An Optimization Based Cartesian Controller for Mobile Manipulation of Service Robots in Domestic Environments

Emilia Brzozowska
Department of Computer Science and Engineering
Instituto Superior Técnico
Email: emilia.brzozowska@tecnico.ulisboa.pt

Oscar Lima, Rodrigo Ventura
Department of Electrical Engineering
Instituto Superior Técnico
Email: {olima, rodrigo.ventura}@isr.ist.utl.pt

Abstract—This work is about robust execution of complex manipulation tasks for service robots operating in dynamic environments. Our goal is the reliable manipulation of objects with vision in the loop. We identify two main problems, the first one being the lack of inverse kinematic solution for particular arm configurations and the second, accurate robot base-camera-arm calibration. To solve these problems we propose an optimization based Cartesian controller that is able to control the robot base and arm in a combined way. Our results show that the controller is able to reach random arm configurations with a high probability of success.

Applications of this controller include for instance: object grasp, place, human to robot object handover, visual servoing, etc.

Keywords—optimization based cartesian controller, whole body motion control, combined control of robot base and arm, extended jacobian, visual servoing, real-time object manipulation

I. INTRODUCTION

A. Motivation

Robust execution of complex manipulation tasks in dynamic environments is essential for service robots in domestic environments to assist their users in their daily tasks. Such tasks could include for instance, the transportation of objects between locations (e.g. from a table to a cupboard), which implicitly involve the successful execution of sub-tasks like perceiving, fetching, carrying and placing that object.

Typically, object grasping problems are approached by using separate offline path planning and open loop execution methods, which expose some disadvantages. For instance, during trajectory execution, the robot is not sensitive to changes in the environment. If the target object pose is changed, the robot should ideally deal successfully with those situations and adjust accordingly. Moreover such methods typically control only the arm and do not take into account the robot base. This situation often leads to failures in finding inverse kinematic solution, specially when the robot base is wrongly placed.

A second problem when it comes to object grasping is related with robot base-camera-arm calibration. This work is motivated by the possibility of bypassing the calibration problem, which can be achieved by fixing a marker to the end effector. Since both the object and the marker (located in the end effector) are perceived w.r.t camera frame, then we can overcome e.g. a badly calibrated camera, arm, or both.

B. Problem Statement

The problem is defined as the design, implementation and testing of a real-time closed-loop Cartesian controller which inputs the target pose from a perception module. The controller includes both robot base and arm combined such that its reachability is maximized. The developed solution is integrated and tested on the MBot robot (Fig 1) both in simulation and real robot. MBot is a service robot, ultimately used in a hospital to interact with autistic children.

We apply this method for a mobile service robot (MBot) in domestic environments, where household objects need to be manipulated, however the algorithms presented in this work are general and can be applied for robots acting in other environments as well.

The controller inputs a target pose from perception data, generated by using off the shelf solutions. The object detection part on itself is out of the scope of our work but rather a tool for pose generation purposes. As output of our model we have velocity commands for the arm servos along with velocities for the robot base. Our work is focused on the arm control to reach a target pose, where the robot end effector frame needs to be matched with a target frame e.g. an object, to perform a grasping task, but grasp planning on itself is out of the scope of this work.

Fig. 1: Robot attempting to grasp dishwasher handle.
C. Contributions of this work

Contributions of this work include:

- We have contributed with a generic and flexible real time base + arm closed loop cartesian velocity controller that is able to achieve in general (with its manipulator), a target pose, by moving robot base and arm combined.
- Development of a velocity controller hardware interface (driver) for Dynamixel motors derived from a open source implementation of position and trajectory control.
- Experimental evaluation of developed controller both in simulation and real robot.
- We contribute to the enhancement of the robot skills, as the controller can potentially be used for the following applications: object grasp, place, human to robot object handover, visual servoing, etc.

II. BACKGROUND

Robotic manipulators are kinematic chains composed of sequence of bodies, called links, and connected by joints. One end of the manipulator is fixed and the other one is free to perform a given task. Joints are the movable angular elements, which enable relative movement between neighbor links. Each joint has one degree of freedom and depending on its motion, we can classify it into two groups: linear or rotational type [3]. For manipulation and control of robotic arm it is important to understand the mathematical description of the position and orientation of its links in space [4].

To describe position and orientation of a robotic manipulator in 3D space, frames are attached to each of the links and end effector. The position and orientation of frames, with respect to a base reference frame, are mathematical description of the body’s location [9]. Knowing simple geometrical manipulator’s frames positions, with respect to other adjacent frames, it is possible to compute coordinates of the end effector from the given joints angles. This process is known as forward kinematics [6].

Inverse kinematics method is used to calculate a set of joints angles such that the end effector is at a desired pose in cartesian space. To solve the inverse kinematics problem, iterative methods can be used which use a sequence of attempts, leading to incrementally better configuration for the joints angles. Achieving a better configuration means minimizing the difference between the current and target positions of the robot’s end effector [5].

