Mobile Manipulation and Grasping of Known Objects

Diogo Rodrigues da Silva
diogo.rodrigues.da.silva@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

February 2024

Abstract

The goal of this dissertation is to propose a manipulation pipeline that makes use of Visual Servoing, a control technique that uses visual information. This method has the objective of reducing the error between a target and the robotic gripper by utilizing visual feedback. Firstly, to be able to achieve robot manipulation using Visual Servoing, visual information, such as RGB and depth, needs to be extracted from the camera mounted on the robot. The method developed and implemented starts by identifying visually the object and compute its visual center. The visual center is then used to determine an adequate pre-grasp pose. After that, the mobile manipulator moves under full-body (mobile base and manipulator) control to the pre-grasp pose, making use of Visual Servoing. Then an orientation correction of the end-effector is performed using MoveIt!. Finally, using Visual Servoing again, the mobile manipulator gripper approaches the object and grasps it. This work was developed to complete RoboCup@Home tasks that require object manipulation. The capabilities of the pipeline were tested using objects of varying geometries and grasp poses. More specifically, we tested using a tennis ball, a mug, a bowl and a spoon. Overall, the system presented a success rate of 90%.
Keywords: Robotics; Mobile Manipulation; Visual Servoing; Object grasping

1. Introduction

Robotics and robot manipulation, driven by advances in artificial intelligence, machine learning, and technology, play pivotal roles in technological innovation. Their adaptation to dynamic environments and the execution of complex, non-structured tasks, particularly object manipulation, represent a formidable challenge. This involves aggregating skills like object recognition and motion planning to enable robots to handle and manipulate physical objects through various control and learning-based techniques.

To enable the practical use of domestic service robots in unstructured environments, a robust object manipulation solution pipeline is imperative. This pipeline should encompass object recognition and pose estimation, optimal grasp position determination, and arm control for effective grasping. Object manipulation in a household setting presents unique challenges [10], including varying object placement and orientation, long distances between locations, diverse object types and appearances, as well as sensory issues like noise and clutter. Mobile service robots in such settings face constant environmental changes and the potential for unreliable estimations.

The project's overarching objective is to create an object grasping pipeline for the TIAGo robot [17], which will be integrated into the SocRob@Home team's existing ROS [18] pipeline. This pipeline is designed for use in competitive settings like RoboCup@Home [11] and household tasks. Specifically, the project aims to implement an object manipulation pipeline that leverages the arm, torso, and mobile base of the TIAGo robot, thereby enhancing motion capabilities and base movement. Visual feedback will be used for controlling both arm, torso and base motion.

2. Related Work

2.1. Visual Servoing

Visual Servoing is an image-based robot control method that uses image information to drive robot motion by minimizing the difference between desired and current poses. One of the first examples was Shirai and Inoue's [24] closed-loop system that minimized the error between a cube and the end-effector. According to Hutchinson et al. [8] the two main domains of Visual Servoing are: Position-Based Visual Servoing (PBVS) that controls the robot's end-effector position according to poses extracted from camera images, relying on precise camera calibration. Image-Based Visual Servoing (IBVS) uses image features to compute pose errors, eliminating the need for calibration but facing challenges like convergence to local minima.

Direct Visual Servoing (DVS) [26] emerged as a

calibration-independent approach, while machine learning, especially neural networks, significantly improved Visual Servoing. In PBVS, convolutional neural networks enhance object pose estimation [5] [2]. In IBVS, neural networks enable end-to-end learning [22], but they require an eye-in-hand camera setup and knowledge of the final image, which may not always be available depending on the task.

2.2. Object Grasping

Early approaches to solve the grasping problem relied on computer vision and heuristics [27], limited by predefined grasp poses. The taxonomy created by Bohg et al. [3] enables the categorization by the object's familiarity (known, familiar, or unknown). Known objects leverage prior knowledge about object geometry to estimate pose and choose appropriate grasps. Familiar objects are categorized based on common characteristics, such as shape, which can be challenging to encode. Unknown objects involve generating candidate grasps using sensory data, with approaches based on object shape, low-level features, or heuristics.

Machine learning, particularly deep learning, has transformed the field of grasping. Schmidt et al.[23] introduced a system using a deep convolutional neural network to generate grasp poses for unknown objects from depth images, with training data comprising pre-computed grasps and rendered depth images. Lenz et al. [13] improved upon this by using two deep neural networks to reduce computational complexity. The first network extracts features and generates initial grasp candidates, while the second network ranks these candidates to select the best grasp configuration. Morrison et al. [15] developed a Generative Grasping Convolutional Neural Network (GG-CNN) that predicts grasp quality at every pixel in an image, offering speed and accuracy advantages over traditional methods, achieving an 88% accuracy in household objects.

3. Theoretical Background

3.1. Frame Transformations and Kinematics

3.1.1 Frame Transformations

Frame transformations are an important concept in robotics, as they allow us to express the position and orientation of robots and their components in different reference frames. This is particularly useful when working with complex robotic systems, as it allows us to easily track the movement and orientation of different parts of the robot.

