.2.1 Working environment

The uwsim simulator contains two folders that were used during the development of the project, pipefollowing and underwater_simulation. Underwater_simulation package was used in order to modify the default scene used in the Uwsim simulator.

The Simulator contains multiple default scenes. For this project the pipeFollowing_basic.xml scene was used. There are some objects in this scene by default, such as terrain, girona 500 underwater vehicle, straight green pipes (Fig. 2.1).

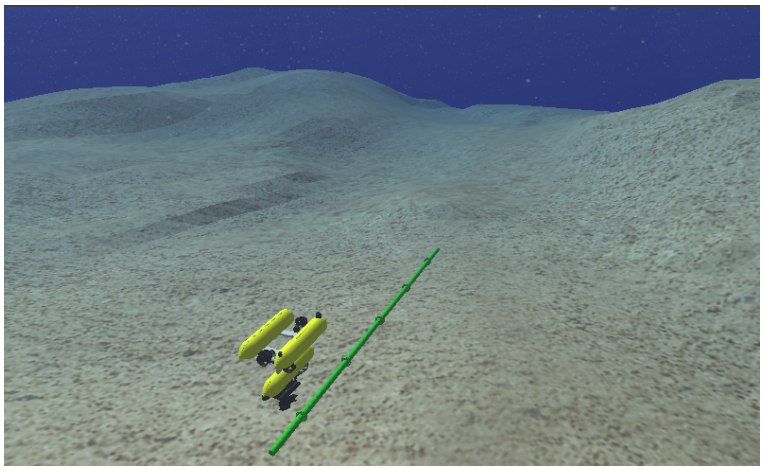

Figure 2.1: Uwsim basic scene

The vehicle by default has camera, gripper, motors, etc.

2.2.2 Scene modification

In order to change the default basic scene of the simulator, the xml file pipeFollowing_basic.xml was modified.

The pipes were modified by adding some pipes with corners, which is needed for variety of objects to be identified (Fig. 2.2).

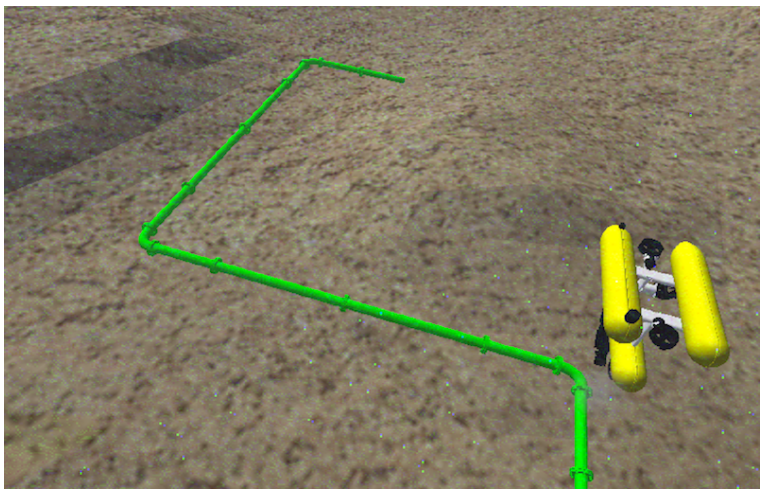

Figure 2.2: Pipes with corners

One more underwater vehicle was added to the scene, so there are two vehicles in total. The second vehicle is almost identical to the first one but has its own sensors and Ros topics (Fig. 2.3).

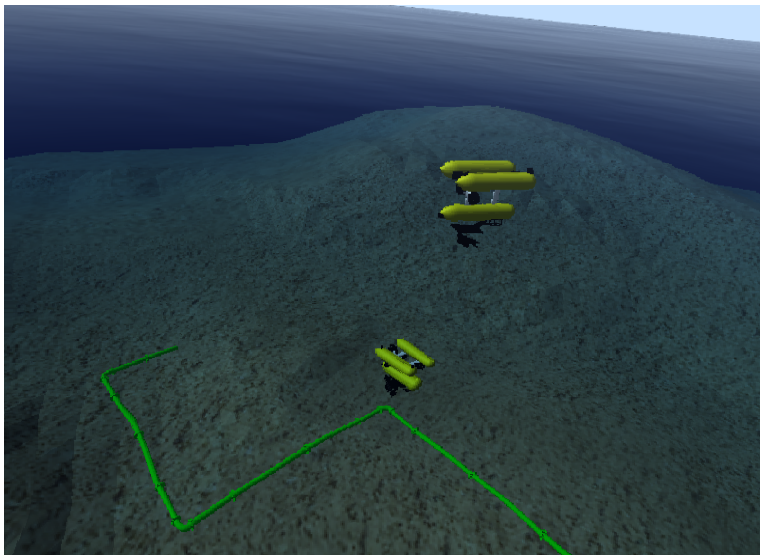

Figure 2.3: Two vehicles

The vehicles were modified by activating additional sensors, such as range sensor to measure distance to objects, and odometry, to measure how much the vehicle has moved.