Another approach is known as closed-form method, where the solution of the joints angles configuration can be straightforwardly expressed as a set of closed-form equations. This method provides with a single solution when is used for 6-degree-of-freedom manipulators with special kinematic structure of kinematic chains [14].

For our case, the goal is to first calculate desired end effector velocity, to then compute correspondent joints velocities. To solve this problem, we need Jacobian matrix, which is obtained from the kinematics arm parameters [4].

III. RELATED WORK

Many research groups currently focus on the development of safe compliant systems for mobile manipulation in unstructured collaborative environments. A prominent example can be found in the work presented by Nimbro@Home team [11], which developed an arm controller based on differential inverse kinematics method to follow computed trajectories. The team solution for the inverse kinematics redundancy problem uses null-space optimization of the previously implemented cost function. Optimization criterion include convenient joint angles configuration and a penalty function for getting close to the joints limits.

Nieuwenhuisen et al. [7] use the kinodynamic motion planning by interior-exterior cell exploration algorithm [13] for the motion planning. Then they filter grasp poses and motion paths before execution against the height map they use, finding a collision-free solution of the inverse kinematics.

Chitta et al. [1] used randomized sampling based motion planners from the Open Motion Planning Library (OMPL),[12]. The approach for trajectory execution includes a state machine concept, that involves moving the sensors of the robot to maintain visibility of the robotic arm, which is executing the planned motion. The controller is tracking and executing the desired trajectory at the same time. For trajectory collision checking Chitta also uses inverse kinematics.

In the work presented by Stuckler et al. [10] mobile manipulation and motion control problem is approached separately for the robots wheeled mobile base and for the robotic arm. Control directions of individual wheel velocities are coming from inverse kinematics analytical solutions. For the arms they developed compliance controller with the servo actuators with limited torque. Collision free path is obtained with implemented differential inverse kinematics.

Ciocarlie et al. [2] research is another example of kinodynamic motion planning usage by interior-exterior cell exploration algorithm from the OMPL Library. In this approach, collision awareness is also provided with an inverse kinematics based method. The motion planner generates paths that are processed by a trajectory optimization planner. In this study, they pass a previously generated path from the OMPL as an initial condition for the optimization problem. The controller is responsible for eventual re-planning of the motion path.

Willow Garage [8] implemented sampling-based motion planning for reaching for the Personal Robot 2 (PR2). Similar to our approach, researchers from Willow Garage are using 3D perception pipeline, that gives dynamic obstacle map, which is used for collision checking.

Even though we have seen many approaches addressing mobile manipulation, few include solutions for both base and arm combined. In order to enhance this relationship functioning, we explore and approach that extends the jacobian matrix for whole body motion control functionality.

IV. APPROACH

Before an object can be manipulated, the robot needs to perceive the environment and the object 3D pose by using a perceptual module. The perception problem on itself, including
the object detection, 3D pose estimation from (noisy) sensor data and object classification are out of the scope of this work and therefore we use off the shelf ready solutions for integration purposes. In this work, we use the robot head camera out of the three that are available in the MBot robot (see Fig. 2, item marked with number 1).

In order to address the proposed challenges, we developed a solution that closes the loop between perception and arm control. Our visual servoing approach allows the robot to overcome problems such as changing the object goal pose in control. Our visual servoing approach allows the robot to find a solution that closes the loop between perception and arm controller module calculates new pose error and sends the updated error value to the optimisation module in a real-time closed-loop fashion.

The arm controller module is responsible for computing the pose error and the optimisation module for its minimization. Based on the error value, the optimization module computes desired updated velocities for the end effector in Cartesian space.

In the described architecture (Fig. 3) in a first step, we do not consider the motion of the robot base yet. We assume that the target pose is reachable and our arm controller module is focused on the manipulator only.

### A. Mathematical Model

The main goal of the arm controller is to minimize the difference between the goal pose \((P_g)\) and the current pose \((P_c)\). Pose \(P_i\) is an array, that gives information about position \((x, y, z)\) and orientation \((\alpha, \beta, \theta)\) of the frame \(i\) \((P_i = [x_i, y_i, z_i, \alpha_i, \beta_i, \theta_i])\). During calculation, we encounter a redundancy problem resulting from dimensions inequality: velocity in cartesian space is composed of 6 dimensions \(\dot{p} = [\dot{x}_c, \dot{y}_c, \dot{z}_c, \dot{\omega}_x, \dot{\omega}_y, \dot{\omega}_z]\) while the arm has 7 degrees of freedom, which gives us 7 dimensions \(\dot{\Theta} = (\dot{\Theta}_3, \dot{\Theta}_2, \dot{\Theta}_3, \dot{\Theta}_1, \dot{\Theta}_5, \dot{\Theta}_6, \dot{\Theta}_7)\). The optimisation defined in this way has an infinite number of solutions. For that reason, we defined the optimisation problem that minimizes the difference between the goal pose \((P_g)\) and the current pose \((P_c)\) of the end effector in the next optimisation step:

