Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

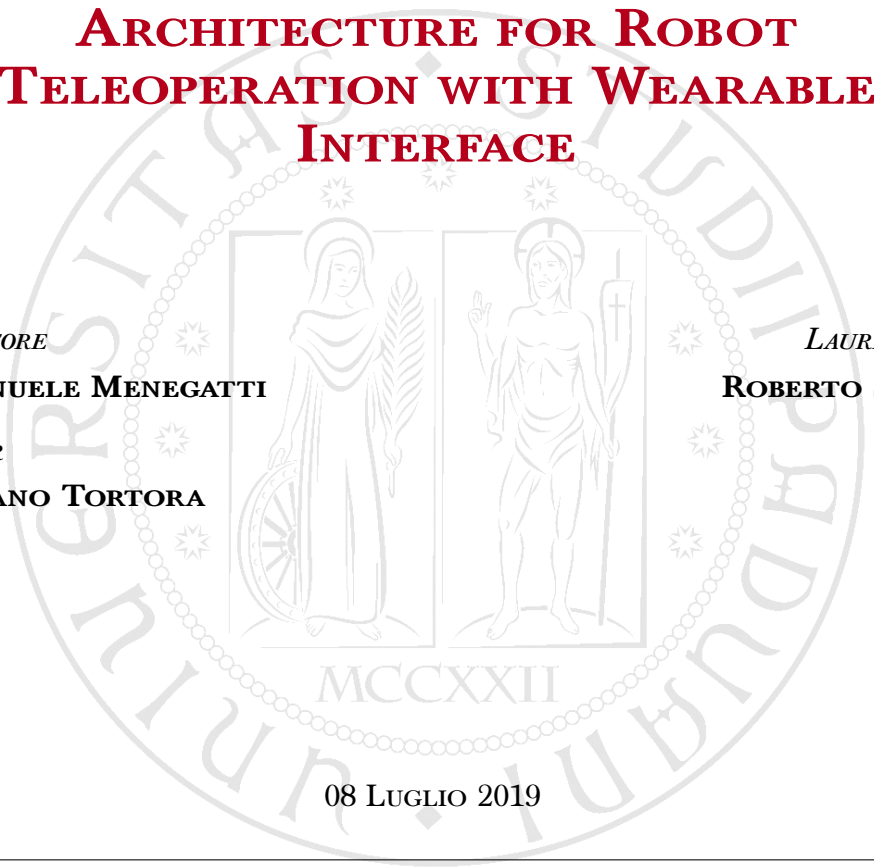
Laurea Magistrale in Ingegneria Informatica

Hybrid Shared-Autonomy Architecture for Robot Teleoperation with Wearable Interface

Relatore
Emanuele Menegatti

Tutor
Stefano Tortora

Laureando
Roberto Sassi



08 Luglio 2019

Anno Accademico 2018/2019

Abstract

Robotic system in industrial and house context with focus on manipulation has increased. The manipulation might be controlled totally by a computer system or it can be realized through devices controlled by human user, this type of control is the teleoperation, which can be difficult as it requires the user to posses total control of the robot, therefore the user is assisted with a shared autonomy. In this work a shared autonomy system which provides assistance in a free space is expanded with collision avoidance, which is important in most of real operating environment with the objective to increase its performance. The two systems are linked together with the proposed hybrid architecture, which is then tested through several experiments with computer generated input and human control together with other configuration. The results are exposed to prove the effectiveness of the system.

Table of Contents

List of Figures

1 Introduction

Robotic system have seen in the previous years a progressive diffusion in many industrial context. This is the Industry 4.0 [3], the introduction of robotics in small and medium-sized businesses where there are limitation of cost and space but also the need to automate simple task which may change from day to day. These new systems are not limited to increase working performance. Another important growing field is the assistive robotics, where robots are used to help a person with a disability to perform physical tasks of daily-living activities [4]. In both industrial and healthcare applications, the user informs the robotic system his intention of performing a certain task and the robotic intelligence plans the motion of the device to accomplish the task. The robot behavior can span from *fully autonomous*, when the robot autonomously fulfill the task once it is defined, to *fully controlled*, when the user teleoperates directly every movement of the robotic device. In this context, the concept of semi-autonomous control, also known as *shared-autonomy*, means the definition of control laws that integrates user's commands with some autonomous behaviors of the robotic device in order to maximize the efficacy of the assistance. Shared autonomy covers different aspects of robotics system, from assisting the grasping to collision avoidance in navigation task.

This work focuses on manipulation, expanding a shared autonomy system capable of target prediction and assistance to target. The expansion adds collision avoidance, to obtain increased performances. The proposed system connects assistance to target and collision avoidance with an hybrid architecture mixing together the results. The architecture has been tested with different experiments in a simulated environment to

prove the effectiveness of the system.

1.1 Background

Robotic systems have proven to be a worthwhile solution in many factory context [5]. Their size which was needed for their task also proved to be a big limitation until recent times with new types of smaller and cheaper robots. If considering the assistive robotics, the task is embedded in the context of normal human activities of daily living and would otherwise have to be performed by an attendant [6]. These ranges from social interaction, feeding, cleaning the house or other domestic activities. In all of these scenario the robots needs to understand the task and to understand the assisted as he might not be able to provide commands with high precision related to their disability or the complexity of the task or of the device used to communicate with the robot. Some of these systems might also be teleoperated by family members but they still have limited information about the environment, and the system should be able to assist into doing such tasks.

Teleoperation is used in many context where the robot provides the physical motion and the user provides the control, giving the user greater strength as robot can lift heavy weight, or allowing to operate in hazardous environment for human users [7]. When communication delays make direct control impractical or it is desired to reduce operator workload, the robot is given more autonomy for example commanding to follow a specified path instead of using direct control. Another possibility is when a robot is teleoperated by an user who doesn't have good knowledge about the environment, the robots should have some collision avoidance system [8] [9]. This also work with self-driving car when they are driven by an user but if a collision might happen, the system takes control of the car and tries to avoid the collision with response time which are faster than that of an able-bodied human [10]. Such systems were the robot has partial autonomy over its own action is defined as *shared-autonomy*.

The idea of a system with shared-control has actually been from some time [7].

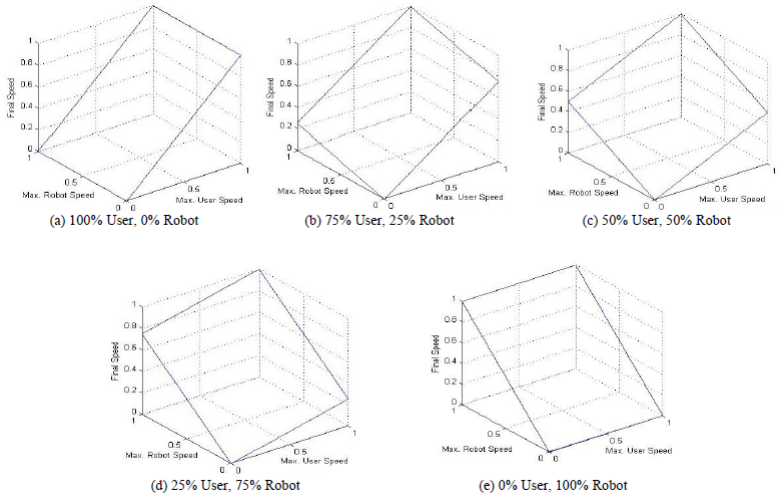


Figure 1.1: Example of sliding scale autonomy where user and robots are given a max velocity usable different in each level. [1]

More recently, with the expansion of robotic systems in a more wide use, this idea has been greatly developed. One of the first thing covered by shared control was driving assistance [1]. Moving a robot from start to goal is called navigation, and being navigation one of the most researched topic in autonomous system, made it one of the first to be researched with shared autonomy as a core aspect. One of the techniques used to achieve shared control, but also covers the navigation and collision avoidance is the potential field method [2] which can be seen in Figure 1.2. In this scenario the user command is used as a vector in a potential field generated with some free space detector (such as sonar, radar, or laser scanner). The user command is weighted by the potential field which gives as result a new command that follows the user intention but also avoid collision. Another technique of shared navigation uses the sliding scale autonomy [1]. In this case a finite number of levels are predefined, and these levels go from total control of the user to total control of the system with the system in charge of deciding which level to use. For example if a collision is avoidable with a small command but the system is not sure if it is close to the user target, the level selected is the one with most of the control to the user but enough for the robot to help avoiding it. Some key aspects of this system are deciding the number of levels, how much the control is shared in each

level, how the system decide which of them to use in the present scene.

1.2 Related Work

In this work the task is grasping where the shared autonomy covers different aspects of the manipulation: collision avoidance when moving the manipulator, assistance to target to improve grasping, and target prediction to allow continuous use. For example the assistance to target has been already seen that it could be implemented in different ways such as sliding-scale autonomy or in a continuous approach [1].

Instead of working on all the aspects of the task, it is possible to focus on one of them at a time. Target prediction could be done in two different ways: asking the user directly with vocal or visual interface, or alternatively trying to predict in a probabilistic way which goal the user wants. The explicit approach works better in discrete cases where the number of possible goals is limited but it requires the user to provide a constant feedback to the robot, thus being slow and usually needing training for the user as complexity increases. However, there are often a continuum of goals to choose from (e.g. location to place an object, size to cut a bite of food) and previous works have proved that this leads to ineffective collaboration [2] [11]. Therefore gathering implicit information about user intent is preferred, but in this case the system needs to use only the user input intended for control and from that predicting the user objective but it could be unreliable and confidence of the prediction is an important parameter which will take part in the shared control.

With the goal predicted and the confidence over that, the system can now assist the user for the movement to reach the goal and in the grasping phase. The assistance over the grasping is one of the aspects that depends on the gripper which might be very different from a human hand and also different to control for a normal operator. Therefore the shared control might take care of this part by providing the best gesture and adapt the final pose for it to be successful [12]. The grasp pose might be computed in many different ways, like a database of predefined poses or through math functions. If the

Figure 1.2: In the first image from the left, the sonar readings of free space are visible with the command of the user as an arrow. In the second the movement gets weighted with the potential field. In the third, the result can be seen moved towards the free space. [2]

focus is on the movement, there are other problems depending on the input type. The high-level interface is not always available and usually the input device is a joystick. This type of input works only on two axis but most of real task needs all the three dimension, so the joystick has different "modes" that switch between different set of planes. This mode switching is time consuming and for a proper use it requires the user to be trained or having a lot of experience with the device. Therefore a shared control system might take care of this part by switching between modes [13] reducing the difficulty of the movement.

The intervention when moving needs to be described by some policy that will use the previously calculated parameter. The policy itself can be of really different implementation, for example it describes how much the system can intervene from no intervention to a total takeover of the robot over the use, with the most common implementation taking total control of the robot when a threshold on the predictor is reached [14]. While this approach is effective at accomplishing the task, studies have shown that users often prefer having more control [15]. Potential field method has been used to push away from obstacle [2] and towards goal. In other works the user has at disposal a high-level interface, and generate the commands with various level of autonomy after the user intention is provided [16]. One possible approach is to give the user high-level visual interface. The interface might provide the user with way-points to follow, or with predefined grasping target computed from the scene [16]. In this

case the user has different options from way-points to selecting a precomputed grasping pose of the chosen object. At this point the problem of shared control becomes a standard pick problem already described in literature with many tools already developed that provides all the functionality previously described, navigation with collision avoidance and grasping. The system would be limited in providing assistance only to discrete sequential tasks. However, for a more natural control, it would be more desirable that the user provides continuous inputs to the system that should provide a greater degree of assistance when it is more confident on the user's high-level intention and decreases otherwise. This approach could be called *blending* as it merges robot and user input. Autonomous takeover, potential field methods, virtual fixtures can all be seen as *blending* methods [17]. In this sense, the system should firstly predict the user's intention from the stream of input commands and then provide a level of assistance proportional to the confidence over the prediction.

For the prediction, one possible strategy is to compute the probability for the command to be directed towards taking a specific goal [17]. At each command the system computes the distance from the future state of the robot with the current command to one of the goals. If the future state is closer to one of the goals, then that goal has a greater probability than the others. One step further to prediction is using all the commands provided by the user and not only the last one [18]. This type of system is important as it merges the prediction with the assistance over the movement allowing to do two steps together with less computation. The choice also comes from other two consideration: it can generalize what is a state and a goal, and it can give assistance over all of the possible goals when there is little confidence on prediction.

1.3 Expanding the Shared Autonomy

In the shared autonomy context, this work focus on the grasping task: the user has to move the manipulator to a desired object and take it with the robot. This is one of the most common task as it take part in almost any daily and working activities. Most of the

implementation of this system have been developed in a controlled environment: the manipulator is fixed and it has a working area where it can operate which is the common installation of these devices. To control the robot the focus is on the gripper movement as it is simpler for the user and possible with a joystick. In this work instead of a joystick the control is achieved with wearable sensors called MYO which are low costs sensors and provides inertial and muscular information of the arm. The wearer should be capable of easily providing direction information in a three dimensional space by moving the arm as command for the gripper to reach the desired object, the robot will be in charge of the grasping simplified for the user. But easier command comes at the cost of lower precision, the command is difficult to manage correctly, even more difficult if the target of the system is a physical impaired person, and that is where the shared control aid the user. The system tries to predict the user intent and correct the movement accordingly, which means less intervention from the user and faster at reaching the goal.

The assistance over the movement presented earlier lacks of the collision avoidance which is an important part in a shared controlled system and its inclusion would benefits the user. In this work the task given to the robot is manipulation and even if some of the concept of navigation still applies, they need to be revised and adapted. Reaching an object for grasping is still navigation but the obstacles now are usually more complex and the actual grasping needs to be decided correctly as it depends from the object and from the gripper mounted on the manipulator. Therefore a shared autonomy system for manipulation can manage different aspects of the manipulation task: navigation of the gripper from starting position to pick position, avoidance of obstacle on the path, how to effectuate the grip depending on the tools available and the object.