For example, if we want to express the point p_C in frame A we can multiply it by the transformation matrix between frame A and C as such

$$p_A = T_C^A p_C \tag{1}$$

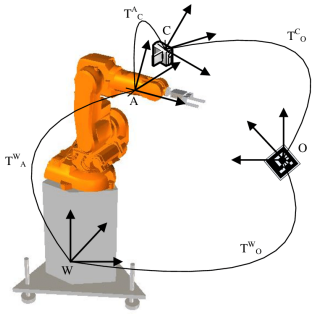


Figure 1: Robot arm with various frames corresponding to the joints [25].

3.1.2 Direct Kinematics

Direct kinematics is about determining the end-effector's position and orientation based on joint displacements and angles. Allowing us to find the end-effector's location from joint configurations.

Solving this problem involves using a kinematic model of the robot's structure. We calculate the end-effector's position and orientation by propagating joint poses through the kinematic chain.

So, if we want the transformation matrix that describes the end-effector pose w.r.t base frame [12], T_n^0, and the transformations between the n links that form the manipulator and their respective joint configurations are known, q_i, we can calculate it as follows:

$$T_n^0 = T_1^0\left(q_1\right) T_2^1\left(q_2\right) ... T_n^{n-1}\left(q_n\right) \tag{2}$$

3.1.3 Inverse Kinematics

Inverse kinematics is a critical aspect of robotics, involving the determination of joint angles required to position the end-effector at a desired location and orientation. This process is essential because it allows for control of the end-effector's position and orientation by manipulating the joint angles. This is typically achieved using a kinematic model of the robot's structure, considering its interconnected links and joints, and finding the joint angles necessary to achieve the desired end-effector pose.

However, inverse kinematics presents several challenges and constraints that may need to be taken into consideration. Such problems can include: non-uniqueness, particularly in redundant manipulators; singularities; collision avoidance; and joint velocity limits.

Various techniques exist for solving the inverse kinematics problem. These methods range from classical approaches like analytical, numerical, and optimization techniques tailored to the robot's kinematic structure and computational requirements, to machine learning-based approaches that

can simplify tasks such as singularity and joint limit avoidance.

3.1.4 Differential Kinematics

Differential kinematics involves examining the relationship between the velocities of the joints in a robot and the velocity of the end-effector. This is a crucial problem in robotics because it allows us to calculate the velocity of the end-effector based on the velocities of the joints and vice versa.

The common approach to this problem is to relate the joint and end-effector velocities, $\dot{q}$ and v_c respectively, using the Jacobian matrix, J.

$$v_c = J(q)\,\dot{q} \tag{3}$$

According to Siciliano et al. [21], v_e and $J(q)$ can be represented by their respective translational and rotational components like this

$$\begin{bmatrix} \dot{p}_e \\ \dot{\omega}_e \end{bmatrix} = \begin{bmatrix} J_P \\ J_O \end{bmatrix} \dot{q} \tag{4}$$

Thus, each column of the Jacobian matrix can be decomposed into two (3 x 1) vectors that describe the translational and rotational velocities of each joint.

$$\begin{bmatrix} J_{P_i} \\ J_{O_i} \end{bmatrix} = \begin{cases} \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix} & \text{for prismatic joint} \\ \begin{bmatrix} z_{i-1} \times (p_e - p_{i-1}) \\ z_{i-1} \end{bmatrix} & \text{for revolute joint} \end{cases} \tag{5}$$

The variables p_e, p_{i-1} and z_{i-1} are given by the transformations between joints presented in (2).

Finally, by applying the same thought process used in inverse kinematics we can invert the Jacobian matrix to produce the joint velocities for a desired end-effector velocity. Like the inverse kinematic problem this method may result in a non invertible Jacobian matrix. Instead, we need to use the Moore-Penrose inverse J^+ that guarantees that the matrix generated is unique. This matrix can be computed using singular decomposition as such

$$\begin{cases} J = UDV^T \\ J^+ = VD^+U^T \end{cases} \tag{6}$$

The matrices U and V come directly from the decomposition of J while D^+ is the matrix D where we substitute every non zero member with their reciprocal.

3.2. Visual Servoing

Visual Servoing is a control method in robotics and computer vision that uses feedback from one or more cameras to guide the movement of a robot or camera to a desired position or orientation. It involves capturing images of the environment or target object using cameras, processing these images with computer vision algorithms, and using the extracted information to compute control signals that direct the robot or camera towards the desired position or orientation.

3.2.1 Visual Servoing Taxonomy

There are several ways to classify Visual Servoing methods, depending on the specific characteristics and requirements of the application [8].

- Based on visual feedback: Visual Servoing methods can be classified based on the type of visual feedback they use, such as Image-Based Visual Servoing, which uses the positions of distinctive features in the image, or Position-Based which uses the 3D poses estimated using the image.