Minimize \[\|P_g - (P_c + J(\Theta)\dot{\Theta}t)|^2 + P(\dot{\Theta})\] w.r.t. \[\dot{\Theta} \] subject to \[\dot{\Theta}^{Min} \leq \dot{\Theta} \leq \dot{\Theta}^{Max}\] (1)

where \(\dot{\Theta}^{Min}\) and \(\dot{\Theta}^{Max}\) are velocity bounds, \(J(\Theta)\) is the Jacobian matrix, \(\Delta t\) is the time stamp and \(P(\dot{\Theta})\) is the barrier function, that penalizes for getting closer to joint limits and favours convenient configurations:

\[P(\dot{\Theta}_i) = \left\{\begin{array}{ll}
-\lambda \ln(\dot{\Theta}_i + \dot{\Theta}_i\Delta t) + (\dot{\Theta}_i^{Min}) + \beta & \text{for } \dot{\Theta}_i < \dot{\Theta}_i^{L} \\
0 & \text{for } \dot{\Theta}_i^{L} \leq \dot{\Theta}_i \leq \dot{\Theta}_i^{H} \\
-\lambda \ln((\dot{\Theta}_i + \dot{\Theta}_i\Delta t) + (\dot{\Theta}_i^{Max}) + \beta & \text{for } \dot{\Theta}_i > \dot{\Theta}_i^{H}
\end{array}\right.\] (2)

Where:

\(\lambda\) - Slope of the logarithmic function
\(\beta\) - Shift of the function relative to the y axis
\(\dot{\Theta}_i^{Min}\) - Joint \(i\) lower limit
\(\dot{\Theta}_i^{Max}\) - Joint \(i\) upper limit
\(\dot{\Theta}_i^{L}\) - Joint \(i\) low repulsion starting point
\(\dot{\Theta}_i^{H}\) - Joint \(i\) high repulsion starting point

The presented barrier function covers joint limits constraints, therefore they will not be mentioned in the extended mathematical model.

### VI. ARM AND BASE CONTROLLER DESIGN

In the second part of the development process, we extended the architecture design to include the base controller. The difference between the extended version and the previously described architecture is the robot base motion. In the arm controller version, we assume that the target pose is reachable for the robot manipulator without moving the base.

As previously presented (see Fig. 4) the distance between the arm and the object is unrestricted. Arm and base controller

---

**Fig. 2: MBot: 1) Head camera used in this project 2) Robot manipulator used in this project**

**Fig. 3: Proposed architecture, version only with arm.**

The optimisation module computes the best solution of joints velocities to the defined constraints and the repulsion potential function. Optimized cost function consists of desired Cartesian end-effector velocity and Jacobian matrix based on the current state of seven joints of the manipulator. The optimisation results with joint velocities and sends final speed information to the servos of the robotic arm. Finally, the forward kinematics method computes updated current end-effector pose and send it back to the arm controller. Arm controller module calculates new pose error and sends the updated error value to the optimisation module in a real-time closed-loop fashion.
module is receiving as input an error value, which is the difference between target end effector pose and current end effector pose. According to previous architecture, the controller computes the cartesian velocity of the end effector but this time with the base included. Calculated velocity is an input for the optimisation module that computes velocities for the manipulator servos and the base platform.

Considering arm and base motion, the developed controller might compute multiple possible motion solutions, for instance not moving the base at all and stretching the arm to its limits, or moving the base to the closest possible point, keeping end effector within a convenient distance from the robot.

In this work we distribute weights for obtained solutions, such that the base moves only when the target pose is hard to reach for the end effector or not reachable at all. An array of weights has been obtained experimentally as default values, but are left as parameters for the user to configure the desired behavior. Moreover, to avoid "backside grasping", we used an additional repulsive potential logarithmic function, which penalizes for rotating back to the object and favors situations where the robot base is oriented towards the object.

The function $W(v)$ is the importance weight function for each optimized value:

$$W(v) = \sum_{i=1}^{10} w_i |v_i|$$

(8)

Where $w$ is an array of weights coefficients $w$:

$$w = [w_1 \ w_2 \ w_3 \ w_4 \ w_5 \ w_6 \ w_7 \ w_8 \ w_9 \ w_{10}]$$

(9)

and array $v$ are the optimized velocities:

$$v = [\dot{\Theta_1} \ \dot{\Theta_2} \ \dot{\Theta_3} \ \dot{\Theta_4} \ \dot{\Theta_5} \ \dot{\Theta_6} \ \dot{\Theta_7} \ \dot{b_x} \ \dot{b_y} \ \dot{b_\Theta}]$$

(10)

VII. ARM AND BASE CONTROLLER IMPLEMENTATION

The implementation is based on the well known middleware ROS (Robot Operating System). For Jacobian matrix calculations we use the open-source package PyKDL\(^1\), which allows to extract kinematic chains from URDF\(^2\) model to then calculate Jacobian matrix based on current joint states.