Additional 3d objects of the type 3DS were downloaded from open source websites and imported into the scene. These objects include a bear, a hammer and a frying pan. Unlike the green pipe, they have white color, which will be helpful in object identification (Fig. 2.4).

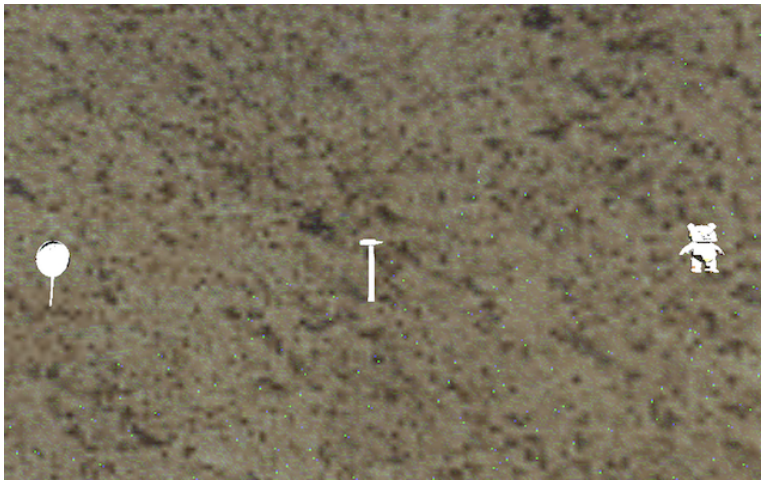

Figure 2.4: objects

2.2.3 Data acquisition

For the classifier to work properly 5583 unique training examples were collected. The procedure for data collection is as follows:

- With a constant rate images are captured using the vehicle's camera
- The vehicle is teleoperated using the keyboard in real-time, in order to be able to take images

from any angles and any distances (Fig. 2.5)

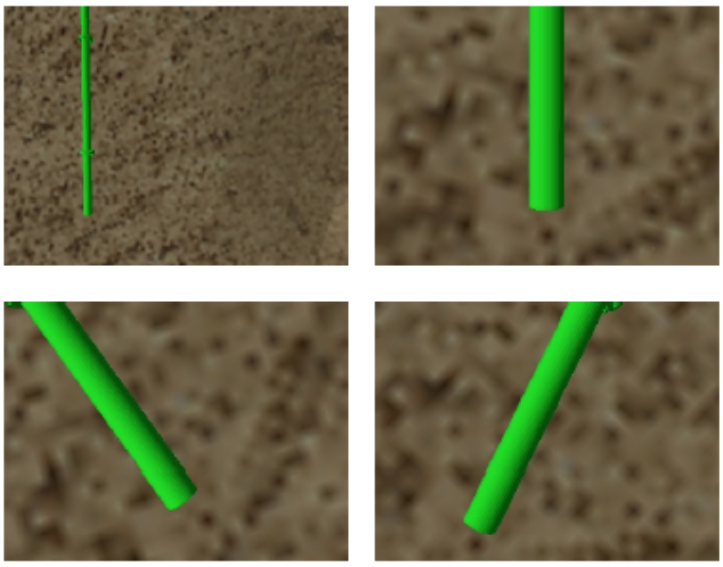

Figure 2.5: Object from different perspectives

- Every time an image is available, it undergoes necessary image processing using OpenCv library
- The user indicates for which object the data is collected
- From every picture some features of an object are extracted in order to calculate 7 mathematical descriptors. These descriptors include:
  - Color descriptor. It helps to differentiate green pipes from the white object.
  - Centroid location descriptor, which helps to differentiate corner pipe from other objects, as it is the only object that has its centroid outside the object area.
  - Compactness descriptor (Fig. 2.6) [2], where P is the perimeter of the object, and S is the area

$$C = \frac{P^2}{S}$$

Figure 2.6: Compactness descriptor formula

  - Elongatedness descriptor (Fig. 2.7) [2]