Collision avoidance is important in navigation for both mobile robots and manipulators, therefore adding collision avoidance to the system should improve the performance of this method, as not every object in the scene is a goal. The first direct improvement of the collision avoidance is safety. Avoiding collision means avoiding damage to objects and people, which is very important as the system described in some of the first example works at close contact with people. The second improvement is related to nav-

igation, the shared control might prefer a path that leads directly in collision but user aware of that will try to avoid the obstacle leading to conflicting commands. Instead if the shared control predict a collision it will try to avoid it and assisting the user in the collision avoidance, which means a shared control with avoidance is a better shared control.

There are many algorithm for the collision avoidance but the focus is on what is needed: it must be online as the user is the one in charge of the global path by providing continuously command, and it must work in a three dimensional space because it is a manipulation task. Working in a three dimensional space is in fact one of the reasons for planning ahead of the offline approach because it is computational expensive, it requires more memory to store collisions and more computational power to update direction and the path planned. That's why these offline algorithms compute the plan without considering occupation and then recompute the path only for the part with a collision, reducing total time required for the planning. Being the online requirement the most important requisite of the system, the collision avoidance implemented in this work comes from navigation of mobile robots and has been adapted to work with a manipulator.

2 Methods

The work focuses on providing assistance to the user when moving a manipulator. The system acts as in Figure 2.1. The user sees the environment and provides commands according to the goal to be reached, while the robot can model the environment by its external sensors. The inputs are given to the shared autonomy system which processes the information (user input and knowledge of the environment) providing as output a new command which assists the user to reach the target avoiding obstacles. The output of the system goes to the robot which now acts modifying the environment. The process is repeated till completion of the task.

The representation in Figure 2.1 is generic as the devices might be of different types and use different robots. Therefore the system needs to be flexible with changes. In this context, the Robot Operating System (ROS) is used as it is generic and each package can be replaced with another doing the same function. ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms [19]. The distribution of ROS used in this work is Kinetic[1].

In this chapter the components of the shared autonomy system are analyzed. At first the focus will be on a framework that provides target prediction and assistance to move within the same package, reducing time taken at each command and giving a generalized model of goal and action. Then the collision avoidance is introduced and modified to work in the robot workspace. The two packages are then connected together

[1] ***http://wiki.ros.org/kinetic***

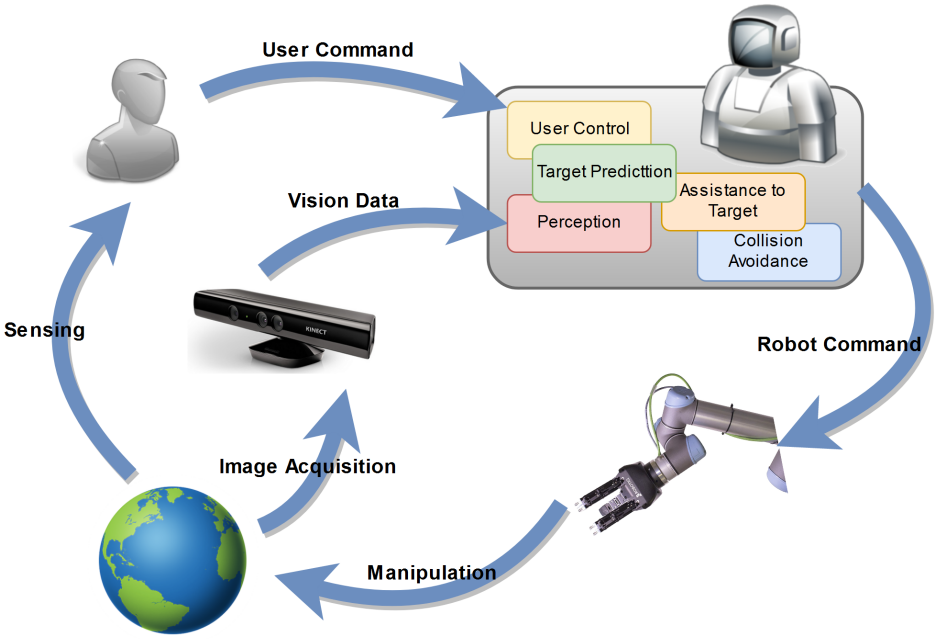


Figure 2.1: Generic shared autonomy system.

with an hybrid architecture that opportunely weights the resulted actions.

2.1 Target prediction and assistance

For the target prediction, the shared autonomy framework used minimizes a cost function with an unknown user goal. The user's goal is assumed as fixed, and the user takes action to reach the goal without considering assistance. The problem is formulated as a Partially Observable Markov Decision Process (POMDP) with the goal as unknown state, while the actions are observable.

2.1.1 Defining the framework

If the goal is known by the system, the problem simplifies as a Markov Decision Process. Formally, let $x \in X$ be the environment state (e.g. human and robot pose). Let $u \in U$ be the user actions, and $a \in A$ the robot actions. As both agents can affect the environment state, a transition is defined as $T(x'|x, u, a)$ where x' is the state result of

the actions u and a in the state x. For simplicity some assumption are introduced.

- The user has an intended goal $g \in G$ which does not change during execution. This allow the state of the system to be expanded with this goal, with new state defined by $s = (x, g) \in X \times G$. The transition function is overloaded to model the transition in environment state without changing the goal, $T((x', g)|(x, g), u, a) = T(x'|x, u, a)$.

- It is assumed that the user has a policy for each goal $\pi^u(u|s) = \pi^u_g(u|x) = p(u|x, g)$. This policy is modeled with maximum entropy inverse optimal control (MaxEntIOC) framework [20], where the policy corresponds to stochastically optimizing a cost function $C^u(s, u) = C^u_g(x, u)$.

- The user is assumed taking action only on s, the current environment state and their intended goal, and does not model any actions that the robot might take.

- At each time step, the user first selects an action, which the robot observes. The robot selects action a based on the state and user input through a policy $\pi^r(a|s, u) = p(a|s, u)$, that minimize a cost function dependent on the user goal and action $C^r(s, u, a) = C^r_g(x, u, a)$.

With this setup, the value function for a robot policy V^{π^r} as the expected cost-to-go from a particular state, assuming some user policy π^u is:

$$
\begin{aligned}
V^{\pi^r}(s) &= \mathbb{E}\left[\sum_t C^r(s_t, u_t, a_t)\right] \\
u_t &\sim \pi^u(\cdot|s_t) \\
a_t &\sim \pi^r(\cdot|s_t, u_t) \\
s_{t+1} &\sim T^r(\cdot|s_t, u_t, a_t)
\end{aligned}
\tag{2.1}
$$

From the value function it is now possible to define the optimal value function V^* as the cost-to-go for the best robot policy. The action-value function Q^* computes the

immediate cost of taking action a after observing u, and following the optimal policy thereafter.

$$V^*(s) = \min_{\pi^r} V^{\pi^r}(s) \tag{2.2}$$

$$Q^*(s,u,a) = C^r(s,u,a) + \mathbb{E}\left[V^*(s')\right] \tag{2.3}$$
$$s' \sim T(\cdot|s,u,a)$$

The optimal robot action is given by $\arg\min_a Q^*(s,u,a)$. In order to make explicit the dependence on the user, the optimal value function and the action value function are written as:

$$V_g(x) = V^*(s) \tag{2.4}$$

$$Q_g(x,u,a) = Q^*(s,u,a) \tag{2.5}$$

With the problem formulated with a known goal, it is now possible to expand it to a Partially Observable Markov Decision Process (POMDP) where the goal is unknown and the objective is the same, minimize a cost function. A POMDP maps a distribution over states, known as the belief b, to actions. It is assumed that all uncertainty is over the user's goal, and the environment state is known. This subclass of POMDPs, where uncertainty is constant, has been studied as a Hidden Goal MDP [21], and as a POMDPlite [22].

In this framework, the distribution of the user's goal is deduced by observing the user actions u. Similar to the known-goal setting value function (2.1), the value function of belief is defined as:

$$V^{\pi^r}(b) = \mathbb{E}\left[\sum_t C^r(s_t, u_t, a_t)|b_0 = b\right] \tag{2.6}$$

$$\begin{aligned} s_t &\sim b_t \\ u_t &\sim \pi^u(\cdot|s_t) \\ a_t &\sim \pi^r(\cdot|s_t, u_t) \\ b_{t+1} &\sim \tau(\cdot|b_t, u_t, a_) \end{aligned}$$

Where the belief transition τ corresponds to transitioning the known environment state x according to T, and updating our belief over the user's goal. Similar to the known goal case, it is possible to define value and action-value quantities over beliefs:

$$V^*(b) = \min_{\pi^r} V^{\pi^r}(b) \tag{2.7}$$

$$Q^*(b, u, a) = C^r(b, u, a) + \mathbb{E}\left[V^*(b')\right] \tag{2.8}$$

2.1.2 Target Prediction

In order to infer the user's goal, the distribution of user actions at state x for user goal g is modeled with a policy π^u_g. The predictor used is the Maximum Entropy Inverse Optimal Control (MaxEntIOC) [20], as it explicitly models a user cost function C^u_g, and the cost for the robot C^r_g is defined as a function of C^u_g. In this work, the user does not model robot actions, this makes possible to define a MDP with states $x \in X$ and user actions $u \in U$ as before, transition $T^u(x'|x, u) = T(x'|x, u, 0)$, and cost $C^u_g(x, u)$. MaxEnt IOC computes a stochastically optimal policy for this MDP.

The distribution of actions at a single state are computed based on how optimal that action is for minimizing cost over a horizon T. To maintain knowledge of the states action, the sequence of environment states and user inputs is introduced as ξ, note that sequences are not required to be trajectories, in that x_{t+1} is not necessarily the result of

applying u_t in state x_t.

$$\xi = x_0, u_0, \ldots, x_T, u_T$$

The cost of a sequence is defined as the sum of costs of all state-input pairs:

$$C_g^u(\xi) = \sum_t (C_g^u(x_t, u_t)) \tag{2.9}$$

It also proved [23] that minimizing the worst-case predictive loss results in a model where the probability of a sequence decreases exponentially with cost:

$$p(\xi|g) \propto \exp(-C_g^u(\xi))$$

Importantly, it is possible to efficiently learn a cost function consistent with this model from demonstrations [20]. The problem of this approximation it's the computation of $p(\xi|g)$. The difficulty lies in the normalizing constant known as the partition function:

$$\int_\xi \exp(-C_g^u(\xi))$$

Evaluating this explicitly would require enumerating all sequences and calculating their cost. However, as the cost of a sequence is the sum of costs of all state-action pairs, dynamic programming can be utilized to compute this through soft-minimum value iteration when the state is discrete [24] [25]:

$$Q_{g,t}^{\approx}(x, u) = C_g^u(x, u) + \mathbb{E}\left[V_{g,t+1}^{\approx}(x')\right] \tag{2.10}$$

$$V_{g,t}^{\approx} = \operatorname{soft}\min_u Q_{g,t}^{\approx}(x, u) \tag{2.11}$$

$$x' \sim T^u(\cdot|x, u)$$

Where the soft min function is defined as:

$$\operatorname{soft}\min_x f(x) = -\log \int_x \exp(-f(x))\, \mathrm{d}x \tag{2.12}$$

Let $\xi^{0\to t}$ be a sequence from time 0 to t, and $\xi_x^{t\to T}$ a sequence from time t to T, starting at x. The log partition function is given by the soft value function (2.12):

$$V_{g,t}^{\approx}(x) = -\log \int_{\xi_x^{t\to T}} \exp\big(-C_g^u(\xi_x^{t\to T})\big)\,\mathrm{d}x \tag{2.13}$$

where the integral is over all sequences starting at x at time t. Furthermore, the probability of a single input at a given environment state is given by [24]:

$$\pi_t^u(u|x,g) = \exp\big(V_{g,t}^{\approx}(x) - Q_{g,t}^{\approx}(x,u)\big) \tag{2.14}$$

Many works derive a simplification that enables them to only look at the start and current states, ignoring the inputs in between [25] [17]. Key to this assumption is that ξ corresponds to a trajectory, where applying action u_t at x_t results in x_{t+1}. However, if the system is providing assistance, this may not be the case. In particular, if the assistance strategy believes the user's goal is g, the assistance strategy will select actions to minimize C_g^u. Applying these simplifications will give positive feedback, where the robot makes itself more confident about goals it already believes are likely. In order to avoid this, it is ensured that the prediction comes from user inputs only, and not robot actions:

$$p(\xi|g) = \prod_t \pi_t^u(u_t|x_t,g) \tag{2.15}$$

By applying Bayes' rule it is possible to compute the probability of a goal given the partial sequence up to t:

$$p(g|\xi) = \frac{p(\xi^{0\to t}|g)p(g)}{\sum_{g'} p(\xi^{0\to t}|g')p(g')} \tag{2.16}$$

That correspond into the POMDP observation model presented, as the transition of belief over goals through τ in (2.6).

The soft-minimum value iteration is able to find the exact partition function when states and actions are discrete. However, it is computationally intractable to apply in

continuous state and action spaces, that is in the presented work. Instead, it is possible to use a second order approximation about the optimal trajectory. This approximation is possible assuming a constant Hessian [17]. With this simplification the difficult to compute soft-min functions $V_g^{\approx}$ and $Q_g^{\approx}$ are replaced with the min value and action-value functions V_g^u and Q_g^u:

$$\pi_t^u(u|x,g) = \exp(V_g^u(x) - Q_g^u(x,u)) \tag{2.17}$$

Finally, as there are often multiple ways to achieve a goal, the function are overloaded to consider them correctly. Each of these ways is referred as a target. For a single goal (e.g. object to grasp), let the set of targets (e.g. grasp poses) be $k \in K$. Each target has a cost function C_k, from which computing value and action-value functions V_k and Q_k, and soft-value functions $V_k^{\approx}$ and $Q_k^{\approx}$. Instead of considering every target as single goal it is possible to merge target of the same goal together [18]. This allows the value and action-value function for the goal to be computed as:

$$Q_g(x,u,a) = Q_{k^*}(x,u,a) \tag{2.18}$$

$$k^* = \arg\min_k V_k(x') \tag{2.19}$$

$$V_g(x) = \min_k V_k(x) \tag{2.20}$$

And the soft value functions are computed as:

$$V_g^{\approx}(x) = \operatorname{soft}\min_k V_k^{\approx}(x) \tag{2.21}$$

$$Q_g^{\approx}(x,u) = \operatorname{soft}\min_k Q_k^{\approx}(x,u) \tag{2.22}$$

2.2 SC in this work

The Shared Control package is a modified version of the one proposed in [18]. In that scenario, a human operator was provided with a joystick capable of performing movement on 2 axis of the plane (x, y). The code was developed in Python2.7 and specifically oriented to the manipulator used in the paper. The shared control node receives the twist command from the joystick, then computes the new probability using the present position of the manipulator hand and the twist command. After that, the system returns the result action that optimizes for each goal according to their probabilities. This node was developed to work with ADA packages (Assistive Dexterous Arm)[2] developed by Personal Robotics Lab[3], which uses the MICO [26] Robot Arm built by Kinova[4].