In terms of optimization (error minimization) we use the open-source package SciPy\(^3\). SciPy is an open source software that provides several ready to use packages to solve mathematical, science or engineering optimization problems. In this work, we use least-squares minimization and in particular a Sequential Least SQuares Programming (SLSQP) algorithm. Using this method allows us to define equality and inequality constraints, as well as to specify the bounds of each independent variable.

VIII. EXPERIMENTAL EVALUATION

To evaluate the optimization based cartesian controller we divided experiments in two parts: simulation and real robot. For simulation we used Gazebo v7\(^4\) and rviz visualization tool\(^5\). For the real robot experiments, we used Mbot robot\(^6\) extended with a Robai Cyton Gamma 1500 Arm \(^7\).

To close the loop between perception and manipulation modules we integrated an alvar marker detection off-the-shelf component. For this purposes, we used the camera located on top of Mbot head. An alvar marker was printed and placed on top of the object to be grasped (see Fig. 5).

We performed different sets of experiments, which vary in

\(1\)http://wiki.ros.org/python_orocos_kdl
\(2\)http://wiki.ros.org/urdf
\(3\)https://www.scipy.org/
\(4\)http://gazebo_ros.org
\(5\)http://wiki.ros.org/rviz
\(6\)http://monarch-fp7.eu/
\(7\)http://robots.mobilerobots.com/wiki/Cyton_Gamma_1500_Arm
terms of the starting arm configuration (see Fig. 6) as well as target pose generation: realistic goal pose orientation vs random orientation, type of the goal pose - static vs dynamic, the area of randomized pose generation - front grasping vs side grasping, the distance of generated poses - base movement both not required vs required.

To evaluate the implemented arm controller we developed an automated test script. The automated test script was modified adequately to the test case scenario and used for both: the simulation environment and on the real robot.

The testing algorithm firstly set success and failure counter to zero. In the next step, it generates randomized poses in terms of position and orientation. In order to obtain a valid success rate of the experiment, the generated random goal pose needs to meet the criterion of feasibility. To check if the generated goal pose is reachable, the testing algorithm calls the forward kinematics module, which uses URDF\(^8\) arm model, and return True/False information. In case of returning False, the testing script goes back to the step when the random pose is generated. After generating feasible pose, algorithm sends arm to the starting configuration (candle or pregrasp) and afterwards runs arm controller. If the pose is not reached in previously defined time equal to 120 seconds, algorithm increases failure counter. Accordingly, if the goal pose was reached the timeout, success counter is increased. The testing algorithm ends when the number of attempts reaches 100.

**A. Arm Controller - Simulation**

Testing algorithm remains in the random pose generation step until it finds feasible goal pose and move to the next step of running arm controller. To increase probability of generated pose being reachable for the manipulator, we predefined area in which those poses will be generated (Fig. 7).

---

\(^8\)http://wiki.ros.org/urdf

---

**Front grasp area (green)**
Dimensions: 0.25m x 0.8m x 0.35
Distance form the ground (Z-axis): 0.3m
Distance from the center of the robot (X-axis): 0.35m

**Side grasp area (blue)**
Dimensions: 0.8m x 0.35m x 0.4m
Distance form the ground (Z-axis): 0.35m
Distance from the center of the robot (Y-axis): 0.45m

---

Fig. 7: The volume in which the goal poses were generated.

The decision, about dimensions of random poses generating areas, was based on the robot use case scenarios. The realistic use case scenario for this particular robot is placing or fetching objects from the small table or chair, therefore our experiments were adjusted to this requirements.

1) **Simulation - Random goal pose**

The first experiment carried out in simulation environment was based on random poses within the predefined orientation area (Fig. 7). Taking into consideration, that developed controller is not aiming to reach the final pose before closing the gripper, but is targeting the pregrasp pose which is an input for grasp planning module, we set following errors\(^9\) tolerances:

\[
\text{Position error : } \sqrt{e_x^2 + e_y^2 + e_z^2} = 0.01m \\
\text{Orientation error : } \sqrt{e_r^2 + e_p^2 + e_y^2} = 0.05rad
\]

where:

\[e_x, e_y, e_z - \text{error between current position and goal position accordingly on X,Y,Z axes}\]

\[e_r, e_p, e_y - \text{error between current orientation and goal orientation accordingly on roll, pitch, yaw Euler angles}\]