- Based on camera configuration: In this domain, there exists mainly two configurations that can be used: eye-in-hand, which is when the camera is attached to the end-effector and provides the point of view of the tool.

- Control Hierarchy: the type of input given to the joint controllers can be the direct output of the visual servo segment, *direct visual servo*, or it can be stabilized using internal joint feedback, this last method is called *dynamic look-and-move*.

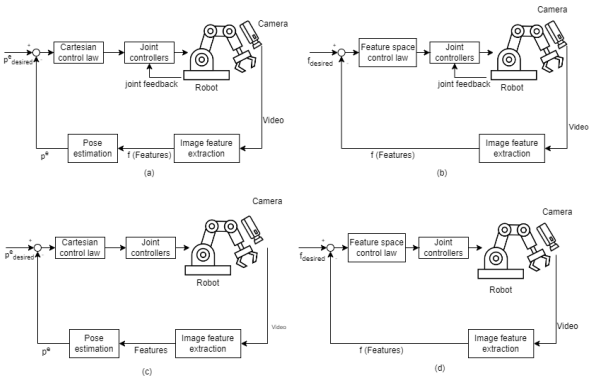


Figure 2: Control schemes according to control hierarchies and visual feedback. (a) PBVS with joint feedback. (b) IBVS with joint feedback. (c) PBVS without joint feedback. (d) IBVS without joint feedback.

3.2.2 Position Based Position Servoing

In Position-Based Visual Servoing, as mentioned in the previous sections, we use features extracted from an image to determine the pose of a target relative to the camera. By calculating the difference between the current and desired pose of the robot in the task space, we can use this information to control the robot [8]. The problem can be formulated mathematically as follows:

$$E\left(T_e^0, P_0, x_e\right) = P_0 - T_e^0 x_e \tag{7}$$

P_0 is the goal position in the robot root frame, x_e is the position of the end-effector in the respective frame and finally, T_e^0 is the transformation matrix that converts points in the end-effector frame to the root frame. This way, we can convert the position of the end-effector to the same frame as the goal position to calculate the difference between both.

To bring the end-effector to the desired position, we need to apply a velocity to it so we can utilize a proportional control law to generate this movement. Another aspect that needs to be addressed is the fact that the goal position will be detected by the camera, and thus another transformation needs to be applied to it to match the frames.

$$u_0 = -k\left(\widehat{T}_e^0 x_e - \widehat{T}_c^0 \widehat{P}_c\right) \tag{8}$$

where u^0 represents the output velocity of the end-effector in the root frame, $\widehat{P}_c$ represents the goal pose in the camera frame and k the proportional gain. The transformation matrices and goal positions are considered approximations since there may exist uncertainty in the joint's position and also uncertainty from the camera measurements.

According to [8], we can now eliminate kinematic and camera calibration errors by considering the goal and end-effector positions in the camera frame like so

$$u_0 = -k\widehat{T}_c^0\left(\widehat{x}_c - \widehat{P}_c\right) \tag{9}$$

now the position of the end-effector also is estimated from the image but since both it and the final position are in the camera frame the possible calibration error are eliminated.

3.3. Object Recognition

3.3.1 Neural Networks

Neural networks, inspired by the brain's structure and function, consist of interconnected "neurons" organized in layers. They're trained on large datasets to recognize patterns and relationships, such as classifying images based on labeled examples.

Neurons receive input from other neurons or external sources, apply weights and biases, and produce output. This output flows to the next layer, and this process continues to generate a final output. During training, weights and biases are adjusted using input-output pairs. The network's output is compared to the desired output, and the error guides weight and bias updates to improve predictions.

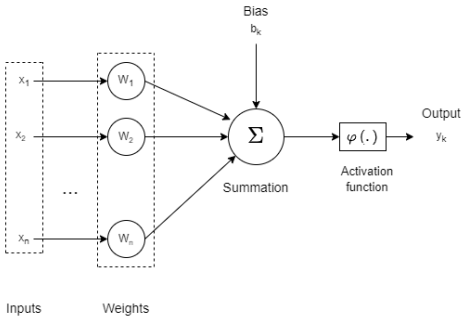


Figure 3: Architecture of a single neuron.

In neural networks, the goal is to reduce the error between the network's output and the desired output. This requires defining an error quantification function. To minimize this function, an optimization algorithm like gradient descent is chosen to converge toward the minimum loss.

To update the network, backpropagation calculates the gradient of the error concerning the neuron weights and biases. This gradient guides the update process, adjusting the weights and biases in the direction opposite to the error gradient using the chosen optimization algorithm.

3.3.2 Convolutional Neural Networks

A convolutional neural network (CNN) [16] is a type of neural network that is particularly well-suited for tasks such as image recognition and classification. It is designed to process data that has a grid-like topology, such as an image, and it is made up of layers of interconnected neurons that process and transmit information.