In this work there are some differences that required to modify the package to make it work in the presented scenario and make it more flexible to use. The user action, previously defined as u are twist command as it has been developed to work with joystick inputs. The state of the system x is the pose of the gripper in respect to the base_link. The user cost function, C_k^u, is defined with a simple model that proved to work well but simple to compute [18]. Let d be the distance between the robot state $x_0 = T^u(x, u)$ and target k:

$$C_k^u(x,u) = \begin{cases} \alpha & d > \delta \\ \alpha\frac{d}{\delta} & d \leq \delta \end{cases} \tag{2.23}$$

That is, a linear cost near the target $d \leq \delta$, and a constant cost otherwise. This is based on the observation that users make fast, constant progress towards their goal when far away, and slow down for alignment when near their goal [18]. For prediction, when the distance is far away from any target $d > \delta$, the algorithm shifts probability towards goals relative to how much progress the user action makes towards the target. If the user stays close to a particular target $d \leq \delta$, probability mass automatically shifts to

[2] ***https://github.com/personalrobotics/ada***

[3] ***https://personalrobotics.cs.washington.edu/***

[4] ***https://www.kinovarobotics.com/en***

that goal, as the cost for that goal is less than all others. The cost for the robot in the original system was set equal to the user: $C_k^r(x, u, a) = C_k^u(x, a)$, causing the robot to optimize for the user cost function directly.

In the original system, the SC Node worked in a modality which will be referenced as "always attracted" by the goals. In this mode the node always publish the assisted command, even if the user is not moving. This type of control has been used to simplify the controls for the user when using a joystick. Instead, the robots moves only on user command giving him more control. The latter will be referenced as "not attracted" continuously by the goals.

Having the system working in "not attracted" mode, the cost function has been modified from the original implementation, where few commands should have great impact over the predictions and the system was allowed to keep moving autonomously, thus even small actions repeated over time were enough to reach the goal. Instead, in this implementation the user is kept in charge of moving and the system provides assisted actions only when receiving new commands. That means the user provides a lot of commands that should be weighted accordingly with a small cost per command. This change in the cost function translates to a lower α than the original. After computing the probabilities, the system computes the optimal action, $\arg\max_a Q^*(b, u, a)$ which depends from the cost $C_k^u(x, u)$. As it is often impossible to calculate directly, a first-order approximation is used, which leads to following the gradient of $Q^*(b, u, a)$, which in turn have lower impact over the assisted action for the change in the parameter α. Therefore, the computed gradient have its magnitude increased by multiplying by another factor m_{far} when far from the goals, and by factor m_{close} when close to it. Two different factors are used with $m_{far} < m_{close}$, as the assistance when close to the goal should be greater to correctly reach the grasp target of the goal.

With this setup the package is already capable of providing assistance but some problem persists. What if the user already sees that the goal is correct? To provide a faster response, two alternatives are introduced which can be used alone or together. The first is to give the user a special "pick" command. The other possibility is a thresh-

old on probability. When one of the goals $g \in G$ has a probability greater than the others of at least a certain value:

$$p(g) \geq p(g') + p_{thresh} \qquad \forall g' \in G, g \neq g' \tag{2.24}$$

When the condition is met, the SC node sends the pick message to the robot controller node. In both cases the package will use the last goal published by the shared control and then will autonomously perform the grasping task.

2.3 Collision Avoidance

The collision avoidance might be done in two ways: offline and online. Most of the navigation tools for mobile robots works online and collision avoidance for manipulator is done offline. This is related to how the motion planning is done, for the manipulator planning is done before starting to move because the robot is fixed and it has knowledge of the working area, thus allows to plan ahead a collision free path [27] [28]. Instead mobile robots cannot trace all the working area and they have to deal with mobile obstacles more frequently (like people or other mobile robots). The planning is therefore done in two steps, a global planning from start to goal which avoid static obstacles like walls, and a local planning which takes care of obstacles as they appear to the robots sensors [29].

The collision avoidance selected for this work is online as the control produces twist command continuously, but being originally developed for mobile robots it has been modified to work correctly with a manipulator. As it was meant to work with mobile robot, the only collisions and commands allowed are on the plane. Even though the manipulator works in a 3d space, the bi-dimensional avoidance is enough for testing the integration of shared control and collision avoidance. From this package a possible expansion in 3D could be separating the space in multiple level over z axis, and creating for each of them a representation of that level. When computing the avoidance, the level used would be the one in which the gripper is present.

2.3.1 Creating the Costmap

For the representation of the collisions, this package uses a costmap2D that describes the environment in which the robot moves. The costmap is divided in squared cell which have their size decided at start, a cell that contains an obstacle has maximum cost per cell whereas free space has cost zero. It uses as input a TwistStamped command which can be provided by different devices (joystick, keyboard, MYO) and will be called v_{in}. At each v_{in} received, the node generates a new command that avoid obstacle which will be called v_{safe}.

The costmap needs to describe collisions in the workspace of the robot. These collisions are detected with a RGB-D camera fixed on top of the table. This type of camera provides a depth image (a pointcloud) which is useful as a representation of a 3D environment. But the system requires a costmap and therefore an intermediate step from pointcloud needs to be done. This step might be done in different ways. One possible approach is to detect the objects in the scene (e.g. shape+color, Apriltag, YOLO) and by knowing their original size it is possible to add them to the costmap as collision.

The approach, which has been used in this work, is converting the pointcloud to an OctoMap [30]. The OctoMap is a model that implements a 3D occupancy grid mapping approach, providing data structures and mapping algorithms particularly suited for robotics. The map implementation is based on an octree and is designed to various requirements with the most important for this work being three. First it has the capacity to provide a full 3D model of arbitrary environments without prior assumptions about it. The second is compactness as the map is stored efficiently, both in memory and on disk, which is important for complex environment. The third is the possibility of projecting the map to a plane, directly generating a costmap2D. The problem with the projection is that the gripper itself is seen as an object and projected as an obstacle. Also the goal are seen as obstacle and they need to be removed or the collision avoidance will push away from them. The goal collisions are removed by request of the UR_Control node, which will publish one message for each goal containing the posi-

tion of the center and the length of the side. The CA Node when receiving the message, clears the costmap in a square with the desired properties of center position and size.

Since the gripper could be detected as obstacle more passages are needed. The first step is delimiting with parameter the maximum and minimum z value on the vertical axis. Only the objects on the workspace which has known height should be considered, and with this threshold all the obstacle that might be on the ground are correctly removed. The second step is previously moving the gripper on top of the workspace above the maximum z. By applying these two steps, the produced costmap considers only objects on the table real obstacles. The third step is to set a parameter that makes the map static. The static map when subscribing to the source of data, the Octomap node, takes only the first message published, which is the first map. This is required because when the gripper will begin moving on the table and seen from the camera, the octomap will be updated and the gripper added as collision on the costmap. But with the three steps described before, the map will be correctly generated at the start and fixed. Even though these steps might be restricting, it is possible to assume that the robot is the only agent that is present in the workspace and whenever a new task is needed the system will restart regenerating the map with the new collisions.

From what described above, the costmap has a static layer which represents the obstacles in the scene. It was assumed that the robot is the only agent that is moving in the workspace and therefore there is no need to update the map. At every new task the map is generated having always the correct representation of the environment. But the static layer proved to be not enough to make the CA Node correctly working. In particular, another layer has been added: the inflation layer, which expands all the obstacles by a set amount with a scaling cost from free space to lethal. This layer is useful as the influence of the obstacles are visible in advance to the system when heading towards one of them. The scaled cost allows the avoidance command to be more flexible as it can move near the collision without being stopped but keeping the distance because it tries to maintain the minimum trajectory cost. It is also safer because the obstacle is represented by the costmap with squared cells of fixed size, affected by approximation errors.

2.3.2 Generating safe commands

The CA Node now has the correct map and is capable of creating safe commands from the user ones. Previously v_{in} and v_{safe} where defined respectively as the twist command from the user and the twist command generated by the collision avoidance. When a new v_{in} is received the node will generate multiple possible trajectories and selects the one with the minimum cost. Finally, it returns the v_{safe} which generated that trajectory. Each of these trajectories is generated by one of possible speed along the x and y axis taken from a specific set.

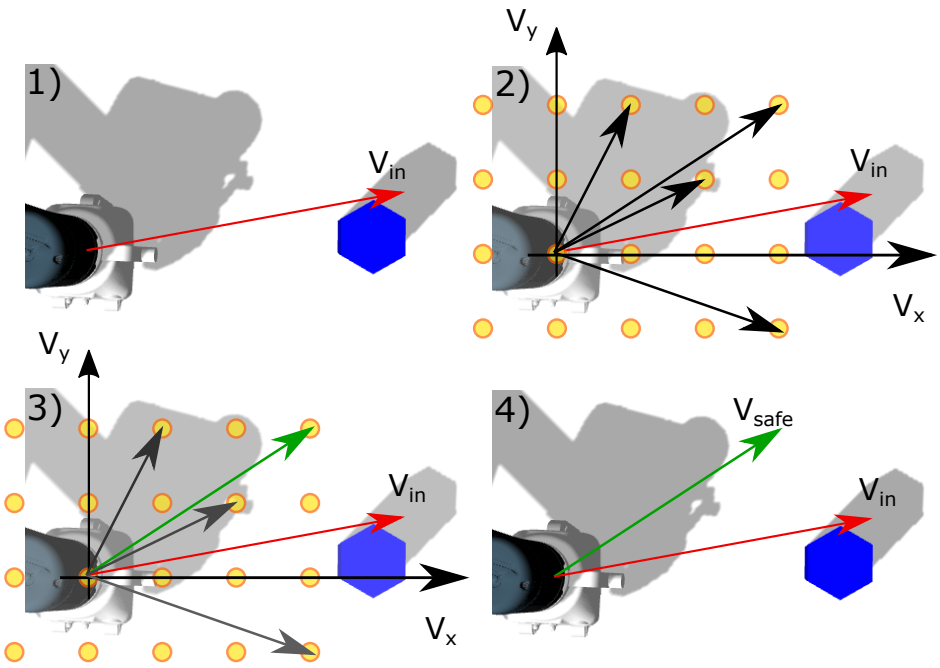


Figure 2.2: Collision avoidance in action: 1) The workspace with the obstacle visible. The user command v_{in} is in collision. 2) The set of possible of (v_x, v_y) is shown as grid of yellow circle. Some possible command are the black arrows starting from the gripper. 3) From all the possible velocity the one that minimize the cost function is taken shown as the green arrow. 4) The optimal velocity is given as the output v_{safe}.

Define v_x as the linear speed on the x axis, define the parameter v^x_{sample} as number of sample speed for the x axis, and define the parameters v^x_{max} and v^x_{min} as the maximum

and minimum speed for x. The set V^x of possible velocities on x is now computed and it contains all the possible speed from v^x_{max} to v^x_{min}, sampled evenly for a total of v^x_{sample}:

$$V^x = \left\{ v_x \,\middle|\, v_x = i \cdot \frac{v^x_{max} - v^x_{min}}{v^x_{sample} - 1},\ \forall i \in 0, 1, \ldots, v^x_{sample} - 1 \right\} \tag{2.25}$$

In the same way as v_x, v_y is introduces as the linear speed on the y axis, v^y_{sample} as the number of samples, v^y_{max} and v^y_{min} as the maximum and minimum speed for y. The set V^y of possible velocities on y is:

$$V^y = \left\{ v_y \,\middle|\, v_y = i \cdot \frac{v^y_{max} - v^y_{min}}{v^y_{sample} - 1},\ \forall\, i \in 0, 1, \ldots, v^y_{sample} - 1 \right\} \tag{2.26}$$

For each combination of possible velocity, $(v_x, v_y) \in V^x \times V^y$, a trajectory S is generated as a set of point starting from (x_o, y_o) the robot pose at this time. The points are sampled between time steps $t_{step} = t_{max}/t_{sample}$, where t_{max} is the maximum time in the future to explore and t_{sample} is the number of time sample. Therefore each trajectory S is composed of all points (x, y) generated by a given (v_x, v_y):

$$S = \left\{ (x, y) \,\middle|\, \begin{array}{l} (x, y) = (x_o + j \cdot t_{step} \cdot v_x, y_o + j \cdot t_{step} \cdot v_y) \\ \forall\, j \in 0, 1, \ldots, t_{sample} - 1 \end{array} \right\} \tag{2.27}$$

At this point there are $|V^x \times V^y|$ trajectories, from which are discarded those that have at least one point which is an obstacle. For the remaining trajectories a cost is defined. Define $C(x, y)$ as the cost of the cells that contains the point (x, y), which is provided by the costmap. Define $u_{dist}(x, y)$ as the distance of the point from the user trajectory S^u generated by v_{in}. Two parameters c_{scale} and u_{scale} are introduced, where c_{scale} is a scaling factor for the cell cost and u_{scale} is the scaling factor for the distance from the desired trajectory. The optimal trajectory S^* which is both safe (no collision)

and tries to follow the user v_{in} is computed as:

$$S^* = \arg\min_{S} \left(\sum_{(x,y)\in S} C(x,y) \cdot c_{scale} + u_{dist}(x,y) \cdot u_{scale} \right) \tag{2.28}$$

Therefore (v_x, v_y) that generated S* are the linear component on the x,y axis that defines the safer linear trajectory. This command is the normalized to have the same size of v_{in}, the result is v_{safe}.