**Experiment summary:**

| Goal position: | random (within the predefined area) |
| Goal orientation: | random |
| Grasp type: | front and side grasp |
| Position error tolerance: | 0.01m |
| Orientation error tolerance: | 0.05rad |
| Starting arm configuration: | candle |
| Total number of attempts: | 100 |
| **SUCCESS RATE:** | 78% |

---

\(^9\)The implementation uses quaternions and quaternion error. Due to lack of time, the text is not updated.
In this case scenario, we focus only on the front grasping. Moreover, the starting arm configuration is changed for pregrasp. The key difference between this experiment and the previous evaluation are additional restrictions in terms of random pose orientation. In the following experiment, we restricted roll and pitch rotation limits correspondingly to the technical report. The limitation of the yaw rotation was increased in comparison to the suggested reasonable limitations as followed:

<table>
<thead>
<tr>
<th>roll axis</th>
<th>pitch axis</th>
<th>yaw axis</th>
</tr>
</thead>
<tbody>
<tr>
<td>between 0° and 0°</td>
<td>between -3° and 3°</td>
<td>between -90° and 90°</td>
</tr>
</tbody>
</table>

The pose error tolerance remains unchanged. The pose position randomization area was reduced to meet front grasp requirements (Fig. 8):

![Front view](image1)
![Side view](image2)

Fig. 8: The volume in which the front grasp goal poses were generated.

**Experiment summary:**

<table>
<thead>
<tr>
<th>Goal position:</th>
<th>random (within the predefined area (Fig. 8))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal orientation:</td>
<td>randomized realistic</td>
</tr>
<tr>
<td>Grasp type:</td>
<td>front grasp</td>
</tr>
<tr>
<td>Position error tolerance:</td>
<td>0.01m</td>
</tr>
<tr>
<td>Orientation error tolerance:</td>
<td>0.05rad</td>
</tr>
<tr>
<td>Starting arm configuration:</td>
<td>pregrasp</td>
</tr>
<tr>
<td>Total number of attempts:</td>
<td>50</td>
</tr>
<tr>
<td>SUCCESS RATE:</td>
<td>98%</td>
</tr>
</tbody>
</table>

**B. Arm Controller - Real robot**

In order to conduct the experiment on the real robot, the testing algorithm required several modifications. The goal pose is no longer generated by randomization but comes from the robot perceptual module. Remaining part of the automated testing algorithm remains unmodified. In this experiment, we close the loop between perception and manipulation, by using head camera of the Mbot for markers detection and obtaining the goal pose. Due to pregrasp planner module being under development process and is out of the scope of this thesis, we used fixed realistic front grasp orientation\(^{10}\) for this test. Position of the goal pose is random (Fig. 9). The starting arm configuration remains as pregrasp.

![Goal pose above the table](image3)
![Goal pose on the table](image4)
![Goal pose on the floor](image5)

Fig. 9: Different goal poses for arm controller experiments on real robot.

For the real robot evaluation we increased the error tolerance for the following two reasons:

1) In simulation we have an ideal synthetic static pose, which is published at constant rate. For real robot we have used an alvar marker detection algorithm with unstable position and publication rate. We recall that real world marker detection is subject to image noise, blur, and network delay. Because of that we have observed in our experiments an unstable (shaky) target pose.

2) Real robot manipulator joints suffer from backlash, and noisy encoder angle feedback which translates into end effector frame uncertainty once forward kinematics are computed. For that reason end effector frame was also presenting a unstable (shaky) behavior

The real robot increased error\(^{11}\) tolerance as followed:

Position error \[\sqrt{e_x^2 + e_y^2 + e_z^2} = 0.05m\] \( (13) \)

Orientation error \[\sqrt{e_r^2 + e_p^2 + e_y^2} = 0.1rad\] \( (14) \)

**Experiment summary:**

<table>
<thead>
<tr>
<th>Goal position:</th>
<th>random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal orientation:</td>
<td>fixed to (-79°,0°,-79°)</td>
</tr>
<tr>
<td>Grasp type:</td>
<td>front grasp</td>
</tr>
<tr>
<td>Position error tolerance:</td>
<td>0.05m</td>
</tr>
<tr>
<td>Orientation error tolerance:</td>
<td>0.1rad</td>
</tr>
<tr>
<td>Starting arm configuration:</td>
<td>pregrasp</td>
</tr>
<tr>
<td>Total number of attempts:</td>
<td>20</td>
</tr>
<tr>
<td>SUCCESS RATE:</td>
<td>80%</td>
</tr>
</tbody>
</table>

**C. Discussion**

The conducted experiments exposed several issues of the implemented software. In regards to simulation testing results, it was expected that in some of the cases controller gets trapped in the local minimum problem. Another thing to discuss is the timeout set to 120 seconds. In the first test case scenario poses are randomized in both: position and orientation. For some of the extremely not convenient and not realistic goal poses, the controller fails to find a solution.