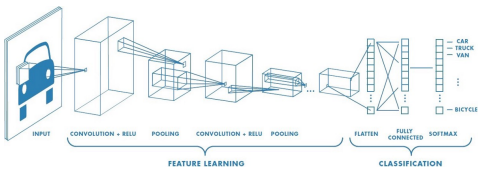


Figure 4: Example of a convolutional neural network [20].

Convolutional Neural Networks (CNNs) employ convolutional layers to extract features by applying weighted filters to local regions of input data. These results pass through a non-linear activation function to produce feature maps capturing diverse aspects of the data. Subsequently, pooling layers

downsample and reduce dimensionality, enhancing computational efficiency and mitigating overfitting.

The output of the pooling layer proceeds through fully connected layers, which leverage the extracted features for predictions.

3.3.3 Mask R-CNN

Mask R-CNN, or Mask Region-based Convolutional Neural Network [6] excels in object detection and instance segmentation, going beyond previous algorithms by introducing pixel-wise mask predictions.

Mask R-CNN identifies objects and provides detailed pixel-level segmentation masks for each one in the images. Its architecture consists of three key stages: a backbone network (often using CNN architectures like ResNet [7]) for feature extraction from input images, a region proposal network (RPN) [19] to suggest potential object locations, and a mask head for refining bounding boxes and pixel-wise masks.

Figure 5: Example of detection and segmentation [6].

3.3.4 Polylabel

Polylabel [1] is an algorithm and library for finding the ”pole of inaccessibility,” which is the farthest point inside a polygon from all of its edges. It begins by representing the polygon as a set of coordinates and initializing a grid over its bounding box. The algorithm iteratively refines the candidate point, starting with the polygon’s centroid and recursively dividing the polygon into quadrants. For each quadrant, it evaluates and selects a candidate location based on its distance from the nearest edge. This process continues until the algorithm converges to the ”pole of inaccessibility” within the polygon. Unlike the centroid, this point is guaranteed to be inside the polygon.

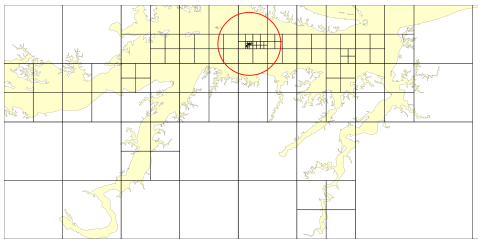


Figure 6: Visual example of the Polylabel algorithm.

4. Manipulation Pipeline

4.1. Manipulation Pipeline Overview

The pipeline starts by analyzing the environment and detecting the specified object using the perception pipeline from the SocRob@Home team. Once recognized, the visual center of the object’s segmentation is computed, and a pre-grasp position transform is published to the TF system after applying an object-specific positional shift. This triggers the Visual Servoing control loop, which calculates end-effector velocity based on the position difference between the object and end-effector, adjusting joint velocities accordingly. The robot moves until the end-effector is within a 3 cm tolerance of the pre-grasp position. MoveIt! then fine-tunes the end-effector orientation to match the object’s orientation. Subsequently, the visual servo control is reactivated to reduce the error in the end-effector and grasp position (visual center plus shift), stopping and closing the gripper when the error is below 3 cm, with a 10 cm lift to indicate a secure grasp. This approach facilitates separate control tuning for pre-grasp and grasp movements.

Figure 7: Example of manipulation pipeline. **a)** Inital arm configuration; **b)** Visual Servoing to pre-grasp position; **c)** Orientation correction; **d)** Visual Servoing to grasp position; **e)** Gripper closing and demonstration of a secure grasp.

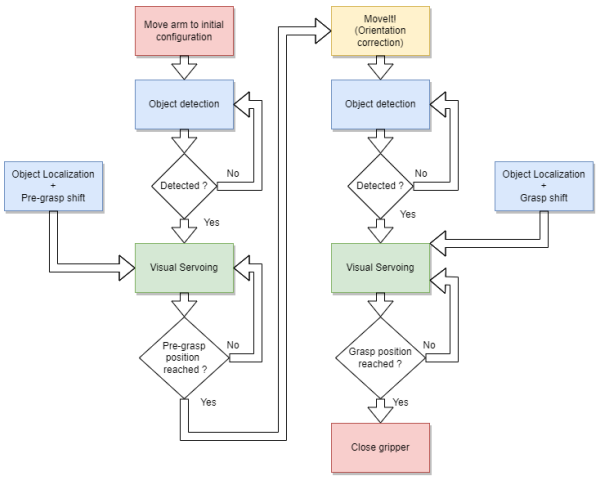


Figure 8: Flow chart describing the overall behaviour of the grasping system.