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q$$

$$I_{min} = \frac{\mu_{20} + \mu_{02} - \sqrt{4\mu_{11}^2 + (\mu_{20} - \mu_{02})^2}}{2}$$

$$I_{max} = \frac{\mu_{20} + \mu_{02} + \sqrt{4\mu_{11}^2 + (\mu_{20} - \mu_{02})^2}}{2}$$

And then, the L_v descriptor is defined as

$$L_v = \frac{I_{min}}{I_{max}}$$

Figure 2.7: Elongatedness descriptor formula

– Spreadness descriptor (Fig. 2.8) [2]

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x,y)\, dx\, dy \qquad \text{for } p, q = 0, 1, 2, \ldots$$

$$R_v = \frac{I_{min} + I_{max}}{m_{00}^2}$$

Figure 2.8: Spreadness descriptor formula

– Two Hu descriptors (Fig. 2.9) [2].

$$\eta_{pq} = \frac{\mu_{10}}{\mu_{00}^{\gamma}}$$

$$\gamma = \frac{p+q}{2} + 1$$

For $p+q = 2,3,\ldots$

$$\phi_1 = \eta_{20} + \eta_{02}$$
$$\phi_2 = (\eta_{20} + \eta_{02})^2 + 4\eta_{11}^2$$

Figure 2.9: Hu descriptors formula

- The descriptors as well as the object class are saved into an csv file.
- One hot encoding was performed on the output classes, in order to form them appropriately for a better performance of the classifier.

2.2.4 Object classification

For a vehicle in order to be able to identify a specific object, a multi-layer perceptron classifier was used.

The Scikit-learn library was imported. It allows to use a complete preprogrammed MLP classifier for multi-class classification.

The parameters of the classifier were found using K-fold cross validation technique. K-Fold Cross Validation is a technique used to compare different models, with the aim of fine-tuning parameters such as the number of hidden layers and neurons in each hidden layer. The reason for which K-Fold is used instead of the conventional validation method - where the dataset is simply split into two partitions: training set and test set - lies on the fact that the conventional validation leaves us with a high probability that difficult cases are not going to be contained in one of the two sets, which, as a consequence, result in a loss of modeling and testing capability.

K-Fold consists of splitting the dataset into k folds, where (k-1) folds are used for the training, while the remaining fold is used for testing. This procedure is repeated k times, thus obtaining k models training with different training and test sets. For each model, there is an accuracy associated with it.

The next step in the K-Fold technique is to find the weighted average with the accuracies of all models. This weighted average is a measure of how good the classifier performs and serves as a mean of comparison between other classifiers.

Multicore processing technique was implemented for training K-fold cross validation, in order to decrease the training time. It is mostly useful for large datasets. In K-Fold Cross Validation, all the trainings are independent of each other. Assigning a training to each core of the CPU will make each training be executed in parallel, at the same time. For example, if a computer has two cores, instead of executing one training at a time, it is possible to execute two trainings simultaneously with this technique.

As an input to the model 5 mathematical descriptors were given, and as an output the object's class. The classes are: bear, pan, hammer, straight pipe and corner pipe.

After the model was trained on the whole dataset using the found parameters, it was saved using a pickle library, to be able to load it and use later.

2.2.5 Grasping points

The trained model was loaded using Pickle library into another file specifically created for the grasping part.

In this phase teleoperation by the user using keyboard is required as well.

Every image is processed again in the same way it was done for the data acquisition. It is needed in order to calculate the mathematical descriptors and to pass them as an input into the trained model for object prediction.

After an object is predicted, a grasping point is identified in a specific way for every object.

For bear and frying pan the procedure is as follows:

- the coordinates of the object's centroid are taken
- the distance is taken from centroid to every point of the contour of the object

- the coordinates of the farthest point from the centroid are taken
- a point in between the centroid and the farthest point is found, so that this point is 70% away from the centroid and 30% away from the farthest point. This point represents the grasping location.

