

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, Telecommunications 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 robots used in experiment

To perform integration and tests, Gazebo and Robot Operating System were chosen. As a holonomic mobile platform Clearpath Dingo robot has been used. As a manipulator with 7 DoF, to provide wide range of possible movements Panda Franka Emika has been chosen.

B. Sensor selection

Sensor selection involves considering various different approaches for subtasks needed to achieve main goal.

Four main subtasks were:

- Localisation and mapping
- Object Recognition
- Graps point definition
- Grabbing

First three tasks are essential in term of sensor selection, because grabbing alone could be performed based on *a priori* gained data. Also, sensors which are commonly used to acquire Object Recognition and Graps point definition are similar so they will be present in common subsection.

1) *Localisation and mapping*: Localisation and mapping is crucial when it comes to going through the environment and there are three approaches. First is using 2d lidar sensor, which gives information about obstacles on 2d plane. It is efficient and easy to implement and in singular-level buildings and without ability to go under the obstacles - completely adequate. If we would like to allow robot to go over (i.e. drones) it could be better to use 3d laser scanners (which give a point cloud data which allows to map 3d environment) or RGBD cameras which allow to create pointcloud with limited field of view and perform computer vision at the same time.

To perform localisation and mapping, Sick LMS1xx 2D Lidar and Intel Realsense RGBD camera were used as a source of both 2D laser scan information and 3d pointcloud and rgb image. It provides possibility of using visual odometry to calculate difference in robot's pose i

2) *Object Recognition and Graps point definition*: To recognise objects, it is necessary to have their rgb image, which has to be captured by the camera. To define grabbing point it is important to have the rgb data and pointcloud to perform shape recognition. To meet both dependencies, it has been decided to use Realsense RGBD camera which has good implementation in Gazebo. Even though Realsense does not

provide pointcloud data directly, it can be computed from rgb image and depth image.

To allow robot to see objects from manipulator perspective, one Realsense camera was placed close to the effector ending.

II. AUTONOMOUS DRIVING

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

The structure of the system has been presented in Figure 1.

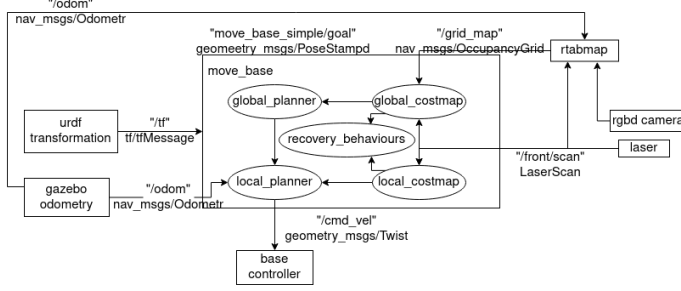


Fig. 1. Autonomous driving system

A. Simultaneous Localisation and Mapping

To achieve mapping and localisation for mobile platform, RTABMAP (Real Time appearance Based Mapping) [1] was chosen as an 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 *NavfnROS* 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 sent 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.

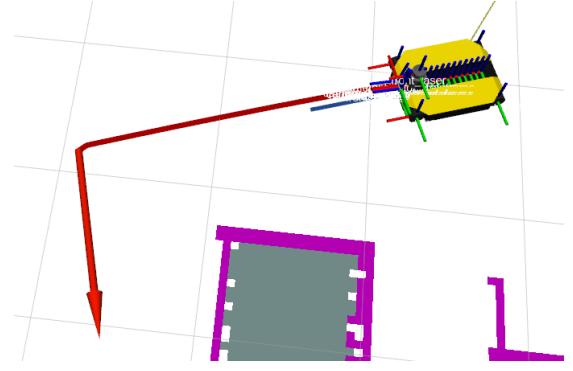


Fig. 2. Goal position (red arrow), global path (red line) and local path (blue line)

III. MANIPULATOR CONTROL

To control Panda Franka Emika, which is 7 DoF robotic arm, it has been decided to use MoveIt. MoveIt is a Motion Planning Framework integrated with Robot Operating System. It is primarily used for mobile manipulation and grasping tasks, and can be integrated with various robot platforms and sensor systems. MoveIt provides a high-level interface for controlling a robot's motion, and includes features such as motion planning, collision detection, and visualization tools [2]. MoveIt uses IKFast the Roboti Kinematics Compiler which is a powerful inverse kinematics solver.