4.2. Mobile Kinematics Derivation

The initial approach one can have to the problem of mobile manipulation is to find the forward kinematics of the entire system, composed by both the base and the arm as shown in [14]. First we defined the relevant frames $\{w\}$, $\{b\}$, $\{0\}$, $\{e\}$, respectively, world, mobile base, end-effector base and end-effector frames. For the sake of simplification, when the arm joints are referred, the torso joint is also included as part of the arm. After the frames are defined, we want to find the transformation matrix that, given the current configuration of the base and the arm, q and θ respectively, relates the end-effector pose w.r.t the world frame.

$$T_w^e(q,\theta) = T_0^e(\theta) T_b^0 T_w^b(q) \tag{10}$$

where $\theta \in R^n$ is the state vector of the n joints of the arm, $q = (x, y, \theta_b)$ is the state vector of the base, $T_w^b(q)$ is the transformation of the base with respect to the world frame, T_b^0 the fixed transform between the mobile base and the end-effector base and, finally, $T_0^e(\theta)$ is the forward kinematics of the arm.

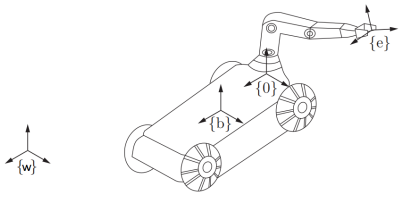


Figure 9: Frames defined by [14].

Starting with the pose of the base in relation to the world frame, we derive from inspection of both frames that the transformation matrix can be defined as such

$$T_w^b(q) = \begin{bmatrix} \cos\theta_b & -\sin\theta_b & 0 & x \\ \sin\theta_b & \cos\theta_b & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

To be able to perform kinematic control we need to be able to relate the 6 dimensional twist of the end-effector w.r.t. the world frame $\{w\}$, v_e, with the velocities of the wheels and the arm joints, u , $\dot{\theta}$ respectively. Given we are working with velocities, differential kinematics becomes the most logical approach to this problem. So, according to differential kinematics, the velocities relate as follows

$$v = J(q,\theta) \begin{bmatrix} u \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} J_{base}(q) & J_{arm}(\theta) \end{bmatrix} \begin{bmatrix} u \\ \dot{\theta} \end{bmatrix}. \tag{12}$$

To note, that the Jacobian, $J(\theta)$, can be decomposed into both $J_{base}(q)$ and $J_{arm}(\theta)$, since $J_{base}(q)u$ and $J_{arm}(\theta)\dot{\theta}$ represent the contributions of both the base and the arm to the velocity of the end-effector.

Starting with the arm Jacobian, the method to be able to calculate it has already been described in Section 3.1.4. The resulting column vectors can then be appended together to form the Jacobian $J \in R^{6\times n}$, where n represents the number of joints. Given that the arm kinematics root frame is the base frame b, the arm Jacobian needs to converted to the world frame b. To achieve that we can multiply it by the adjoint map of the transformation T_b^w.

The final step is to calculate the Jacobian of the base, $J_{base}(q)$, for that we can start by formulating the planar twist, v_b, that describes the speeds of the mobile base in $\{b\}$, also denoted as the instant velocities. Since we assume the base can only operate in the x-y plane, only the linear velocities of the x and y axes and the yaw rate need to be considered. The yaw rate is easily obtainable since it is the time derivative of the current yaw, $\dot{\theta}_b$. The velocity v_{b_x} can be decomposed into its x y components in relation to the world frame $\{w\}$, resulting in $v_{b_x} = v_{w_x}\cos\theta_b + v_{w_y}\sin\theta_b$. To find the velocity v_{b_y}, we can apply the Pfaffian velocity constraint $A(q)\dot{q} = \begin{bmatrix} 0 & \sin\theta_b & -\cos\theta_b \end{bmatrix}\dot{q} = v_{w_x}\sin\theta_b - v_{w_y}\cos\theta_b = 0$ given that we are working with a nonholonomic base and thus, there is no lateral movement.

Combining all this information together and writing it in matrix notation, we obtain the following relation

$$v_b = \begin{bmatrix} v_{b_x} \\ v_{b_y} \\ \omega_{b_z} \end{bmatrix} = \begin{bmatrix} \cos\theta_b & \sin\theta_b & 0 \\ -\sin\theta_b & \cos\theta_b & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{w_x} \\ v_{w_y} \\ \dot{\theta}_b \end{bmatrix} \tag{13}$$

Given this, we want to find the effect the inputs, v_b, have on the velocities of the base in frame $\{w\}$. To do that, we just need to invert the matrix from (13) and the result is the following

$$\begin{bmatrix} v_{w_x} \\ v_{w_y} \\ \dot{\theta}_b \end{bmatrix} = \begin{bmatrix} \cos\theta_b & -\sin\theta_b & 0 \\ \sin\theta_b & \cos\theta_b & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{b_x} \\ v_{b_y} \\ \omega_{b_z} \end{bmatrix} \tag{14}$$

To note that the column respecting to v_{b_y} is removed since it has no relevance given that the only control input we give to the mobile base are instant velocity and yaw rate. The base is not capable of any lateral movement.

$$\begin{bmatrix} v_{w_x} \\ v_{w_y} \\ \dot{\theta}_b \end{bmatrix} = \begin{bmatrix} \cos\theta_b & 0 \\ \sin\theta_b & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_{b_x} \\ \omega_{b_z} \end{bmatrix} \tag{15}$$

