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 robots used in experiment

To perform integration and tests, Gazebo and Robot Operating System were choosen. 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 choosen.

B. Sensor selection

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

Four main subtasks were:

- Localization 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 aquire Object Recognition and Graps point definition are simmilar 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 computated from rgb image and depth image.

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.

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

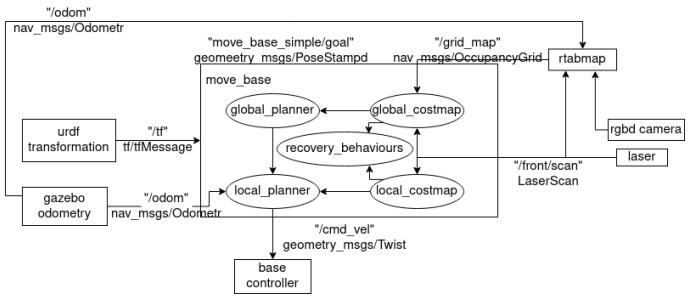

Fig. 1. Autonomous driving system

A. Simultaneous Localisation and Mapping

To achieve mapping and localisation for mobile platform, RTABMAP (Real Time appearance Based Mapping) [1] was 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 in comparison to visual or icp odometry algorithms.

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 as a part of navigation pipeline.

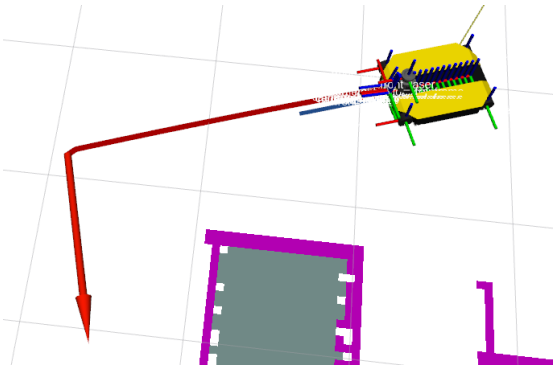

Fig. 2. Goal position (red arrow), global path (red line) and local path (blue line)

C. Environment information for robotic arm planner

To safely operate with robotic arm, it is important to gain the knowledge about the surrounding, which can be performed alone with camera mounted on the arm or during mobile platform mapping. It has been decided to use the second approach and use gathered point clouds to create OctoMap which is a 3D occupancy grid mapping. [2]. Octomap can be used as obstacles map for MoveIt planner for Panda arm in grasping phase.

III. Manipulator control

To control Panda Franka Emika, which is 7 DoF robotic arm, it has been decided to use MoveIt. MoveIt is a powerful tool for controlling the motion of a robotic arm. It is built on top of the Robot Operating System (ROS) and provides a high-level interface for controlling the motion of a robot. This includes features such as:

- Motion planning: MoveIt can plan collision-free trajectories for the robot to move between different states.
- Collision detection: MoveIt can detect collisions between the robot and its environment.
- Visualization tools: MoveIt includes visualization tools that can be used to display the robot's current state, planned trajectories, and collision detection results.
- Grasp planning: MoveIt can be used to plan grasps for grasping an object. It uses the IKFast, a powerful inverse kinematics solver, to compute the required joint angles to reach a target end-effector pose.
- Motion execution: MoveIt can execute planned trajectories on the robot and also includes low-level controllers for sending commands to the robot's joints.

To perform object grasping using a Moveit following steps are required :

1) Object detection: The robotic arm needs to be able to detect and locate the object to be grasped. This can be done using various algorithms in case presented in this work it will be YOLO network.
2) Grasp planning: Once the object has been located, a grasping plugin, such as the Grasp Pose Planner or the MoveIt Grasp Library, is used to generate a set of valid

grasp poses for the object. These grasp poses take into account the geometry of the object and the capabilities of the robotic arm. In these work different aproaches were tested and described below.

3) Motion planning: The robotic arm uses the MoveIt Motion Planning pipeline to plan and execute a trajectory for the robotic arm to move to the grasp pose. The motion planning pipeline takes into account the kinematics and dynamics of the robotic arm, as well as any obstacles that may be present in the environment which in this case are going to be created by Octomap.
4) Grasp execution: Once the robotic arm reaches the grasp pose, the gripper is closed to grasp the object. Force-torque sensor data is used to ensure that the robotic arm applies the appropriate force to grasp the object without damaging it.
5) Object manipulation: Once the object is grasped, the robotic arm uses the MoveIt Motion Planning pipeline to plan and execute a trajectory for the robotic arm to move the grasped object to the desired location.
6) Release: The gripper is opened to release the object.