IV. OBJECT RECOGNITION

In this section YOLO algorithm was described, which was used for object recognition task. This section also contains a decryption of a process of integrating and installing dark-net ROS package [3].

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) [4]. 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. Also, before running YOLO algorithm pretrained weights should be downloaded. There are two possible options: weights for YOLO tiny and regular weights. Weights for YOLO tiny are used in smaller model, which is substantially faster, but less accurate. The weights can be downloaded by running a command:

```
# wget http://pjreddie.com/media/files/yolov3.weights
After dowladnig weights the package can be run by a following command:
```

```
# roslaunch darknet_ros darknet_ros.launch
```

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.

D. Object detection results

Integrated tiny version YOLO algorithm works correctly under certain conditions. The object must be specified in the dataset. What is more, the background and the lightning conditions should be appropriate. The tiny version of the algorithm uses less computing power, but it is less accurate in its predictions. The result of detecting an object is shown in a figure below.



Fig. 3. Detecting a banana using YOLO algorithm.

V. WHOLE BODY MOTION CONTROL ALGORITHM

For the purpose of this experiment, let assume that the approximate location of the desired object is known and it has been added to the map as a label. Then, next steps for the algorithm are as presented below.

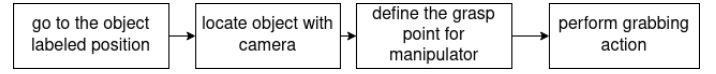


Fig. 4. Sequence of achieving the task

Grabbing action should also be extended to measure the distance from the base_link of the object and execute the base movement if it is necessary. To maintain all states and autonomy, the state node has been proposed with configuration showed below.

Autonomy actions would be maintained by *state_controller* node which could have structure presented below.

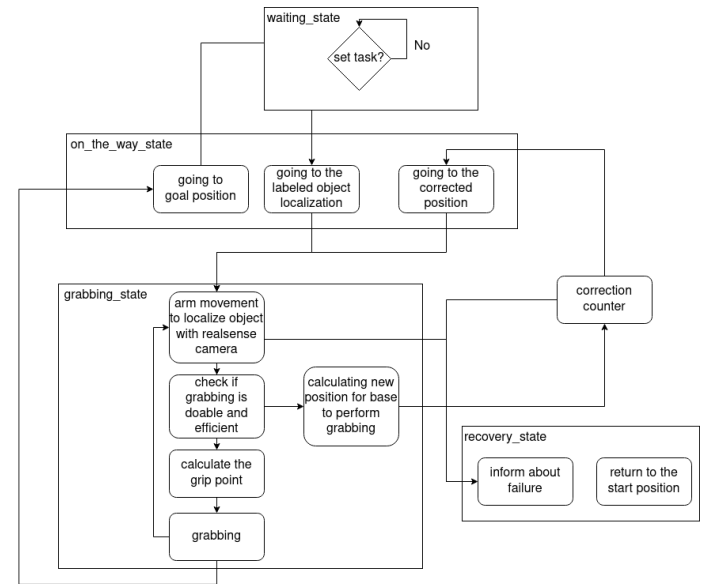


Fig. 5. States controller concept

VI. DIFFICULTIES

During developing whole system, many difficulties occurred. Implementation problems prevented completing the task of creating whole-body-motion control algorithm due to some significant unresolved errors.

A. Integrating Dingo and Panda

During integrating mobile robot Dingo with robotic arm Panda first step was to create common *URDF* file to connect both robots together. However, for the simulation purposes, all movable parts have to be described by *Hardware_Interface* and placed in common set. Because of the written drivers, with checking number of joints it was impossible to run the simulation. After changing the condition values, it turned out that Gazebo simulator is not able to control more sets of

Hardware_Joints than one because of the namespace conflict. Probably it could be repaired by rewriting part of the Panda driver, but it was too time consuming to perform successfully.

VII. 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] David Coleman, Ioan A. Şucan, Sachin Chitta, Nikolaus Correll, Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study, *Journal of Software Engineering for Robotics*, 5(1):3–16, May 2014. doi: 10.6092/JOSER_2014_05_01_p3.
- [3] M. Bjelonic "YOLO ROS: Real-Time Object Detection for ROS", URL: https://github.com/leggedrobotics/darknet_ros, 2018.
- [4] 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.