2.4 Integrating the Collision Avoidance

In a real scenario some objects might not be goal for different reason, some of them might be obstacles present in the workspace. Other objects might be impossible for the manipulator to be grasped and therefore not to be considered as goal. In any of these cases they should be avoided as a collision might damage the robot and the object. The solution is simple and comes from the navigation problem of mobile robot: integrate a collision avoidance system. Collision avoidance is already one of the most extensively researched topic with many possible implementations. For simplicity when referencing the collision avoidance node it will be abbreviated as CA Node and the shared control node as SC Node. It is now possible to create a simple combination of the two node for the system where the output of the SC Node is given as input to the CA Node.

With this setup the system should now be able to avoid obstacle. Define v_{in} as the input twist command received from whichever device used (e.g. joystick, keyboard, MYO) and generated by the user. Define v_{out} as the final twist command which will be used by the UR_controller. Define v_{sc} and v_{ca} respectively as the output of the SC Node and the output of the CA Node. If v_{sc} is a command that would take the robot towards an obstacle, the CA Node will modify that command to avoid the collision. But this combination doesn't consider some possible combination of event that usually happens in the real case. The first problem comes from the shared control, because when computing the assisted action v_{sc} it uses both the input v_{in} which is not modified

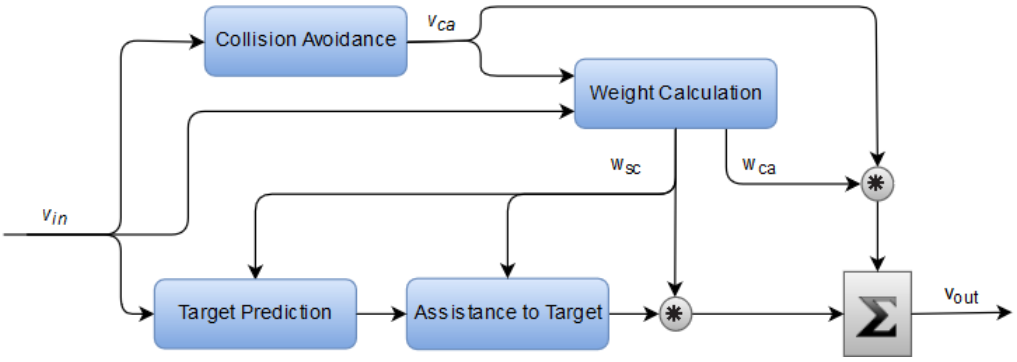


Figure 2.3: The hybrid architecture which mixes the result of collision avoidance and target prediction, based on the weights given to them.

and the position of the gripper. The position on the gripper depends on the previous commands sent to the UR_Controller which were modified by the collision avoidance, therefore the probabilities associated with the goals change without reflecting the real user intention, nor the assisted action which is considered by the model. In the case where the user has only simple commands or he doesn't understand that the robot is avoiding an obstacle, the result will be that the probabilities won't describe correctly the past commands at the end of the avoidance, and the robot might be assisting to the wrong goal. If the user is actively avoiding the obstacle, he will try to correct the path after the avoidance but it will require more time to select the correct goal, because he will need to contrast the wrong assisted action.

But there is another problem with this setting. Consider a possible scenario where the shared control points at an object behind the obstacle but the collision avoidance tries to point far from it. The two commands keep opposing each other resulting in the robot to get stuck in front of the collision object. If the user notice this behaviour, he has to provide more commands to solve the conflict, thus more intervention from the user is required. This means the shared control is not efficiently assisting the user.

Therefore the system is modified by introducing an hybrid architecture (Figure 2.3) in which the two commands are mixed together with a weighted sum. The two weights are called w_{sc} for the shared control and w_{ca} for the collision avoidance. The value of

w_{ca} is based on how much the collision avoidance is intervening to avoid the obstacle. To measure this intervention the factor chosen is the angle between v_{in} and v_{ca}. As the two commands are TwistStamped, it is possible to consider the linear component as a vector in 3D space. The angular component might be added but it is not considered in this formulation. Therefore being the two vectors v_{in} and v_{ca} with three component, the angle is defined as:

$$\alpha = \arccos \frac{v_{in} \cdot v_{ca}}{\|v_{in}\| \cdot \|v_{ca}\|} \tag{2.29}$$

This formulation of angle between vectors returns a result in the range $[0 : \pi]$. With the angle ready, it is now possible to map different angles to different weights. The mapping has been chosen with some properties. The w_{ca} when the angle is over a threshold α_{thresh} should be equal to one or as close as possible to one; the mapping should smooth the convertion from angle to weight; the point at which the two weights are equal is decided by h. The function chosen is the sigmoid function:

$$w_{ca} = \frac{1}{1 + \exp\left(-k(\frac{\alpha}{\alpha_{thresh}} - h)\right)} \tag{2.30}$$

This weight will be in range $[0 : 1]$. The other weight w_{sc} is obtained from w_{ca} with (2.32), and the mixed output is now computed as the weighted sum:

$$v_{out} = v_{sc} \cdot w_{sc} + v_{ca} \cdot w_{ca} \tag{2.31}$$

$$1 = w_{sc} + w_{ca} \tag{2.32}$$

The mixed output describes different scenario where the CA Node is active:

- The user is moving towards the obstacle. The CA Node detects that there will be a collision and starts generating a new command with high deviation from the original path. The resulted weights are in favor of the avoidance as it needs most of the control.

- The user is avoiding the obstacle but pass nearby it. The CA Node detects that

it is near a collision and it would be better to move a bit farther from it. The command is different from that of the user, but has deviation lower than the previous example. The resulted weights are in favor of the shared control as collision avoidance is less important but the shared control still benefits from it.

- The user is correctly avoiding the obstacle maintaining distance from it. The CA Node doesn't detect possible better path as the original one is moving in free space, therefore no angle with the original command. The resulted weights are with total control for the SC Node, as if there were no collision avoidance.

The intervention from the CA Node affects also the probabilities. If the user is moving towards an obstacle, the mixed v_{out} will push away the gripper correctly but the probabilities, as said above, are dependant on the position of the gripper itself. Therefore, it is possible to assume that the value and action-value function are not correctly describing the system changes from user action u. At this point two possible course might be taken to fix the update. The first method is to verify if the collision avoidance is active, and in that case stopping the update. The second method is to apply a weight to the update such that, when the collision avoidance is active the changes in probabilities are proportional to the intervention of ca. By having this intervention already measured and weighted, there is no need of extra computation. To decide which is better, some assumption on a real scenario are needed. It is possible to assume that the user will try to avoid the obstacle, his commands will be mixed and directed away from obstacle, but still up to some degree to the desired object. It is also possible to assume that if the user command doesn't avoid the obstacle, the CA Node intervention will be higher and will affect the position of the gripper, which is the state x. Therefore the second method is preferable as it describes better what happens in the system, the weight given to the update is proportional to the intervention. The computation of the update at time t is given by equation (2.17) and overloaded as:

$$\pi_t^u(u|x,g) = \exp\left(w_{sc} \cdot \left(V_g^u(x) - Q_g^u(x,u)\right)\right) \tag{2.33}$$

2.5 Final scheme

The nodes that provide the described function above, are linked together with the following scheme: The OctoMap node that creates the costmap of the collision avoidance

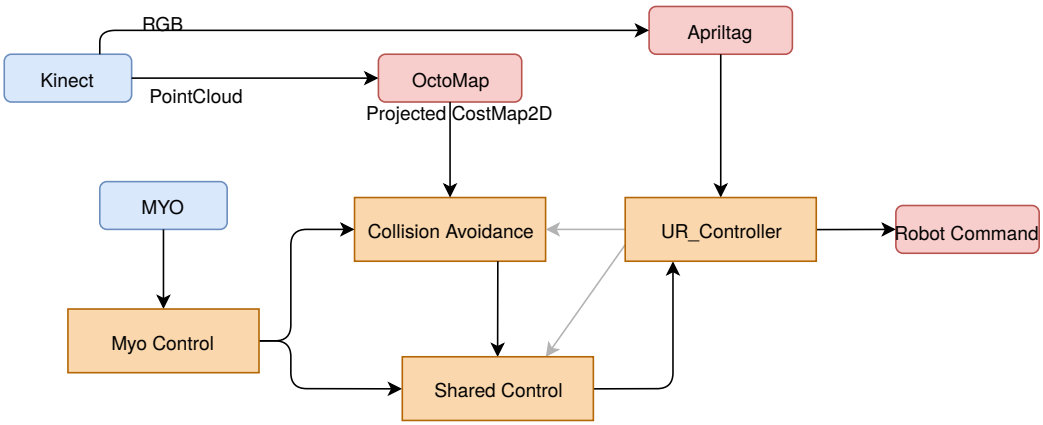


Figure 2.4: Scheme with nodes of the project.

is linked to the Collision Avoidance. The UR_Control with the AprilTag detects the objects and creates the goal with targets that will be used by the target prediction. It also sends to collision avoidance the position of the goal so that it is removed from the costmap. The MYO Control is the node that converts input from MYO [31] [32] to velocity commands used by the others. This node might be replaced with any other input provider, such as keyboard, joystick or also the input simulator seen in the next Section 3.2.

3 Experimental Validation

3.1 Experimental Setup

The system has been tested on a simulated environment seen in Figure 3.1. The manipulator used is the UR10[1] with the Robotiq 3-Finger Gripper[2]. The goals chosen are two cubes with the apriltag id on top of them. They are just two because the validity of the target prediction has already been tested in its proposal work [18] and they are both cubes for the same reason, with the only target available for the grasping as the pick pose on top of them. The manipulator is placed few centimeters from the top of the table. On the other side, the RGB-D camera, a Kinect[3] has been placed. The collision objects are prisms with hexagonal base usually approximated as cylinder and two parallelepiped of different sizes. Even though one collision object and two possible goals might be a simple scenario, it is enough to verify the improvement of the system with the hybrid control from the two nodes, Shared Control and Collision Avoidance.

Together with the hybrid architecture system, others combinations of nodes have been tested. The configurations with only the shared control and the non-assisted configuration provide baselines for the others. The direct configuration where the output of the first node is given as input to the second is the simplest approach and the hybrid configuration is the one proposed in this work and seen in Figure 2.3.

To provide a solid base line for the experiments, consistency of inputs is needed

[1] ***https://www.universal-robots.com/it/***

[2] ***https://robotiq.com/***

[3] ***https://developer.microsoft.com/it-it/windows/kinect***

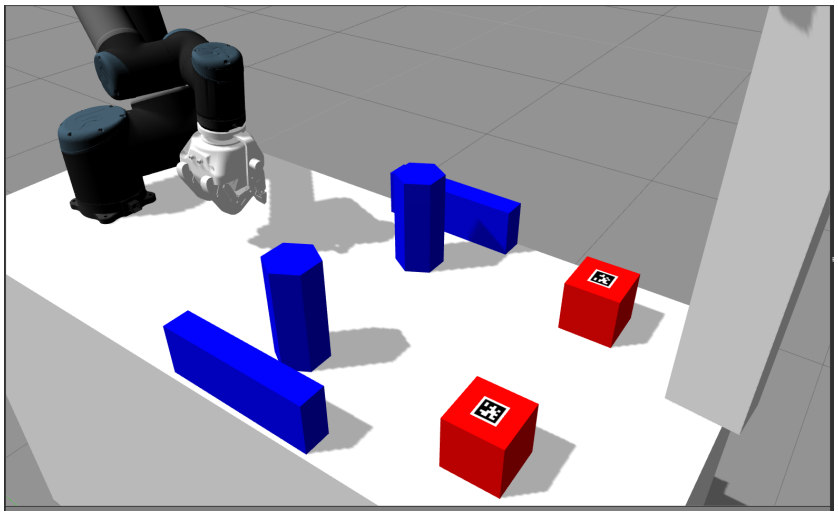


Figure 3.1: The simulated environment used in one of the test. The obstacles are in blue while the goals in red. The gripper is placed just above the table and cannot be controlled directly in the vertical direction, thus collisions might happen during the experiments.

between tests. They must be similar in how they try to reach the goal, and they must be similar to those provided by a human user. These consistency would be difficult to achieve for human operators, therefore they have been simulated by an appropriate node which maintains the described properties over the various experiments. With the results of the various architecture and motivated the choice over the hybrid, other experiments were conducted on human volunteers with the MYO.