For the hammer and the straight pipe, the grasping point is at the centroid of the object.
For the corner pipe, the stable grasp is possible in case the pipe is grasped in two places .The procedure to find two grasping points is as follows:

- the borders are extracted from the binarized image
- the straight lines are extracted from the borders
- the centers of two lines are found and the line connecting these two points
- the center of this final line is one grasping point
- since with the corner pipe we extract minimum four lines, therefore we are able to find two grasping points

2.2.6 Grasping the object

After finding the grasping pint the robot moves until the grasping point is in the center of the image. After it is centralized the information from the range sensor is obtained. The range sensor is located very close to the camera. In this way we obtain the x, y and z coordinates of the grasping point with respect to the camera's frame. X and y are zero because the grasping point is centralized with the image of the camera. Once these coordinates are obtained with respect to the camera's frame, the coordinate transformation is made from the camera's reference frame to the gripper's reference frame. Finally with this information the robot moves until the grasping point is very close to the gripper. The robots gripper is initially wide open.

Every robot detects two grasping points on the corner pipe. Both points are located at each straight part of the corner pipe, and the selection on which point should be grasped, is done by a human operator.

3. Results

3.1 Neural network results

From all the collected math descriptors only 5 were used, as others dramatically worsen the accuracy of the neural network. The included descriptors are Compactness descriptor, Elongatedness descriptor, Spreadness descriptor, Color descriptor and Centroid location descriptor.
The final parameters of the neural network, which showed the best results were:

- Activation function = relu (rectified linear unit)
- Solver "adam"
- Alpha = 0.01
- Initial learning rate = 0.01
- Number of iterations = 1000
- Number of hidden layers = 2
- Number of hidden units = 100
- etc.

The final accuracy of the neural network was 99.2%.

3.2 Grasping points illustration

The centroids of the objects are shown as green dots, while grasping points for the objects are indicated with a pink color:

- Bear (Fig. 3.1)

Figure 3.1: Grasping point for bear

- Frying pan (Fig. 3.2)

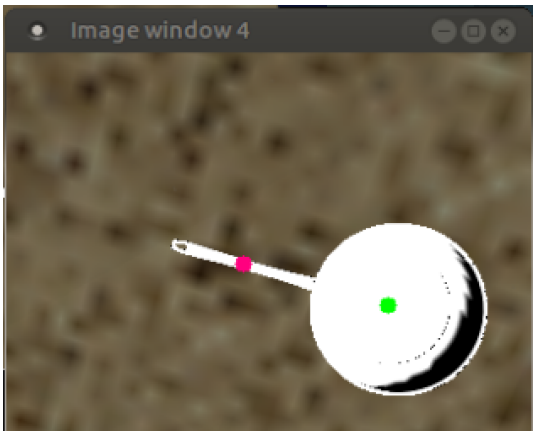

Figure 3.2: Grasping point for pan

- Straight pipe (Fig. 3.3)

Figure 3.3: Grasping point for straight pipe

- Hammer (Fig. 3.4)

Figure 3.4: Grasping point for hammer

- Corner pipe (Fig. 3.5)

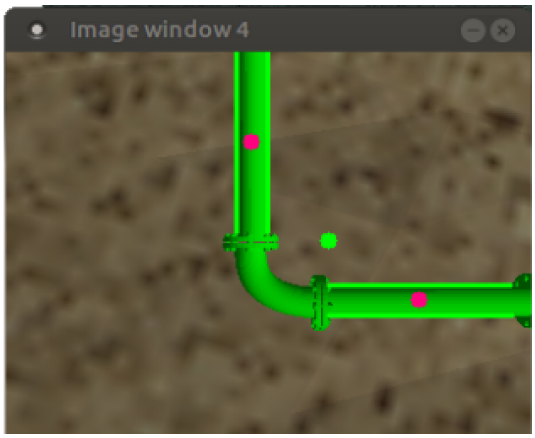

Figure 3.5: Grasping point for corner pipe

4. Conclusion

In this project, it has been studied the case of a underwater robot whose task is to grasp different types of objects. While for humans, it is a rather simple task, for robots, much processing is involved, as it needs to identify the object and then select a feasible grasping point on it. The methods used for the proposed task have shown to be very accurate in the simulator, thus being good candidates in a real life application of such scope.

Bibliography

[1] Sanjay K Dhurandher, Sudip Misra, Mohammad S Obaidat, and Sushil Khairwal. Uwsim: A simulator for underwater sensor networks. *Simulation*, 84(7):327–338, 2008.

[2] Raúl Marín Prades et al. The uji online robot: a distributed architecture for pattern recognition, autonomous grasping and augmented reality. 2002.