To add the contribution the base has on the velocity of the end-effector, we need to add rows

of zeros so that the dimension of the base twist matches the 6-dimensional twist that describes the velocity of the end-effector.

$$v_{w_6} = J_{base}(q)u \Leftrightarrow \begin{bmatrix} v_{w_x} \\ v_{w_y} \\ v_{w_z} \\ \dot{\theta}_{w_x} \\ \dot{\theta}_{w_y} \\ \dot{\theta}_{w_z} \end{bmatrix} = \begin{bmatrix} \cos\theta_b & 0 \\ \sin\theta_b & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{16}$$

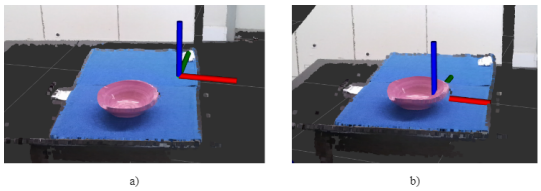
Now that both $J_{base}(q)$ and $J_{arm}(\theta)$ are known, the pose of the end-effector can be predicted as a function of the input velocities the user gives to the base and the arm.

4.3. Target and End-effector Pose Estimation

In previous sections, we've established the kinematic model for controlling both the robotic arm and base. Now, the focus shifts to determining how to infer the poses of the target object and the end-effector. To obtain the end-effector's pose, we utilize the ROS TF system, which effectively tracks relationships between various frames, including those associated with robot components and sensors. We define the gripper's transform for object manipulation, which gives us the required position and orientation relative to another frame in the robot's arm.

For object localization, we employ Mask R-CNN for identification and then extract a control reference point using the Polylabel library. This reference corresponds to the visual center of the segmentation outputted by the Mask R-CNN algorithm.

However, the visual center alone isn't an ideal grasp pose for the end-effector. To enhance the grasp position, we apply a positional shift to the object frame, creating both pre-grasp and grasp positions. These positions include predetermined gripper orientations based on the specific object, eliminating the need for orientation correction during the grasping process. This simplifies the control objective to positioning alone.



a) b)

Figure 10: Example of **a)** Pre-grasp position; **b)** Grasp position.

4.4. Visual Servo Control

The final step, to have our manipulation pipeline active, is to take the kinematic derivation presented in section 4.4 and the control law from section 3.2.2 and create the algorithm for our manipulation. To do that, we find the displacement between the target and the end-effector. After that, we want to transform the displacement to the world frame so that the resulting velocity is with respect to the world frame.

$$e_w = T_t^w h_t - T_e^w g_e \tag{17}$$

where T_t^w is the transform of the object frame to the world frame, T_e^w the transform of the end-effector frame to the world frame, and h_t and g_e the positions of the object and the gripper, respectively, in their respective frames. Now that we know the displacement between the object and the end-effector, e_w, we can apply it to the control law as such:

$$u_w = -ke_w \tag{18}$$

where u_w is the velocity that reduces the distance between the end-effector and the target and k is a proportional gain suitable to the task.

The velocity u_w now needs to be applied to the end-effector, to do that we take the inverse differential kinematic approach stated in section 3.1.4.

$$u_w = J\dot{q} \tag{19}$$

where J is the Jacobian of the mobile manipulator and $\dot{q}$ the velocities of the joints and the mobile base. To find the joints and base velocities that produce the desired end effector velocity, we take equation (3) and invert the Jacobian J. This way, we can now calculate the velocities as a function of the end-effector velocity.

$$\dot{q} = J^+ u_w \tag{20}$$

To compute the inverse Jacobian J^+, we can simply apply the method stated in equation (6). Thanks to that, even if the Jacobian is non-invertible, the *pseudo-inverse* allows us to compute it.

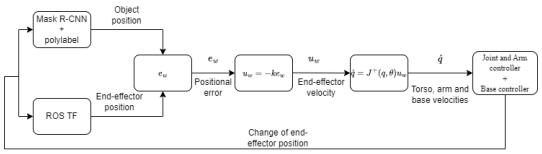


Figure 11: Visual servo control scheme.

5. Implementation

5.1. Hardware

The work utilized the TIAGo robot [17], developed by PAL Robotics. TIAGo's hardware and software serve as a foundation for experiments in perception, manipulation, navigation, and human-robot interaction.

TIAGo features a mobile base with a differential drive, a laser plane for SLAM, and safety ultrasound sensors. Its upper body includes a height-adjustable torso (110 to 145 cm) and an Azure Kinect camera in place of the standard RGBD camera. It's powered by an Intel i5 CPU, 8 GB of RAM, and a 250 GB SSD running Ubuntu 18.04.6 LTS.

The robotic arm comprises 4 M90 Module motors with 4 DoF each, employing PID control for position, velocity, and torque.