3.2 Simulated Input

The commands from the user might be provided with different tools, but when the input is provided by a human user it is difficult to replicate the test with the same sequence of commands or requires a great number of subjects. Therefore a special simulated command is used which place the focus on the objective while maintaining a realistic sequence of commands. To achieve these objectives, a new node has been added. At first this node takes the position of the selected goal $p_{goal} = (x_{goal}, y_{goal})$ and the position of the gripper in the workspace $p_{grip} = (x_{grip}, y_{grip})$. Only the coordinates

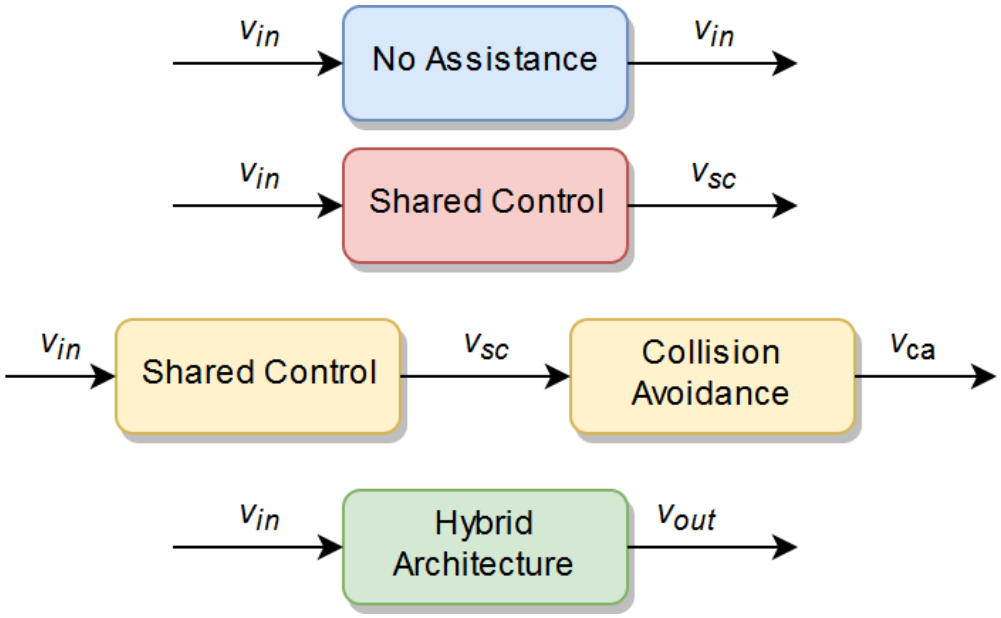


Figure 3.2: The tested architecture of nodes are listed. *No Assistance* is used only with the test with user input where the user has direct control of the robot. Shared Control that provides only target prediction and assistance to target. Shared Control with Collision Avoidance in direct configuration where the output of the first node is the input of the second called *SC to CA* in the experiments. The last combination is the *Hybrid* architecture seen in Figure 2.3.

(x,y) are considered because the collision avoidance doesn't use the third dimension, as explained in Section 2.4. Therefore the vector from gripper to goal is considered. This vector can be easily computed and provides the direction of the command from start to end that in a collision free environment would results in the correct movement to reach the goal. Previously the error over the command where exposed as the user won't be able to provide such precise direction, therefore an error over the direction is added to better simulate the user. The error in direction translates to an error over the angle of the vector, therefore it is simple to generate a random error e resulting in the angle of the command ϕ_c.

$$\phi_{goal} = \arctan\left(\frac{y_{grip} - y_{goal}}{x_{grip} - x_{goal}}\right) \tag{3.1}$$

$$\phi_c = \phi_{goal} + e \tag{3.2}$$

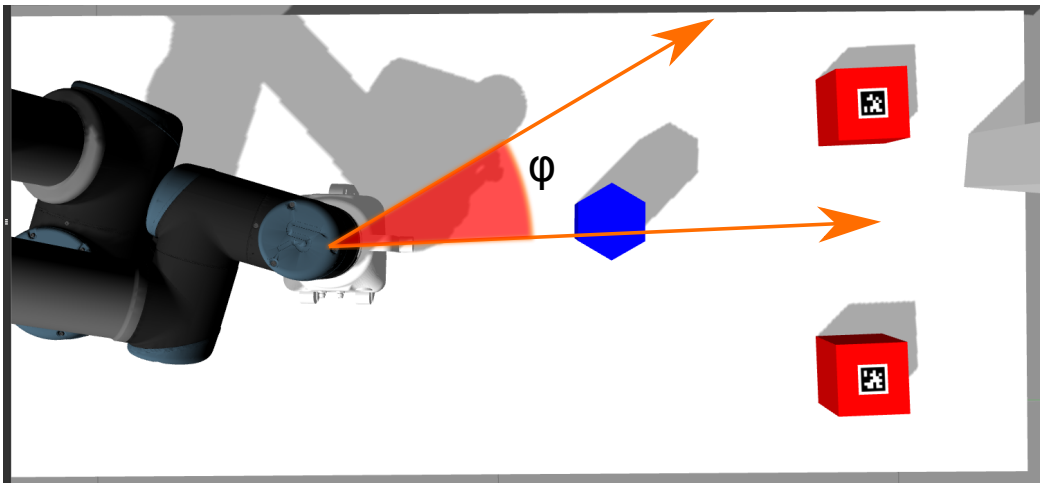


Figure 3.3: With target "box0", the angle ϕ of direction ranges based on the added error.

The error e should be always less than the maximum angle that would point towards the wrong goal, which would be impossible to counter for the system without having explicit information about user intention.

$$\max(e) < \frac{|\phi_{goal1} - \phi_{goal2}|}{2} \tag{3.3}$$

With the maximum possible error found, the error can now be computed to be of fixed size:

$$e = i \cdot \epsilon \qquad\qquad i \in \{-2, -1,\ 0,\ 1,\ 2\} \tag{3.4}$$

In such way and with a fixed ϵ (in the experiments $\epsilon = 7°$ degrees) there are always 5 cases per experiment, with the same deviations from the original direction. They are easily computed and simulate an user with low control over direction but enough to point at the right goal. The components of the command v_{in} are computed:

$$c_x = v_o \cdot \cos(\phi_c) \tag{3.5}$$

$$c_y = v_o \cdot \sin(\phi_c) \tag{3.6}$$

3.3 Control with MYO

The MYO provides both the EMG signals and the sensor orientation. The EMG signals are analyzed by the classifier provided by the MYO software, which is capable to identify different hand gestures. The gestures used are "Double tap" and "Fist". "Fist" has been used to command the robot to pick the target as it resembles closing hand to grasp an object. The "Double tap" is the easiest to identify for the classifier, therefore it has been used as safety command that start or stop the control. The orientation has been used to generate the movement direction. The command needed by the SC Node and CA Node is a velocity command, therefore a conversion from orientation to twist is needed to correctly map the user input.

3.3.1 From orientation to velocity

The orientation provided by the MYO is computed respect to a global frame and therefore it cannot be used directly. Thus, it is necessary to change the reference system from global to user's reference frame. Define q_{in} as the quaternion that represents the orientation at a specific time, the first quaternion received by the MYO Control is considered to be in a neutral pose for the user, with his arm extended in front of him, and it is saved as q_{start}. The difference between the present orientation and the neutral orientation is therefore the orientation computed from the user reference system. Being the orientation expressed in quaternion the computation of q_{arm} is:

$$q_{arm} = q_{in} \cdot q_{start}^{-1} \tag{3.7}$$

The computed q_{arm} is the correct orientation of the arm with respect to the user. If the user moves from his starting position the correction no longer holds as q_{start} was the orientation of the arm in the original pose. For this reason the "Double tap" has been used as start/stop command. Whenever the control is restarted, the actual orientation of the arm is assumed as neutral and saved as new q_{start}.

The arm orientation obtained at the previous step, is now converted to its Euler's representation in roll, pitch, yaw. Each angle is mapped to one of the three axis (x,y,z). Therefore there are three tuple a, γ where:

$$a \in \{x, y, z\} \tag{3.8}$$

$$\gamma \in \{\text{roll}, \text{pitch}, \text{yaw}\} \tag{3.9}$$

For each (a, γ), a threshold γ_t over the angle γ is defined. With the threshold, the velocity over that axis a is defined as:

$$v_a = \begin{cases} v_o & \gamma \geq \gamma_t \\ 0 & -\gamma_t < \gamma < \gamma_t \\ -v_o & \gamma \leq -\gamma_t \end{cases} \tag{3.10}$$

With v_o being the set base speed. The three velocity obtained are the components of v_{in}, which is then published for the other nodes to be used:

$$v_{in} = (v_x, v_y, v_z) \tag{3.11}$$

The velocity obtained in such way has in total eight possible directions, those along with the axis and the four bisectors. Thus proving what has been said previously, the user has low control over the direction and the system should be tested with some error, validating the method used to generate the error in the simulated input.

3.4 Performance with Simulated Input

In the first experiment with simulated input, the command has been generated with frequency of 10Hz, while the frequency of the SC Node was 60Hz which was also set in "always attracted" thus moving with the last command received. The SC Node has been set with both the "pick" command and the threshold over probability disabled.

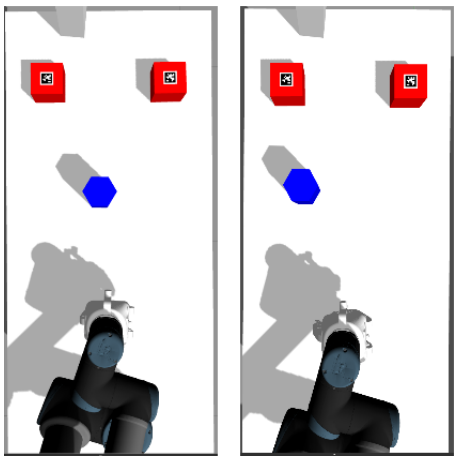


Figure 3.4: The workspace for simulated input. The left configuration has the obstacle closer to the center, instead the right configuration has the obstacle closer to the side of the table. The cube on the left has been called ”box0” while the other is ”box1”. Those names have been maintained in every tested environment.

The task is considered completed when the distance from the center of the gripper to the goal is less than a certain threshold. If after 200 command the goal has not been reached, the test is considered failed. The limit on the number of generated commands corresponds to twenty simulated seconds. The workspace for simulated input contains two cubes which are the goals, and one obstacle placed between the gripper starting position and the goals. The obstacle is taller than the cubes and therefore collisions might happen. The obstacle has been placed in two position, one closer to the center and one more on the side as in Figure 3.4.

In Figure 3.5, the obstacle is represented as a black circle, with the collision distance around it. As it can be seen the only case with collision is *SC Only*. Obviously with no collision avoidance and simulated input towards the goal it was impossible for the system to avoid collision, but it shows how the assistance works and the trajectories are correctly directed towards the goal.

In both of the cases where the output was directly connected to collision avoidance, the obstacle is avoided and no collision happens but it shows another event. To avoid the obstacle the collision avoidance tries to push to one of the sides but the shared control

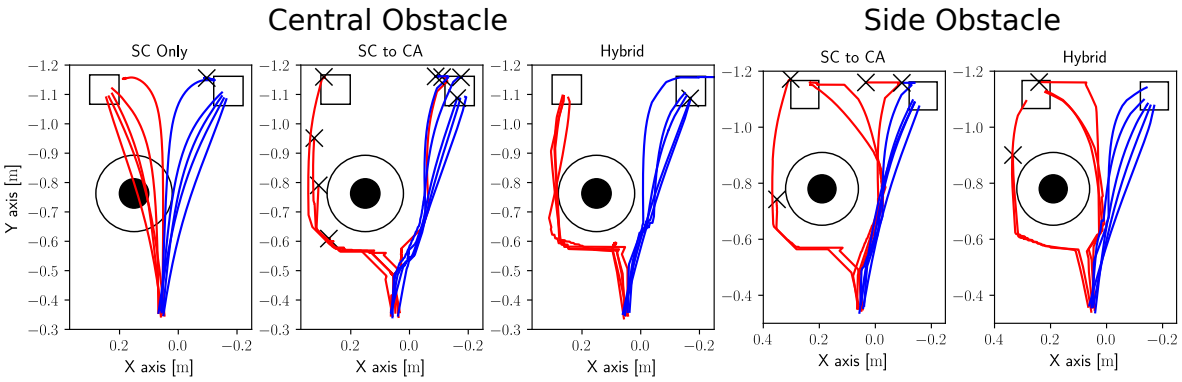


Figure 3.5: Trajectories of the gripper in the workspace with different configuration of nodes: *SC Only* with no collision avoidance, *SC to CA* for direct configuration, and Hybrid for hybrid configuration. The selected goal is shown with the color of the path: red for the goal "box0" and blue for goal "box1". The trajectories that reach the time limit finish with a cross.

tries to reach the goal behind. The resulting behaviour is a conflict between the two nodes that keeps the gripper in place, allowing only small movement to avoid the obstacle, which also require more time than the previous configuration and reaching the time limit. In those trajectories that manage to avoid the obstacle on the right side, the path are then steered towards the opposite goal, as the avoidance moved the gripper closer the incorrect target. With the constant simulated input this behaviour cannot be avoided. In presence of a human user, he could eventually correct the behaviour, but it would require more input and time to reach the goal.
In the last case the *Hybrid* architecture is shown. The obstacle is avoided in any path and the goal is reached in most of them. The wrong selection of goal is not present even in the most difficult path, even though it requires more time to reach the goal.

In Figure 3.6 the times for all simulated input are represented. It can be noticed that with the direct architecture the time limit was reached dominating the mean. The *Hybrid* and *SC Only* have similar times and they are both faster than the direct configuration. In the side obstacle experiments the same behaviour between *Hybrid* and *SC to CA* is observed. The higher times for "boxo" are dependent from the obstacle which is moved closer to that goal.

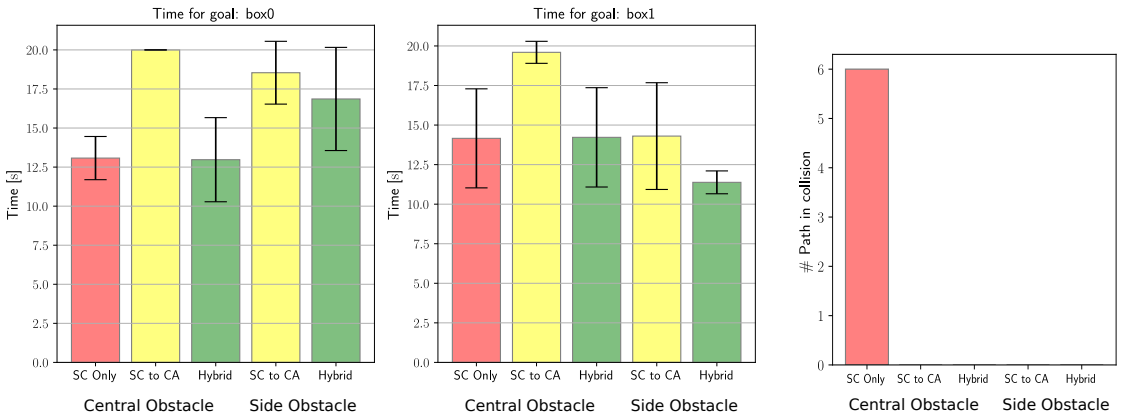


Figure 3.6: Average times for first simulation divided in two plot one for each goal. The order of the bar is kept with Central Obstacle: *SC Only*, *SC to CA*, *Hybrid* then Side Obstacle: *SC to CA*, *Hybrid*. On the right side the number of collision for each of them.