IV. Object recognition

In this section YOLO algorithm was described, which was used for object recogniton task. This section also contains a decryption of a process of integrating and installing dark-net ROS package [5].

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) [6]. 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.

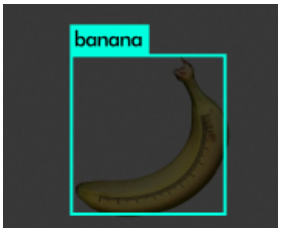

Fig. 3. Detecting a banana using YOLO algorithm.

V. Grasp detection

This section describes the problems our team encountered when implementing the grasp detection algorithms into the project.

A. Dex-Net and GQC Neural Networks

Dex-Net and GQCNN packages are well suited for both Python, and ROS environment. While Dex-Net focuses on generating datasets of grasp robustness, the GQCNN is based

on the Grasp Quality Convolutional Neural Networks which finds the best grasp from a group of candidates. Structure of this system has been shown below.

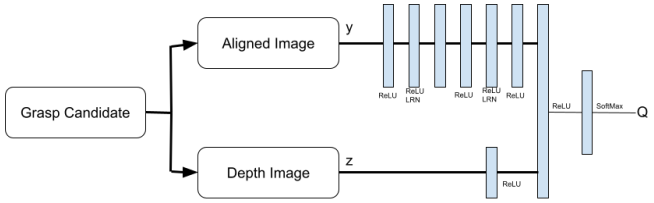

Fig. 4. Structure of Grasp Quality Concolutional Neural Network

Dex-Net is the library that has seen the most progress during the tests. It was run and tested using a high-level Python programming language. Unfortunately, though, issues occurred during its integration into the ROS environment.

B. MoveIt Deep Grasps

MoveIt Deep Grasps is a ROS package developed by Picknik.ai. It is a more advanced version of the MoveIt Grasps library. The package uses three tools:

- Grasp Pose Detection
- Dex-Net
- MoveIt Task Constructor

The MoveIt Task Constructor is used to ensure ROS communication with the decision-making system, which can be either Dex-Net or Grasp Pose Detection package. Both libraries must be installed on the computer before installing the ROS MoveIt Deep Grasps package.

During the installation process, many problems have occurred. With time, most of them have been resolved, however two issues could not have been fixed. It revolved around the compatibility of the OpenCV version, used by MoveIt Deep Grasps. Another error that was not resolved was the absence of certain files in the package itself.

C. MoveIt Grasps

MoveIt Grasps is yet another package designed for a Robot Operating System, maintained by the aforementioned Picknik.ai. Its main task is to generate grip points for simple objects, such as blocks and cylinders. This package is continuation of the MoveIt pick and place library. It is also the most popular package for grasp detection.

Attempts of installing the package have failed due to file compatibility issues. Extensive research has shown that due to MoveIt changes made in 2021, the package could no longer be supported. As a result, it became impossible to use it in our version of software.

D. Conclusion

The task of determining a grip point has not been achieved. There are several reasons for this situation. The first may have been little experience with tools such as dex-net along with little experience of ROS. Another reason is the fact that there are few tools available dedicated for the grasp detection. Moreover, not only are such tools scarce, but most of them ceased to be supported at least a few years ago.

Creating such a tool in a small group and with such a limited amount of time is very difficult. This is due to the need to collect a large dataset of images of objects, and to create an entire algorithm based on neural networks. Such time was in short supply due to the diversion of our efforts to solve problems with publicly available tools.

VI. 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.

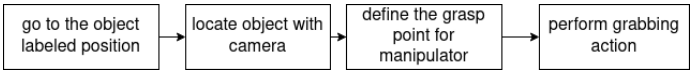

Fig. 5. Sequence of achieving the task

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.

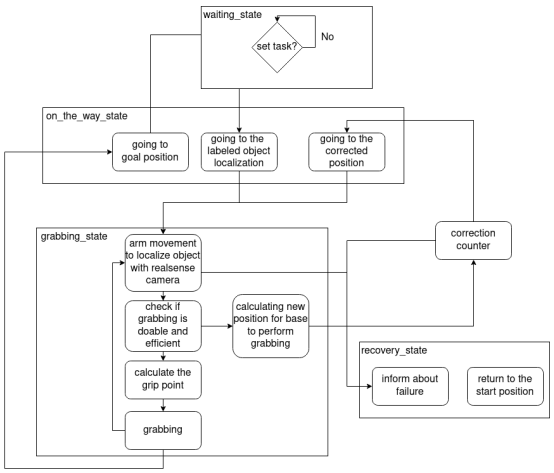

Fig. 6. States controller concept

Next steps of achieving whole-body control could be to implement shared URDF file with mobile robot and robotic arm to MoveIt planner ad creating MoveGroupInterface object, to perform whole-body control with plan() function. When the pose is created, next goal for *move_base* could be send to force mobile platform to change its location. Due to not achieving URDF file describing whole robot, it was not possible to evaluate this solution. This problem has been described below.