\(^{10}\)Goal orientation is given in respect to the center of the robotic platform, not to the end effector

\(^{11}\)The implementation uses quaternions and quaternic error. Due to lack of time, the text is not updated.
poses, the controller needs more than 120sec to achieve the goal, what resulted with higher failure rate than in the case of the second experiment, where the poses were restricted to reasonable rotational variations. On the other hand, while the human orders the robot to perform grasping object from the table, we assume that one expects the execution time to no more than 2 minutes.

Considering the real robot testing we encounter issues which don’t apply in case of simulation environment such as arm calibration error or not reachable goal pose. Since the goal pose is not generated but perceived from the vision, modified testing algorithm no longer validate if the perceived goal pose is reachable. The last problem that caused 50% of failures on the real robot is the imperfection of perceptual module and markers detection. While the robot performs the task and gets closer to the goal pose, the end effector hardware part covers marker and the pose is no longer visible for the controller.

As the second part of the real robot experiment we tested arm following moving objects in real time, which results can be seen on the published video\textsuperscript{12}.

\textbf{D. Arm and Base Controller - Simulation}

To evaluate arm and base controller it was required to modify the automated test script. Since the main goal of the combined arm and base controller is to reach poses, that are not reachable for the arm only controller, we removed the part that validates if the pose is reachable. The remaining part of the test script is not modified. Considering experience from the arm controller testing, experiments conducted for the arm and base controller were considering only realistic orientations of the goal pose. In terms of position, we defined the area of generating the pose to ensure, that the randomized pose is not reachable without moving the base platform (Fig. 10).

The predefined areas have the following dimensions:

\textbf{Front grasp area (green)}
- Dimensions: 0.5m x 1m x 0.25
- Distance from the ground (Z-axis): 0.35m
- Distance from the center of the robot (X-axis): 0.8m

\textbf{Side grasp area (blue)}
- Dimensions: 2m x 0.5m x 0.35m
- Distance from the ground (Z-axis): 0.35m
- Distance from the center of the robot (Y-axis): 1m

1) \textit{Simulation - Front grasp}

The first experiment conducted for the arm and base controller was dedicated to the front grasping task only. Nevertheless, the distance between the goal pose and the robot is significantly bigger than in the arm controller tests, we decided to not increase the timeout and leave it equal to 120sec. The Fig.11 shows area of pose randomization. Accordingly to the predefined optimization weights, the base motion is limited to the cases, when its movement is necessary to reach the goal pose.

![Fig. 11: Volume for randomized poses for arm and base controller - front grasp.](https://youtu.be/CqsQnreHD2A)

\begin{center}
\begin{tabular}{ll}
\hline
\textbf{Experiment summary:} \\
Goal position: & random \\
Goal orientation: & randomized realistic \\
Grasp type: & front grasp \\
Position error tolerance: & 0.01m \\
Orientation error tolerance: & 0.05rad \\
Starting arm configuration: & pregrasp \\
Total number of attempts: & 50 \\
\hline
\textbf{SUCCESS RATE}: & 100% \\
\hline
\end{tabular}
\end{center}

2) \textit{Simulation - Side grasp}\n
The second experiment conducted for the arm and base controller was dedicated to the side grasping task only in the simulation environment. The position of the random pose generation was predefined with the same criterion to enforce base movement, by defining unreachable positions (Fig. 12). The timeout setting remained unmodified. The orientation’s randomization remained like in the previous front side arm and base experiment. The starting arm configuration was changed to the candle.

\begin{center}
\begin{tabular}{ll}
\hline
\textbf{Experiment summary:} \\
Goal position: & random \\
Goal orientation: & randomized realistic \\
Grasp type: & side grasp \\
Position error tolerance: & 0.01m \\
Orientation error tolerance: & 0.05rad \\
Starting arm configuration: & candle \\
Total number of attempts: & 50 \\
\hline
\textbf{SUCCESS RATE}: & 100% \\
\hline
\end{tabular}
\end{center}

\textsuperscript{12}https://youtu.be/CqsQnreHD2A
E. Arm and Base Controller - Real robot

The final stage of the evaluation was implementation and testing the arm and base controller on the real robot. The orientation of the goal pose for both, front grasp and side grasp, was fixed during the whole experiment. Such as during the arm controller test, for the real robot evaluation we increased the error tolerance due to hardware defects as followed:

\[
\text{Position error : } \sqrt{e_x^2 + e_y^2 + e_z^2} = 0.05m \quad (15)
\]
\[
\text{Orientation error : } \sqrt{e_r^2 + e_p^2 + e_y^2} = 0.1rad \quad (16)
\]

1) Real robot - front grasp

The first experiment on the real robot was dedicated to the front grasping task only. During the evaluation, we recorded video images from 3 different perspectives: rviz visualization of the real world, the image from the head camera used for markers detection and the university live camera placed in the laboratory, where the experiment was taken (Fig. 13).