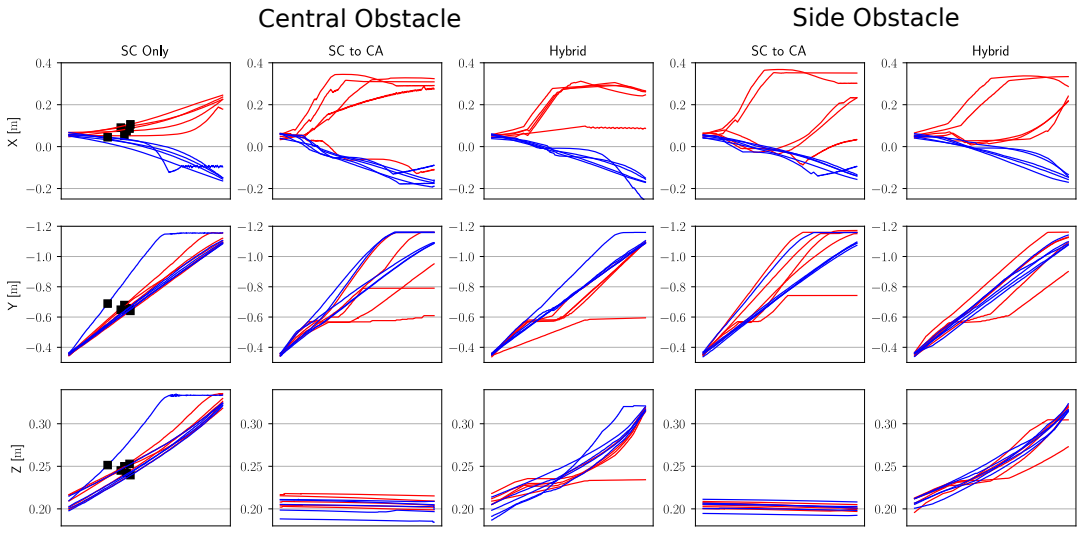


Figure 3.7: The position values of the gripper are shown. The collision are marked with a black square. All sequences have their time length normalized.

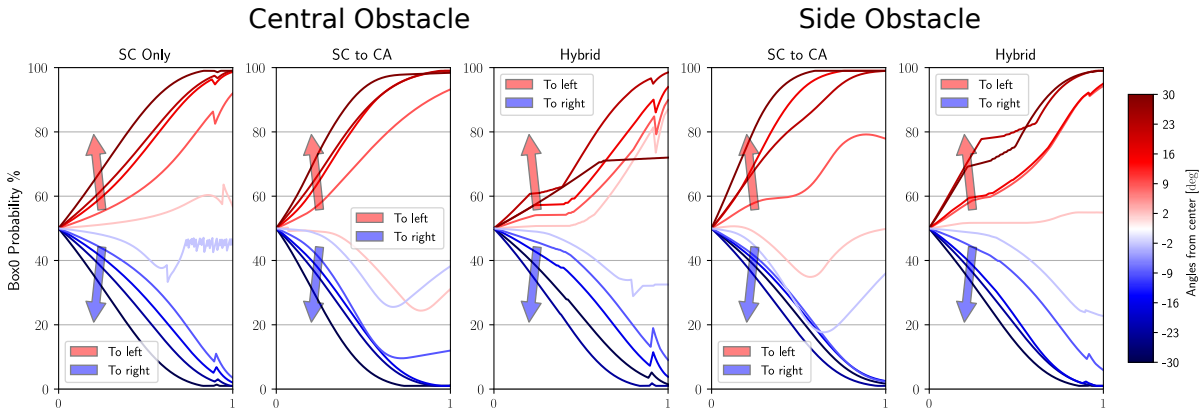


Figure 3.8: Probabilities of goal "box0" over time. The times are normalized to have same length. The probability of "box1" is not shown as it is complementary to "box0".

One important aspect of the assistance which could not be seen in the path of Figure 3.5 is the assistance over the vertical axis. In Figure 3.7 the z axis corresponds the vertical direction which is important as the target is placed on top of the goal but the collision avoidance comes from navigation and it cannot move on that axis. This limitation from the collision avoidance results in reaching the target with an error, or in some cases colliding with the object.

Lastly for this simulation, the probabilities over time are represented in Figure 3.8. Those tests with the path closer to the middle of the goals, proved to be the most difficult for the system to understand the user goal. In some cases the wrong goal has been selected requiring more time to correct the probability. These cases can be seen in Figure 3.5 where some trajectories reached the wrong target. The Hybrid configuration showed more resistance at this behaviour maintaining the correct goal selected over time.

The previous simulation covers the type of user which select the goal first then provide just one intention to which goal he wants (one direction at start), but the user could be of another type. If the user has continuous control, while the robot moves towards the goal he is capable of updating his intention based on the position of the robot. This type of user could be simulated with a new type of input where at each step the new direction is computed and then the new command is published. This type of

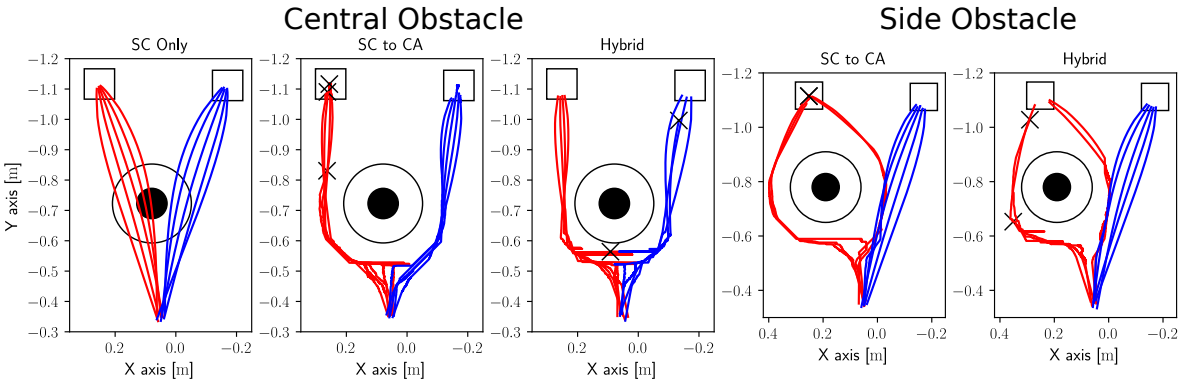


Figure 3.9: Trajectories of the gripper in the workspace with different configuration of nodes: *SC Only* with no collision avoidance, *SC to CA* for direct configuration, and *Hybrid* for hybrid configuration. The selected goal is shown with the color of the path: red for the goal "box0" and blue for goal "box1". The trajectories that reach the time limit finish with a cross..

input has been tested on the previous environments. The "always attracted" mode has been disables and the number of command generated is 1200 with a frequency of 60Hz, therefore the total time given for a task is 20 seconds as before.

In Figure 3.9 the trajectories are shown, and similar to the previous paths in Figure 3.5 the configuration *SC Only* alone is the only one colliding with the obstacle. The direct configuration for central obstacle shows three cases where the time limit is reached even though it seems the contrary from this perspective. This behaviour depends from the target which is placed on top of the goal, forcing the assistance node to intervene over that axis as seen in Figure 3.11. The two configurations, *SC Only* and *Hybrid*, seem to have similar performance in reaching the goal and avoiding the obstacle.

The average times in Figure 3.10 manifest the same behaviour between times of the previous simulation (Figure 3.6). The *SC Only* configuration is the fastest, and the *Hybrid* configuration is faster than the direct configuration even with the different input of this simulation.

The new input type showed more reliability in targeting the correct goal. In Figure 3.12 the probability for boxo are reported, and in all the test the correct goal has been selected at start and maintained throughout time.

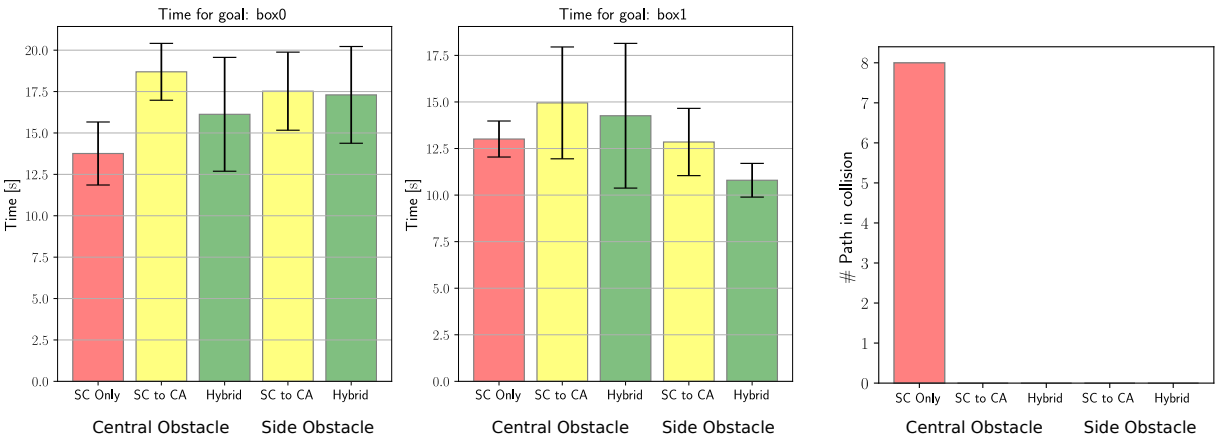


Figure 3.10: Average times for second simulation divided in two plot, one for each goal. The order of the bar is kept with Central Obstacle: *SC Only*, *SC to CA*, *Hybrid* then Side Obstacle: *SC to CA*, *Hybrid*. On the right side the number of collision for each of them.

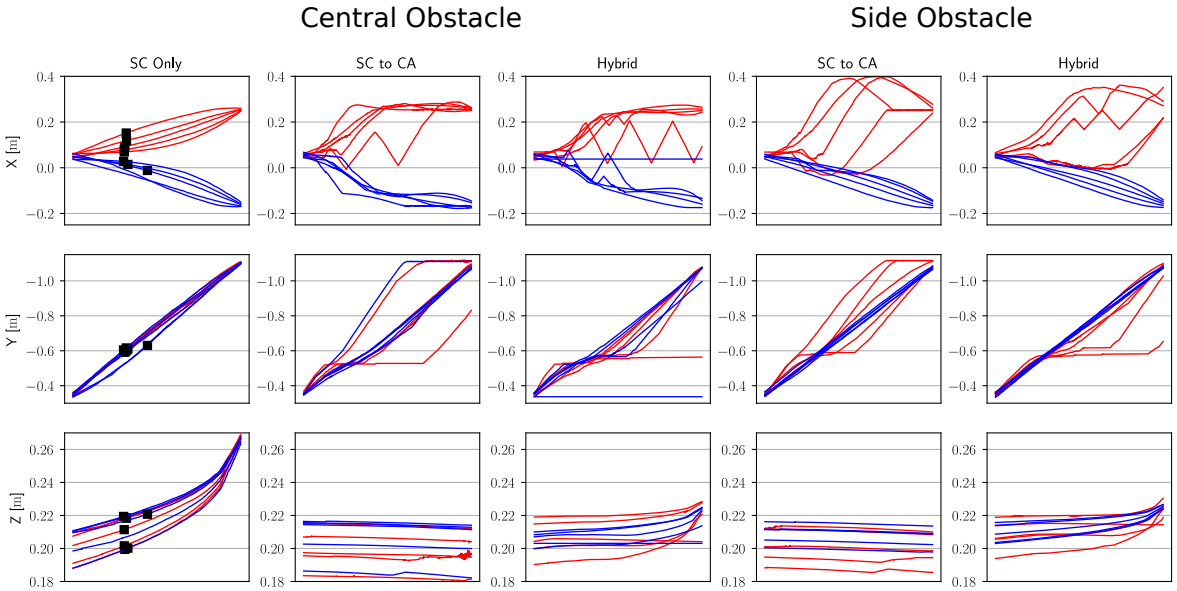


Figure 3.11: The position values of the gripper are shown for simulation 2. The collisions are marked with a black square. All sequences have their time length normalized.

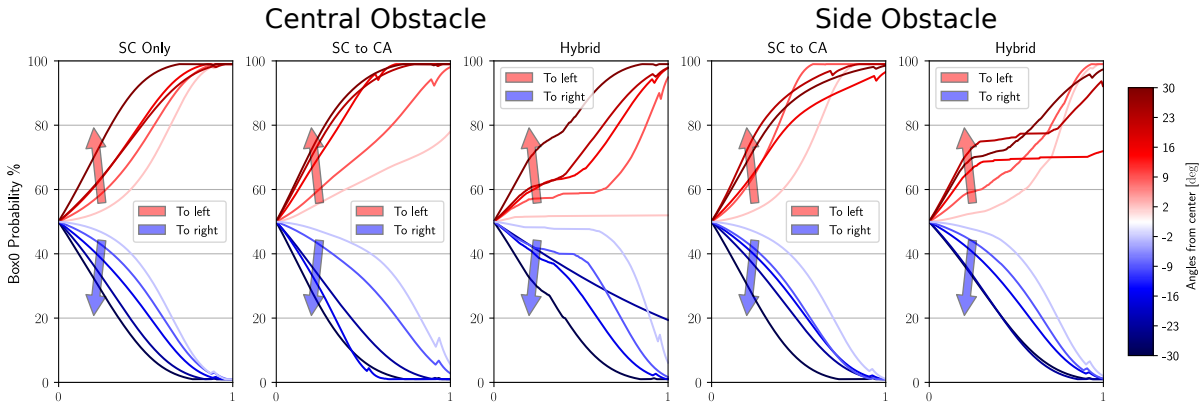


Figure 3.12: Probabilities for the second simulation of goal ”box0” over time. The times are normalized to have same length. The probability of ”box1” is not shown as it is complementary to ”box0”.