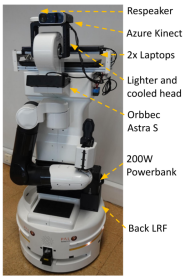


Figure 12: TIAGo robot used in this work and modifications made to the original hardware.

5.2. Software

5.2.1 ROS

ROS is a middleware framework designed to simplify robotics research and development. It offers a modular architecture that eases the creation and reuse of software components, allowing the development of ROS packages for various functionalities. ROS also provides a communication protocol for interaction between nodes (modules), enabling communication within robots and with external devices. One ROS package utilized is MoveIt! [4] because of its simplified computation of inverse kinematics. This improves the process of determining the joint's configurations necessary to move the robot's end-effector to different positions and orientations. Incorporated into ROS and MoveIt! is the OctoMap package. Its volumetric representation of the workspace is achieved by utilizing an octree grid to precisely encode the spatial characteristics of the environment. Finally, the ROS TF [18] system is used to manage and handle transformations between different coordinate frames. The main purpose of the TF system is to express the relationships between different components.

6. Results and Discussion

6.1. Experimental Setup

The tests for this project were conceived in the IS-RoboNet@Home Testbed [9] in ISR. The main purpose of the testbed is to benchmark domestic robot functionalities and tasks, using a model domestic environment. The tests utilized a tennis ball, mug, bowl and spoon, placed on top of a dining table with 74 cm of height. The metric utilized for evaluation is grasping accuracy i.e., the percentage of successful grasps over all tries. The benchmarking is comprised of the following tests: Grasping different objects multiple times in the same position (20 tries for each object); Grasping different objects and varying their positions on the table (12 tries for each object, 3 for each table position); Grasping different objects, varying their positions on the table and varying the initial position of the robot (4 grasping tries, 1 in each table position, for each of the 3 initial robot positions); testing visual feedback to correct the end-effector's position. Example grasps can be visualized in the following video of the manipulation pipeline.

The dining table will be divided into a 4x4 matrix of positions. The positions form a rectangle with a height of 20 cm and a width of 50 cm. The possible robot starting positions are composed of three different locations, forming a 1x3 matrix of positions. Each location is separated by 60 cm.

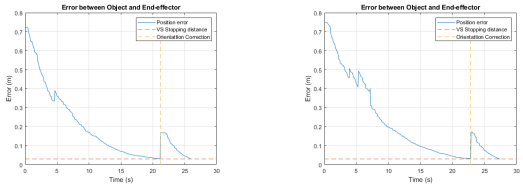
7. Results and Discussion

7.1. Grasping without varying object initial position

Tennis ball	Bowl	Mug	Spoon
100%	80%	100%	90%

Table 1: Performance by object type without varying position.

In terms of success rate per object, we can observe that both the tennis ball and the mug produce very positive results. Otherwise, the spoon has a lower rate, given that this object is naturally harder to grasp and detect. Variations in the grasp pose or in the end-effector pose estimation can cause the grasp to be performed too high, missing the spoon handle, or too low, hitting the table with the end-effector. Finally, the bowl has the lowest rate as a result of its grasp orientation being more restricted in terms of kinematic feasibility. That is, in some tests, the object position is not favourable. and thus, the MoveIt! planner can not find a joint plan to reach the desired grasp orientation.



(a) Tennis ball **(b)** Bowl

Figure 13: Error variation during grasping of different objects

Overall, the plots show similar behavior, first converging to the pre-grasp position, after that, the error increases instantly since the target position changes from the pre-grasp pose to the grasp

pose. Finally, the error decreases once again until it reaches the final grasp position. The stopping distance used for both the pre-grasp and grasp positions was 3 cm, while the proportional gain for the first servo control was 0.3 and 0.8 for the second. This usage of two different gains is also noticeable in the convergence speed. The higher gain control produces a steeper error decrease, thus resulting in a faster convergence speed comparatively.

7.2. Grasping with varying object initial position

Tennis ball	Bowl	Mug	Spoon
100%	75%	100%	83%

Table 2: Performance by object type with varying positions.

	x_1	x_2
y_1	92%	83%
y_2	92%	92%

Table 3: Accuracy for each position of the table

Continuing the analysis, the variation of object position in table 3 shows an overall positive result, not presenting any table position as more favorable. Thanks to mobile manipulation, even if the target is outside of the initial arm reach, the robot can still approach it until it is able to grasp it.

7.2.1 Grasping with varying object and robot initial position

	x_1	x_2
y_1	75%	83%
y_2	83%	92%

Table 4: Accuracy per table position

Left	Center	Right
75%	94%	69%

Table 5: Accuracy per robot starting position

Finally, observing the success rate per initial robot position, it is easily concluded that the center position is significantly more advantageous than starting in the side positions. The major problem encountered during this test was the robot colliding with the table. This mainly occurred when the robot started in a side position and was tasked with grasping the object on the opposite corner of the table. Since this method has no collision awareness, if the target is far away enough the robot will collide with the table.