![Fig. 13: Video data captured while conducting experiment - front grasp.](image)

**Experiment summary:**

<table>
<thead>
<tr>
<th>Goal position:</th>
<th>random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal orientation:</td>
<td>fixed to (-79°,0°,-79°)</td>
</tr>
<tr>
<td>Grasp type:</td>
<td>front grasp</td>
</tr>
<tr>
<td>Position error tolerance:</td>
<td>0.05m</td>
</tr>
<tr>
<td>Orientation error tolerance:</td>
<td>0.1rad</td>
</tr>
<tr>
<td>Starting arm configuration:</td>
<td>pregrasp</td>
</tr>
<tr>
<td>Total number of attempts for static goal:</td>
<td>20</td>
</tr>
<tr>
<td><strong>SUCCESS RATE:</strong></td>
<td>90%</td>
</tr>
</tbody>
</table>

13The implementation uses quaternions and quaternic error. Due to lack of time, the text is not updated.

1) Real robot - side grasp

The second part of the experiment on the real robot was dedicated to side grasping task only. During evaluation we collected same video images as in the front grasp testing (Fig. 14). This is the only side grasp experiment conducted on the real robot, therefore we fixed the goal pose orientation with a new orientation suitable for this type of grasping. The position of the goal pose is without any limitations.

![Fig. 14: Video data captured while conducting experiment - side grasp.](image)

**Experiment summary:**

<table>
<thead>
<tr>
<th>Goal position:</th>
<th>random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal orientation:</td>
<td>fixed to (-90°,0°,-0°)</td>
</tr>
<tr>
<td>Grasp type:</td>
<td>side grasp</td>
</tr>
<tr>
<td>Position error tolerance:</td>
<td>0.05m</td>
</tr>
<tr>
<td>Orientation error tolerance:</td>
<td>0.1rad</td>
</tr>
<tr>
<td>Starting arm configuration:</td>
<td>candle</td>
</tr>
<tr>
<td>Total number of attempts for static goal:</td>
<td>20</td>
</tr>
<tr>
<td><strong>SUCCESS RATE:</strong></td>
<td>75%</td>
</tr>
</tbody>
</table>

F. Discussion

Tests conducted in the simulation environment, not only had 100% success rate, but also had lower execution time than in the arm controller simulation tests. Ones can wrongly expect, that because of the significantly increased distance between goal pose and the end effector the total time of reaching the goal pose should also increase. Instead of that, thanks to base motion, the controller move the whole robotic platform into convenient for the arm configuration position, what clearly smoothed the arm motion and expedited the execution time.

Considering the real robot testing, we encounter the same issues as in the previous real robot tests, such as an arm calibration error or covering the marker detector with the robot’s end effector, what leads to loosing the target pose. The side grasping success rate is significantly lower than the success rate for front grasping. The reason for that is the starting arm configuration and the different fixed orientation of goal pose, which contributed to covering marker by robots own mechanical parts more often than in the case of front grasping. In terms of real robot side grasping occurs another limitation that doesn’t have place in the front grasp task. The robot to be able to detect the goal pose needs a marker to be in the vision of the camera. During the side grasp, robots starting head position is already close to its limits, what restrict the
marker detection. Due to this issue, the marker was out of the camera cone more often than in the case of the front grasping scenario.

IX. Results

Altogether, we conducted 7 different experiments, 310 reaching a goal pose attempts, out of which 89% were successful and 11% failed. We tested the developed software in the simulation environment as well as o the real robot, facing all hardware connected issues. We tested entirely random goal poses and realistic poses with a division for front grasping and side grasping task. To summarize reliability of the developed controller we gathered all results in the table below. Following summaries compare success rate only for experiments with the orientation goal pose restricted to the reasonable rotations roll, pitch and yaw.

<table>
<thead>
<tr>
<th>Success rate</th>
<th>Sim. arm</th>
<th>arm and base</th>
<th>arm + arm and base</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front</td>
<td>98%</td>
<td>100%</td>
<td>99%</td>
</tr>
<tr>
<td>side</td>
<td>–</td>
<td>100%</td>
<td>–</td>
</tr>
<tr>
<td>Real</td>
<td>80%</td>
<td>90%</td>
<td>85%</td>
</tr>
<tr>
<td>Front</td>
<td>–</td>
<td>75%</td>
<td>–</td>
</tr>
</tbody>
</table>

A. Discussion

Considering arm controller (without the base motion) in the simulation environment, the developed solution shows 22% failure rate. Since the velocity controller is based on a real-time feedback in a closed loop between the perceptual module and arm motion, there is no motion planning part. Having controller without motion planning results with a high risk of getting trapped in a local minimum of optimisation and fail to reach the goal pose. On the other hand, not using the Moveit! library, which is responsible for trajectory planning, arise with a faster start of the manipulator motion. Another drawback of using separate planning and execution is having “blindfolded” robot. In the presented solution the manipulator is able to follow the moving object in a real time. The first experiment conducted on entirely randomized goal poses was predictably less successful than following experiment with a reasonable orientation of generated goal poses. The arm controller experiment on the real robot was less successful than expected results, based on the simulation success rate. Using real robot entails hardware and integration issues, unknown in simulation experiment where we assume that each velocity command execution is ideal.