3.5 Complex Workspace

The previous simulations showed two different types of user in a simple scenario, but they have some limits. While an user with continuous control would change direction based on observation like in simulation2, he needs more time than the system to update his current command. To overcome this limitation the direction is no longer changed at each step, instead the update for the simulated input is done every two seconds. After that the command is maintained for the next two seconds till the new update of direction. With a total time of twenty seconds there are a total of nine different command.

The new input has been tested with two different environments, they have different placement of obstacles making the test more challenging. In the first workspace, called simulation 3, the only path for the gripper is a narrow passage in the center. The passage forces all trajectories to pass through the center, making goal probabilities difficult to estimate. Therefore the system has less time to select the correct goal and then assisting to reach the target. The other workspace, simulation 4, has two narrow passages one for each goal and allows the system to detect the correct target from the beginning. The two simulations share the same data for *SC Only* as the results are not affected by obstacles giving the same output.

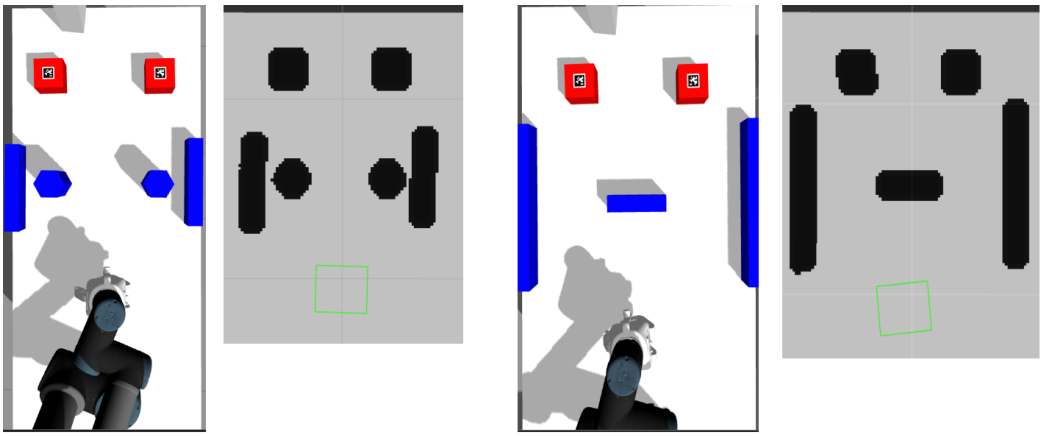


Figure 3.13: From the left: simulation3 in simulation, sim3 costmap, simulation4 in simulation, sim4 costmap. The passages are just big enough for the gripper to pass with the collision avoidance but they need a precise straight path through the obstacles.

The path of the new simulated input can be seen in Figure 3.14. The direct configuration avoided the obstacles but in most of the tasks failed to reach the goal in time. Nine out of ten task failed, two before surpassing the obstacles. In the other it failed as the avoidance required a lot of time, ending before the target was reached. In some cases, the tasks ended on top of "box1" and it may seems from this perspective that they were successful. With the extra information of the gripper position over z axis, it is evident that the gripper wasn't close enough to the target, and the system couldn't assist within the time frame resulting in failed task. In the *Hybrid* configuration, the obstacles are avoided and the goals are reached in time, and even though two paths of "box1" showed to move in the wrong direction for some time, they were recovered.

The trajectories of the gripper for sim4 are shown in Figure 3.15. The central obstacle forced the robot to move to the side, but in most of the test with direct configuration it couldn't reach the passage. While close to the obstacle the avoidance has most of the control but sometimes the robot reach one of those position with opposing command where the avoidance change direction to avoid the obstacle but on the other side. In each test, these positions have been encountered different times resulting in the robot being stuck in front of the obstacle. In the *Hybrid* configuration experiments some direction changes also happened but they were less influential, allowing the robot to avoid

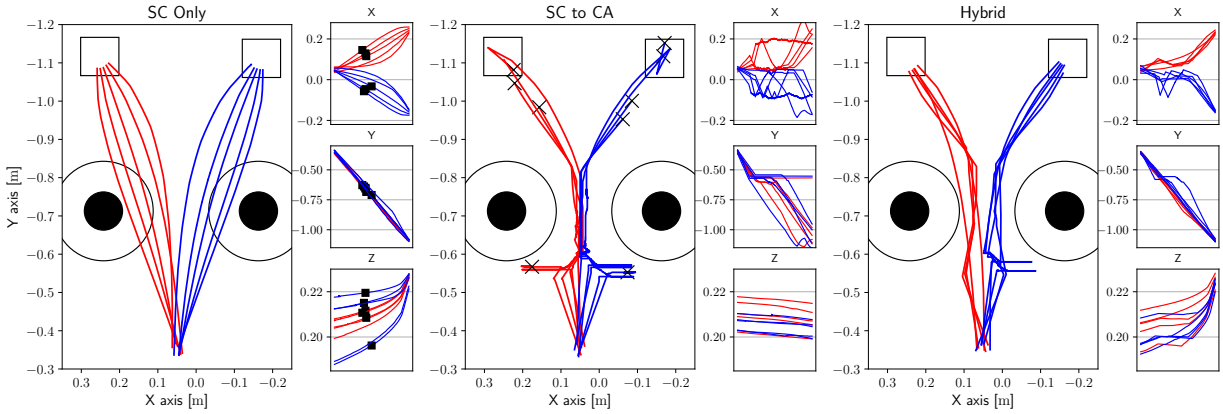


Figure 3.14: Trajectories of the gripper in sim3 with central passage. Each plot with path has its coordinate over time shown on the right side of it. The length are normalized for time axis.

the obstacle correctly and reaching in time the target. By counting the number of successful task, the best result is achieved by the *Hybrid* with only three failures, whereas the direct configuration has eight failures due to time limit.

Average times follow the same behaviour of the previous simulations as seen in Figure 3.16. The *SC Only* configuration has the best times, while the direct configuration is the slowest with an average close to the time limit. The *Hybrid* configuration maintained similar performance to *SC Only*.

The new type of simulated input updated over time is more stable as it always gives the correct path minus a small error, and even with different failed test the correct goal has been maintained over time as seen in Figure 3.17. One interesting aspect is found on sim4 with *Hybrid* configuration. There are three probabilities which remained stable for most of the time. They correspond to the three tests with time limit reached (Figure 3.15) and they selected correctly the goal but were unaffected by side movements due to collision avoidance.

In the figures where time is shown (Figure 3.10, Figure 3.6, Figure 3.16) the same pattern between configuration is visible, but for a better readability all the times have been normalized. This step was necessary as the position of the obstacle and its dimensions greatly affect the result of single tests. Therefore each sample of tests having the

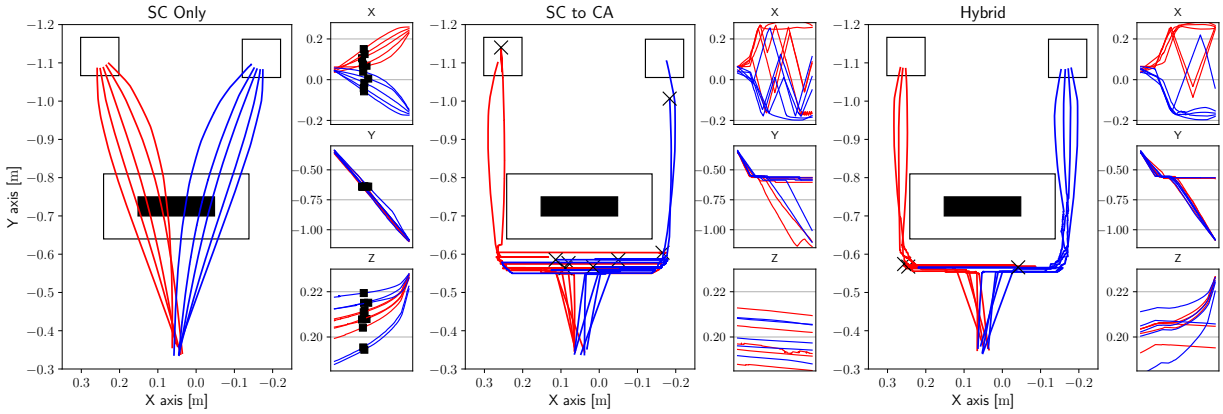


Figure 3.15: Trajectories of the gripper in sim4 with side passages. The obstacles on the sides are not visible due to being outside of the x range. Each plot with path has its coordinate over time shown on the right side of it. The length are normalized for time axis.

same simulation, configuration of nodes and goal have been normalized between its maximum and minimum value. The results are shown in Figure 3.18. The *SC Only* configuration and *Hybrid* have similar average value while the direct configuration is the slowest, confirming the observation of the previous time plot. Having the times normalized, allowed to test the statistical difference between the configuration and the result is shown in the same plot. Between *SC Only* and *SC to CA* the result is $p < 0.05$ showing that their main difference is only the average value. Between *SC to CA* and *Hybrid* the result is also $p < 0.05$, as they effectively behaved similarly but with the latter being better in many test and faster.

The results of the simulated input give some observations on the effectiveness of the architecture. The *SC Only* works correctly giving a lot of assistance, resulting in overall faster times and good precision for the target. The limit of this system is the lack of knowledge of the obstacles in the path resulting in many collisions. The simple approach of direct architecture, (*SC to CA*), is capable of avoiding the obstacle but in some condition it leads to conflict between the two controllers (assistance to target and collision avoidance) resulting in the robot having problem in difficult position like sim3 and sim4 where the distance between obstacles is narrow or the obstacle is placed between robot and goal. The *Hybrid* configuration retains the ability to avoid the obstacles,

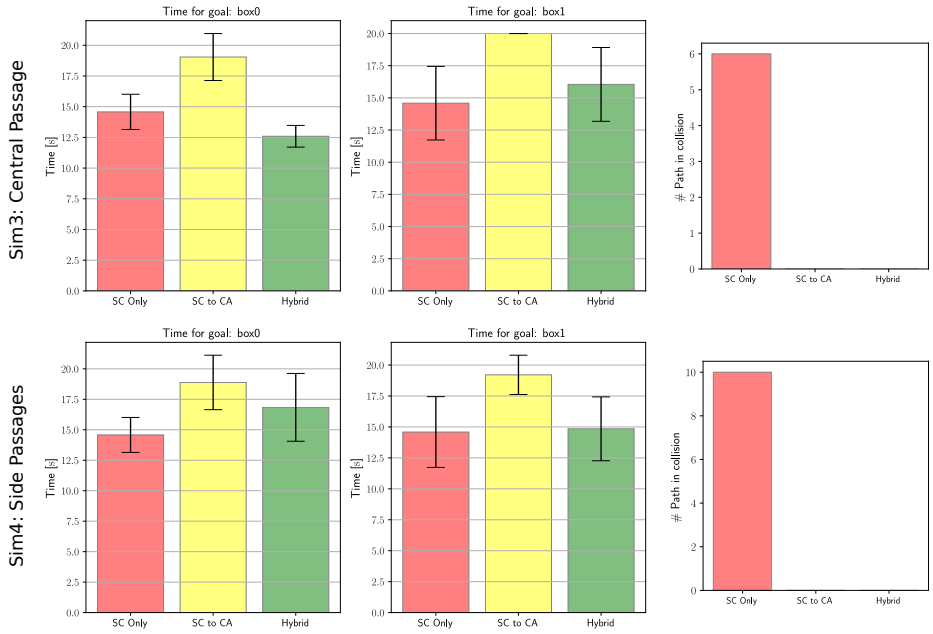


Figure 3.16: Average times for third and fourth simulations divided in two plots, one for each goal. On the right side the number of collisions for each of them. The first line is for simulation3 with central passage while the second line for simulation4 with the two side passages.

while also weighting the two systems. The result is the ability to give more importance to the critical part, avoidance of obstacle and assistance to target, based on the current situation.

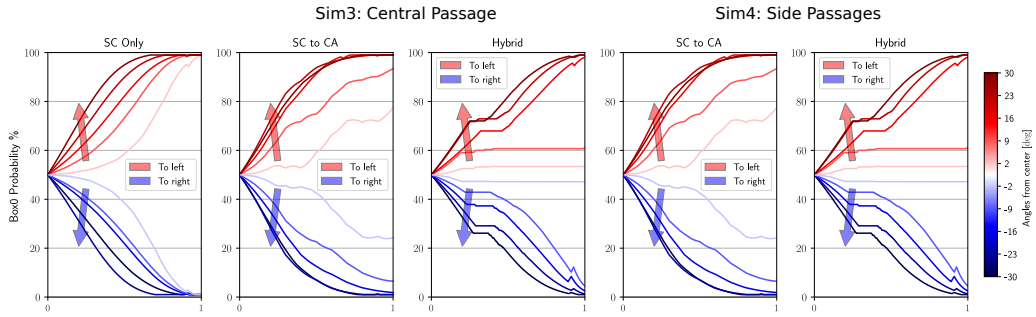


Figure 3.17: Probability of goal "box0" for simulation3 (central passage) on first row, simulation4 (side passages) on second row.

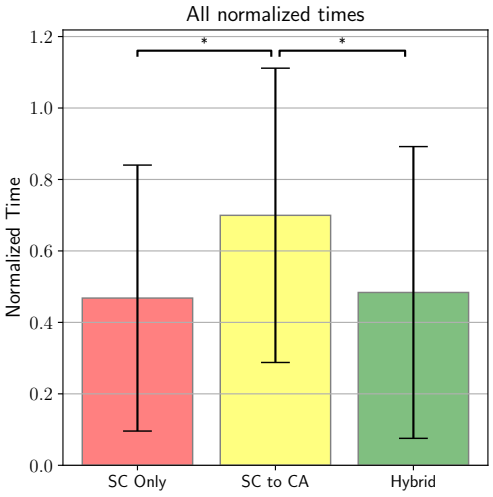


Figure 3.18: Average normalized times of simulated input. Normalized between maximum and minimum for their test. Statistical difference test result marked on bars.