7.3. Grasping object with changing position

From the plot, we observe a major increase in the error around 12 seconds, corresponding to the instant the change of position is perceived by the system. After the position is corrected, we can notice the error once again starts decreasing exponentially, behavior expected of a proportional controller, until it reaches the pre-grasp position. Following that, the system acts as expected, correcting the gripper orientation and converging to the grasp position.

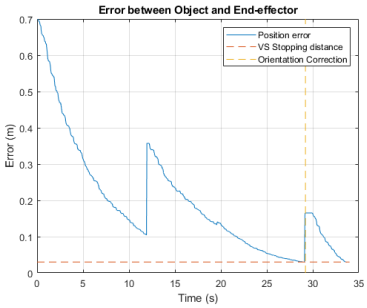


Figure 14: Error plot when object changes position.

8. Conclusions and Future Work

The dissertation aimed to develop a versatile grasping system for domestic settings, demonstrating its robustness and adaptability to object type, position variations, and robot positioning through visual feedback and mobile manipulation capabilities. The system employed machine learning to identify and locate target objects. Their poses are used in a visual servo control algorithm to guide the robot's arm and mobile base to successfully grasp objects. However, certain limitations were observed, such as orientation correction issues when the arm reached unfavorable joint configurations, slow detection and processing speeds, and a lack of obstacle awareness.

The following points can be added to improve the robustness and flexibility of the system: Improve the speed of the perception phase; Integrate a grasp pose detection algorithm; Integrate a method for obstacle avoidance; Take into account kinematic feasibility to avoid joint limits and kinematic singularities.

References

[1] V. Agafonkin. Polylabel: a fast algorithm for finding the pole of inaccessibility of a polygon, July 2016.

[2] A. Al-Shanoon, Y. Wang, H. Lang, et al. Deepnet-based 3d visual servoing robotic manipulation. *Journal of Sensors*, 2022, 2022.

[3] J. Bohg, A. Morales, T. Asfour, and D. Kragic. Data-driven grasp synthesis—a survey. *IEEE Transactions on robotics*, 30(2):289–309, 2013.

[4] S. Chitta. Moveit!: an introduction. *Robot Operating System (ROS) The Complete Reference (Volume 1)*, pages 3–27, 2016.

[5] C. Copot, L. Shi, E. Smet, C. Ionescu, and S. Vanlanduit. Comparison of deep learning models in position based visual servoing. In *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4. IEEE, 2022.

[6] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] S. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE transactions on robotics and automation*, 12(5):651–670, 1996.

[9] ISR. ISRoboNet@Home Testbed. `https://welcome.isr.tecnico.ulisboa.pt/isrobonet/`, 2014. [Online; accessed 23-October-2023].

[10] C. C. Kemp, A. Edsinger, and E. Torres-Jara. Challenges for robot manipulation in human environments [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):20–29, 2007.

[11] H. Kitano, M. Asada, I. Noda, and H. Matsubara. Robocup: Robot world cup. *XRDS: Crossroads, The ACM Magazine for Students*, 4(3):8–10, 1998.

[12] S. Kucuk and Z. Bingul. *Robot kinematics: Forward and inverse kinematics*. INTECH Open Access Publisher London, UK, 2006.

[13] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724, 2015.

[14] K. M. Lynch and F. C. Park. *Modern robotics*. Cambridge University Press, 2017.

[15] D. Morrison, P. Corke, and J. Leitner. Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. *arXiv preprint arXiv:1804.05172*, 2018.

[16] K. O'Shea and R. Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[17] J. Pages, L. Marchionni, and F. Ferro. Tiago: the modular robot that adapts to different research needs. In *International workshop on robot modularity, IROS*, volume 290, 2016.

[18] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[19] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[20] S. Saha. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-netwo` 2018. [Online; accessed 05-January-2023].

[21] P. Sanz. Robotics: Modeling, planning, and control (siciliano, b. et al; 2009)[on the shelf]. *IEEE Robotics & Automation Magazine*, 16(4):101–101, 2009.

[22] A. Saxena, H. Pandya, G. Kumar, A. Gaud, and K. M. Krishna. Exploring convolutional networks for end-to-end visual servoing. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3817–3823. IEEE, 2017.

[23] P. Schmidt, N. Vahrenkamp, M. Wächter, and T. Asfour. Grasping of unknown objects using deep convolutional neural networks based on depth images. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6831–6838. IEEE, 2018.

[24] Y. Shirai and H. Inoue. Guiding a robot by visual feedback in assembling tasks. *Pattern recognition*, 5(2):99–108, 1973.

[25] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. Kinematics. *Robotics: Modelling, Planning and Control*, pages 39–103, 2009.

[26] G. Silveira and E. Malis. Direct visual servoing: Vision-based estimation and control using only nonmetric information. *IEEE Transactions on Robotics*, 28(4):974–980, 2012.

[27] S. A. Stansfield. Robotic grasping of unknown objects: A knowledge-based approach. *The International journal of robotics research*, 10(4):314–326, 1991.