Application of the whole-body-control approach for a mobile robot with a manipulator

Bartosz Multan, Dawid Mościcki,
Tomasz Walburg, Mateusz Witka-Jeżewski
Faculty of Electronics, Telecomunications and Informatics
Gdańsk University of Technology

***Abstract*—This paper deals with the issue of a whole-body-control approach for a mobile robot with a manipulator. The goal of the project was to design a whole-body-control algorithm, which would allow the robot to perform pick and place tasks autonomously. The work contains a description of used algorithms such as Dex-Net, YOLO or SLAM. The process of integration of all system components was discussed. The outcomes of the preformed simulation were presented. The achieved results as well as encountered problems were also described.**

***Index Terms*—mobile robot, robotic arm, object detection, ROS, whole-body-control, YOLO**

I. Introduction

The whole-body-control approach is a method for controlling the motion of a robot by considering the interactions between the robot's various components, such as its base, manipulator, and sensors. The goal of whole-body control is to coordinate the motion of the robot's different parts to achieve a specific task or set of tasks, while also taking into account the robot's dynamic constraints and environmental factors. The whole-body control approach is distinct from traditional control methods, which typically focus on controlling the motion of individual components, such as the joints of a manipulator, in isolation. By contrast, whole-body control considers the robot as a holistic system and seeks to coordinate the motion of its different parts to achieve a more efficient, stable, and safe motion.

This approach allows the robot to achieve a greater degree of flexibility and versatility in its movements. Also, it enables the possibility to perform more complicated pick and place tasks.

To implement a whole-body-control algorithm there was a need for the integration of various sub-systems. A vision system with the ability to detect and distinguish objects was necessary. For this purpose, the presented result uses the YOLO algorithm. To perform a grasping action in a pick and place task Dex-Net algorithm was used. What is more to define the robot's position LiDAR sensor and RGBD camera were integrated.

In this paper the challenges that were encountered during the implementation of the approach and the solutions that were developed to overcome them were described. The results of experiments conducted to evaluate the performance of the whole-body-control approach were also presented and discussed.

A. Simulation environment and used components

To perform integration and tests, Gazebo and Robot Operating System were choosen. Clearpath Dingo was used as a mobile platform and Panda Franka Emika was used as a manipulator. To perform localisation and mapping, Sick 2D LMS1xx lidar and Intel Realsense RGBD camera were used.pp[]. To allow robot to see objects from manipulator perspective, one Realsense camera was placed close to the efector ending.

II. Autonomous driving

To aquire autonomous driving in indoor environment, system consisted of two main parts - localisation and mapping algorithm and motion planner algorithm.

Structure of the system has been presented below.

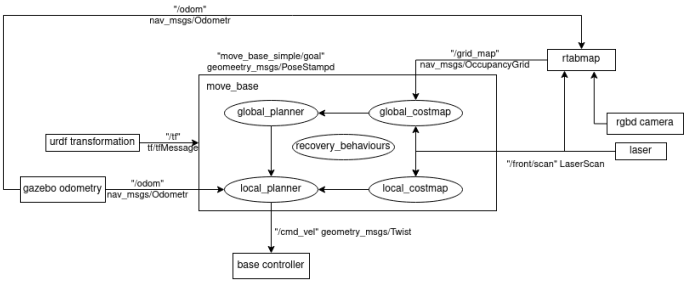

Fig. 1. Autonomous driving system

A. Simultaneous Localisation and Mapping

To achieve mapping and localisation for mobile platform, RTABMAP (Real Time appearance Based Mapping) [1] was choosen as a effective algorithm to implement 3D slam with loop closure detection. It uses lidar *LaserScan* messages from Sick lidar and rgb image and depth image from Realsense camera to create 2d grid map used for navigation and 3d pointcloud presenting the surrounding environment. As an odometry source, it has been decided to use odometry information given by the Gazebo simulator because of the most efficient way of getting odometry in terms of computation power needed performing simulation.

B. Motion planner

To move the platform in simulated environment, it has been decided to use *move_base* package with global planner *Navfn-ROS* and local planner *TrajectoryPlanner ROS*. *NavfnROS* is an implementation of fast, interpolated navigation function used to create paths to the target position. *TrajectoryPlanner ROS* is a local planner with creates set of different trajectories with kinematic constraints and scores them in terms of how close they are to the created global path. Local plan with the highest score is the send to the base controller as a *geometry message*. Octomap was considered as an extension of the system to calculate the possibility of going under the obstacles such as tables but it has not been implemented.

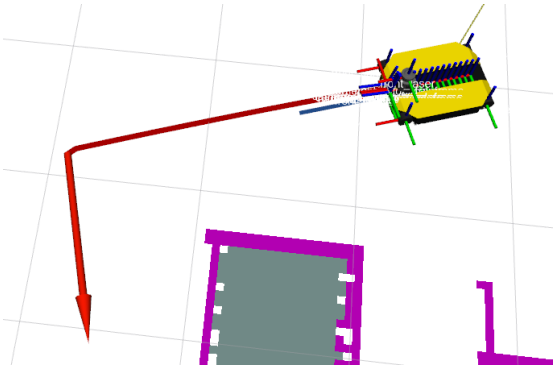

Fig. 2. Goal position (red arrow), global path (red line) and local path (blue line)

III. YOLO ALGORITHM

In this section YOLO algorithm was described. This section also contains a decryption of a process of integrating and installing dark-net ROS package [2].

A. Algorithm description

YOLO (You Only Look Once) is a real-time object detection algorithm. YOLO contains single convolutional neural network (CNN) that is able to detect objects in an image or video stream in real-time.

The YOLO algorithm divides the input image into a grid of cells, and for each cell, it predicts a set of bounding boxes and their corresponding class probabilities. Each bounding box is represented by a set of four numbers, which denote the coordinates of the top-left corner and the bottom-right corner of the box. The class probabilities are represented by a set of numbers, one for each class in the dataset.

YOLO algorithm is known for its fast detection speed, and good accuracy-speed trade-off compared to other methods such as R-CNN (Region-based Convolutional Neural Networks) or DPM (Deformable Part Model) [3]. However, the YOLO algorithm has some limitations, such as its tendency to miss small objects and its lack of rotation invariance. This nonetheless was not the issue in the application of the YOLO algorithm in the project.

B. Used dataset

YOLO algorithm used in the project is trained on COCO dataset. Based on that dataset YOLO can detect 80 object classes.

- person
- bicycle, car, motorbike, aeroplane, bus, train, truck, boat
- traffic light, fire hydrant, stop sign, parking meter, bench
- cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe
- backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard, sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket
- bottle, wine glass, cup, fork, knife, spoon, bowl
- banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake
- chair, sofa, pottedplant, bed, diningtable, toilet, tvmonitor, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair drier, toothbrush

C. Installation and integration

To run YOLO object detection package OpenCV and boost libraries are needed. In order to integrate the package with the project there was a need to modify ros.yml file and change camera_reading topic to /camera/color/image_raw.

IV. CONCLUSION

The conclusion goes here.

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] M. Labbé and F. Michaud, “RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation,” in Journal of Field Robotics, vol. 36, no. 2, pp. 416–446, 2019.

[2] M. Bjelonic ”YOLO ROS: Real-Time Object Detection for ROS”, URL: https://github.com/leggedrobotics/darknet_ros, 2018.

[3] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, ”You Only Look Once: Unified, Real-Time Object Detection,” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.