To minimize the energy consumption movement, the cost function has to be properly described as a sum of torque of each link needed for achieving desired position and energy

needed for moving mobile platform to the necessary position. Function has been presented below

$$\arg_{v,\tau} \min = \sum_{i=1}^{7} \tau_i^2 + E_{pos} \tag{1}$$

where τ is a torque for i-th link and E_{pos} is energy needed for the robot to move required distance given by path length, its mass and velocity v.

VII. Other difficulties

During developing whole system, many diffculties occured. 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. At the begginig, the *tf_tree* was broken, due to not all links and joints beeing created. Because of the written drivers, with checking number of joints it was impossible to run the simulation. After changing the condition values which are responsible for checking the amount of joints to register to the *ros_control* package, it turned out that Gazebo simulator is not able to control more sets of *Hardware_Joints* than one, probably because of the namespace conflict of Dingo and Panda robots. Probably it could be repaired by rewritting part of the Panda driver which use Hardware Joints interface and to separate these joints from joints used by *move_base*, but it was too time consuming to perform succesfully.

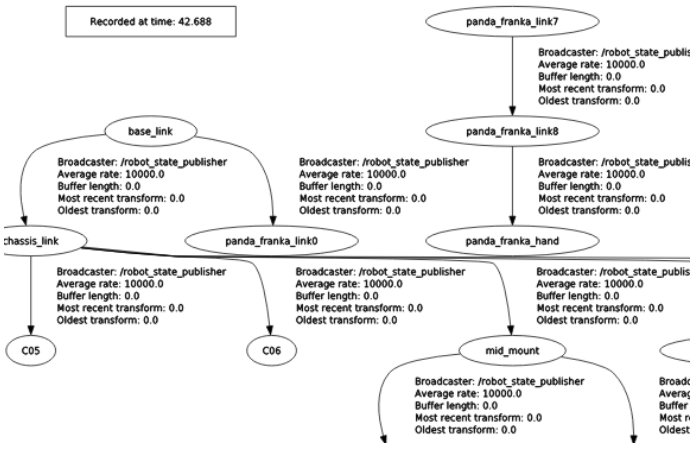

Fig. 7. Broken tf tree without panda links and joints other than base and gripper

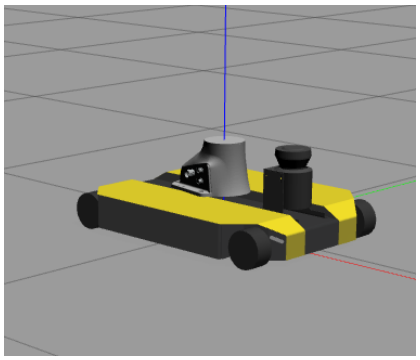

Fig. 8. Generated in gazebo robot model with broken tf tree

After some work and rewritting URDF files to provide connectivity and proper parametrisation, full model presented in Gazebo was achieved, but due to drivers issue mentioned above gazebo simulator dies after launching full mode with message

```
Segmentation fault (core dumped)
gazebo-5 process has died pid 8354, exit
code 139
```

VIII. Conclusion

The presented algorithm for the whole-body-control approach might be useful in various robotic systems. However, its implementation using Robot Operating System proved to be problematic especially with the newest distribution of ROS and Ubuntu. Various integration problems indicate the selection of another technological stack. Further research is needed to fully explore the potential of this approach and implement the cost function for arm and mobile base simultaneous control in terms of energy optimization. With longer time horizon, rewritting presented algorithms could be performed to allow them operate with currently available versions of libraries needed, but in therms of quick implementation and develop the algorithm, using older and better debugged versions is worth considering, but would not be a solution because of the necessary features which have been brought up in last versions.

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] http://wiki.ros.org/octomap

[3] David Coleman, Ioan A. Șucan, 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.

[4] http://docs.ros.org/en/noetic/api/moveit_msgs/html/msg/Grasp.html

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

[6] 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.