When it comes to arm and base controller experimental evaluation in the simulation environment, the success rate was 100%. Since the robot is able to move the base to the convenient configuration for grasping, not reachable poses are no longer existing. Being able to control the arm along with the base, not only shorten execution time but also significantly decrease the possibility of a goal pose not being feasible.

In terms of the real robot testing, as previously, the success rate decreased because of hardware and integration issues. All failures encountered in the real robot testing, were caused by the incorrect arm calibration, losing marker detection or not reachable goal pose. Failures resulted by inaccurate calibration were revealed by running Rviz real-world representation, along with the real arm execution. In the simulation software, robot reached the goal pose, while in the real world end effector was slightly shifted with respect to the marker. From this example, we can assume that robot “thinks” that the arm is in a different position than in it is in reality. Another issue was covering marker detector with robot’s hardware while execution. During running Rviz real-world representation we could easily observe that the goal pose disappears from the vision, which results with no goal pose. From this analysis, we draw as conclusion that the perception module, based on marker detection, should be improved to prevent “losing” the goal pose when the manipulator physically covers the marker. One possible solution for this problem could be done by keeping the previous target pose value, for when it disappears while running controller. This problem of losing the marker detection affected negatively our experiments, by increasing the failure rate in the side grasping real robot experiment. Because of the starting arm configuration, the probability of covering the marker with the manipulator physical part was higher than in the front grasp task. Another issue is limited camera vision. In most of the cases, we were able to turn the robotic head to keep the marker in the vision. The issue encountered during side grasp tests when the starting head position is turned into the direction of the goal pose and its already close to its limits on the left side. Last observed issue, (that contributed to higher failure rate) on the real robot testing, was missing a pregrasp planning module. In order to bypass this problem, we hardcoded reasonable realistic orientation for the end effector, at the same time increasing probability of the goal pose being not feasible (in the arm only controller).

The obtained results show, that simulation evaluation is usually more successful than the real robot testing results. We can also conclude, that using arm and base controller guaranty higher probability of reaching the goal pose, than the arm only controller. Another conclusion concerns type of the grasp. When considering the integration of the controller with the perception module based on marker detection, the front grasping is more reliable than the side grasping.

X. Conclusions

In this document, we presented a solution for manipulation control for mobile service robots in a domestic environment. We developed real-time closed-loop Cartesian controller base on optimization, where we give a target pose for the end effector. The implemented solution control both, manipulator and robotic platform motion together.

During our work, we contributed to the team controlling skills, by developing velocity controller for the manipulator. Both software, were fully integrated with the SocRob@Home team’s base and implemented on the real robot. We also delivered automated testing software, that might be used for future parameters tuning, depending on the need of the use case. In addition, the developed solution is based on the robot’s local coordinates and does not require any navigation nor mapping data.

With this work, we enhanced team’s manipulation skills not only by integrating arm and base motion, but also creating the possibility of having feedback from perceptual object detection and become able to grasp moving objects in a real time.
Having conducted various test cases scenarios, we point out some issues that need to be improved, in order to increase controller reliability such as minimizing arm calibration error or improving detection skills. The implemented controller can be fully integrated with any other robot and manipulator by setting various coefficients and uploading relevant unified robot description model.

XI. Future work

To enhance presented solution we propose following improvements that can be considered as a future work. Since the developed controller does not include obstacle avoidance, we suggest to integrate it with a collision path module that is already used in the team.

Another issue are the obstacles on the manipulator’s way. To solve this problem we suggest to add Octomap solution for mapping the Cartesian space, where manipulator motion is executed. Combining Octomap method and real-time control approach we can develop software, that is able to handle grasping in the dynamic environment. Taking into account, that robots general purpose is interaction with children in the hospital, being able to react in a dynamic environment is an advantage.

During evaluation of the controller, none of the goal poses were behind the robot. In case we would like to integrate robot with external cameras and the goal pose would be published behind the robot, there is a high probability that manipulator will heat the robot physical part. The Controller chooses the shortest way to the goal pose and not taking into consideration self-collision model of the robot. Adding this model to the optimisation would improve the reliability of the presented solution.

REFERENCES


14https://octomap.github.io/