3.6 Performance with User Control

The tests with simulated input have shown that the *Hybrid* configuration provides the collision avoidance with time performance similar to the *SC Only* configuration. With these information the *Hybrid* configuration is therefore better than the direct configuration. What may limit these tests is the input. While in simulation3 and simulation4, the simulated input has been modified to better adapt to human input, a real user might greatly differ. Therefore some tests with human users were made, where the input has been provided with the MYO described previously in section 3.3.

3.6.1 Testing the control

The first set of tests used a workspace similar to that of sim1 and sim2, but instead of using the cylinder as obstacle, it uses a parallelepiped placed in the center and slightly moved to the left side making the object more difficult to be reached. The user had control on movements over the plane (x,y) but no control over the elevation of the gripper (z axis). Some time has been given to the user to become familiar with the controller. The task was considered completed on user decision when he was satisfied with the position of the gripper respect to the goal. This type of control led to many tasks ending not exactly on top of the object as from the perspective of the simulation it is difficult for the user to understand correctly the relative position of the gripper to the goal.

In Figure 3.19 the trajectories of the gripper are presented and it is clear that the user ended most of his tasks before reaching the correct pose. The human user is capable of avoiding the obstacle also in *No Assistance* case, even though it required a lot of direction changes and corrections to reach the target. In the test with only assistance to target, the user collided with the obstacle for the first two test, when he was heading towards goal "box0". This is the result of the assistance giving more speed to reach the two goal but ignoring the obstacle. In the other tests of the same configuration, no collisions occurred as the user adapted his own behaviour pushing the gripper far from the obstacle before reaching the collision. Therefore the task was completed but the user

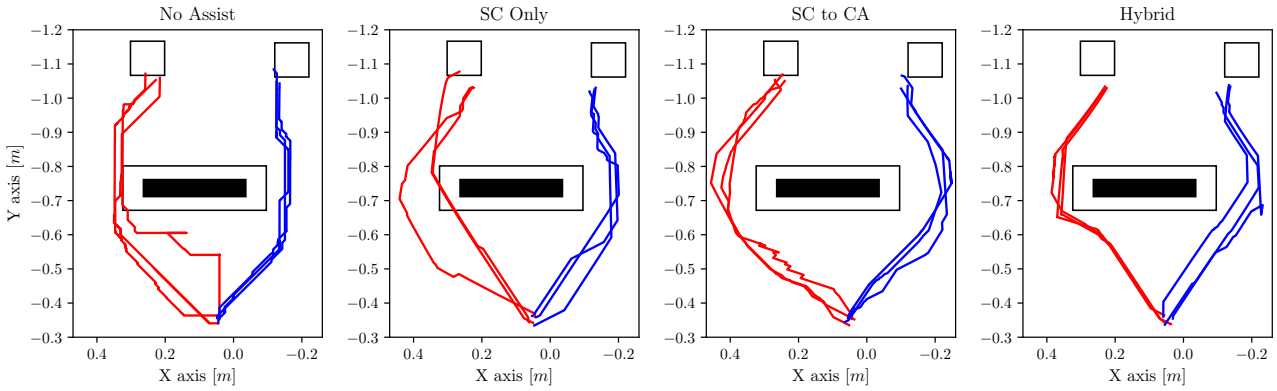


Figure 3.19: Trajectories of the gripper in the first MYO experiment. The obstacle is placed in the center forcing the user on the sides. Differently from sim4 the user can go how much he desires to the side. The trajectories end when the user decided he was satisfied with the position.

had to fight against the assistance.

In the direct architecture, the obstacle is correctly avoided but it can be seen that the path is jumping between different directions. The inclusion of the avoidance even with the simple direct configuration provided smoother trajectories. With the *Hybrid* architecture the trajectories are smoother and the collision is avoided with no jumping direction.

Observing the times in Figure 3.20, the non assisted case proved to be the slowest while the *SC Only* configuration was the fastest. The *SC Only* configuration has been faster because the assistance always pushed the gripper towards the goal, but it is also the only configuration which caused collisions as in the simulated input scenario. The *SC to CA* and *Hybrid* achieved similar timings, with slightly better performance for the direct configuration.

At this point, the direct configuration would seem to be better, but observing the number of direction changes required to the user to reach the goal, shown in Figure 3.21, the *Hybrid* configuration had the smallest number of direction changes. Therefore this configuration maintained time performance close to the case of non assistance,

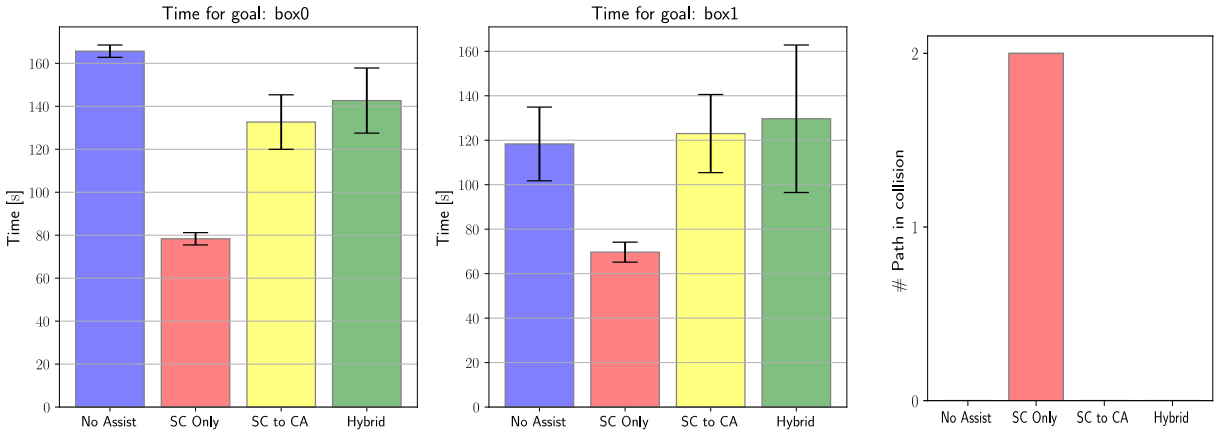


Figure 3.20: Average times for MYO control divided in two plot, one for each goal. On the right side the number of collision for each of them.

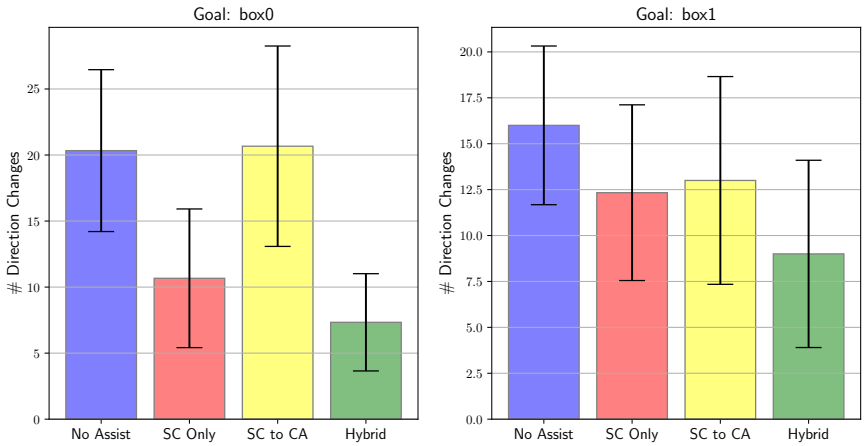


Figure 3.21: The average numbers of direction changes divided per goal in two plots.

but requiring less intervention from the user and correctly avoiding the obstacle. The assistance to target is the main objective of the shared autonomy system and the *Hybrid* configuration allowed the user with a smaller number of commands to reach the goal. The limit of these tests is the possibility to stop it on user decision which makes difficult to verify the performance in assisting precise movements.

3.6.2 Second part of user test

In the previous set of tests, the user ended too early not reaching the target and therefore the performance in assisting precise movements was not clear. For this reason, the user goal for the next experiments is the target position with a small threshold set to 3 cm of distance. As the target is placed on top of the object, the user control has been updated with the vertical axis giving full control on the 3D space. The control of an additional degree of freedom (i.e., gripper elevation) makes the task more complex as it is difficult to have a correct estimation of the 3D gripper position from the user's perspective. The configuration with only the assistance (*SC Only*) has been discarded as it has been shown that it cannot avoid obstacles without the need of strong intervention from the user.

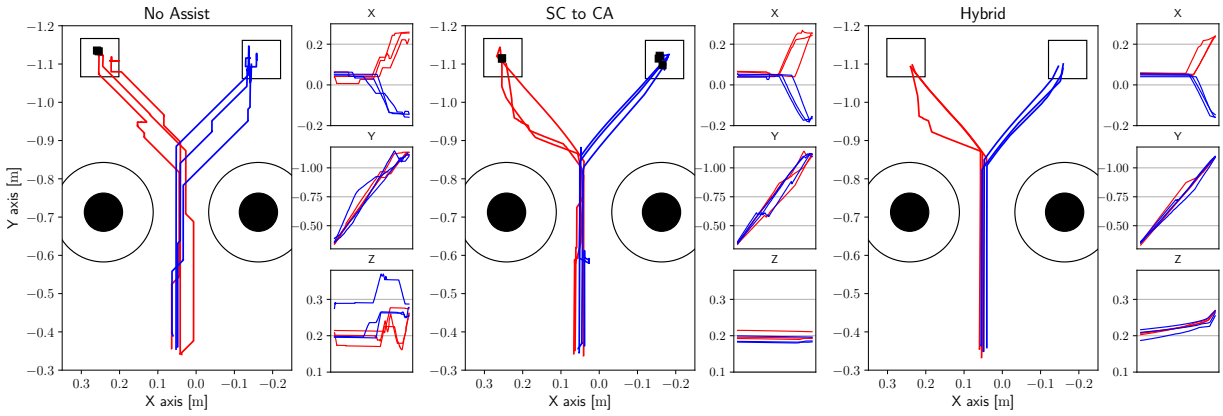


Figure 3.22: Trajectories of the gripper in the second MYO experiment. The trajectories which are marked with a black square collided with the object.

In Figure 3.22, the trajectories of the gripper are shown, and on the side the posi-

tion over time for each configuration. The assistance to target is clearly visible as the paths in *No Assist* have many direction changes to reach the goal while the other two have their paths smoother. It is also visible the challenge of controlling the elevation of the gripper as it required many changes to reach the correct target and sometimes leading to collision with the object. Proceeding with the direct architecture it is visible how the assistance and the collision avoidance gave the user better control, requiring far less changes of direction and at the same time avoiding the obstacle. The limit of this architecture was also visible in the test with simulated input, the elevation is not considered as the avoidance works on the horizontal plane, thus resulting in many collisions with the goal. The *Hybrid* architecture instead reaches the correct target maintaining smooth trajectories.

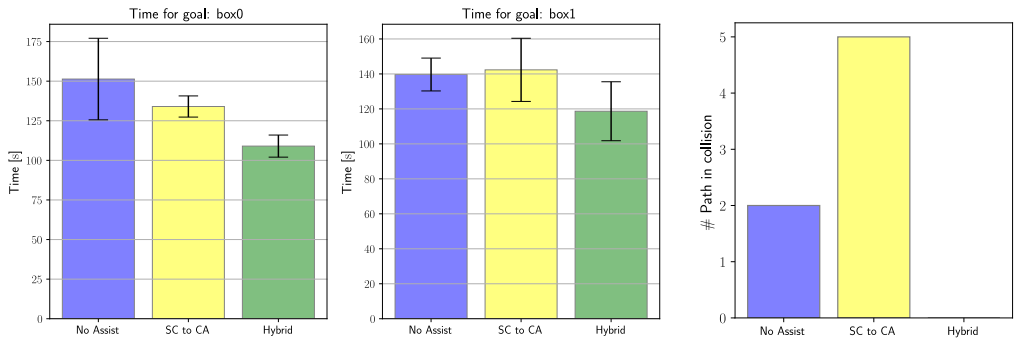


Figure 3.23: Average times for the second simulation with MYO control divided in two plot, one for each goal. The order of the bar is with: *No Assist* for direct control, *SC to CA* direct configuration, *Hybrid* architecture. On the right side the number of collision for each of them.

The time for test with *Hybrid* configuration showed to be faster than the other two, as seen in Figure 3.23. The collision with the object are counted and visible in the same figure showing the limitation of the collision avoidance.

The smoothness of the trajectory can be measured as number of different commands the user needs to send in order to correct the robot behavior. In Figure 3.24 the results for the second test with the MYO are shown. The conclusion taken previously from the paths are confirmed, the *Hybrid* configuration required less commands from the user in respect to the other two configurations.

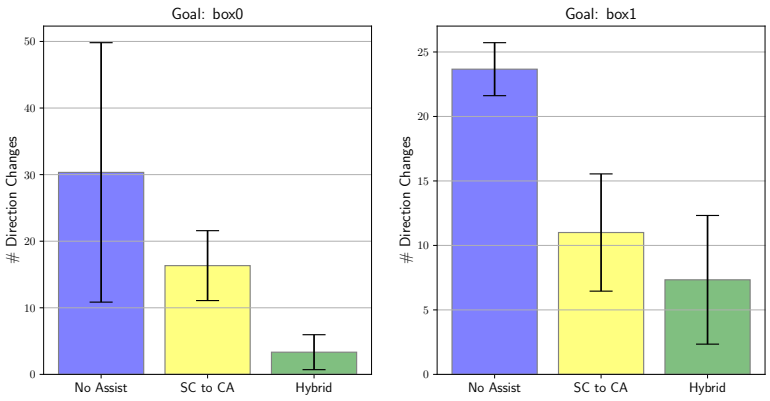


Figure 3.24: The average numbers of direction changes for the second simulation with MYO control, divided per goal in two plots.

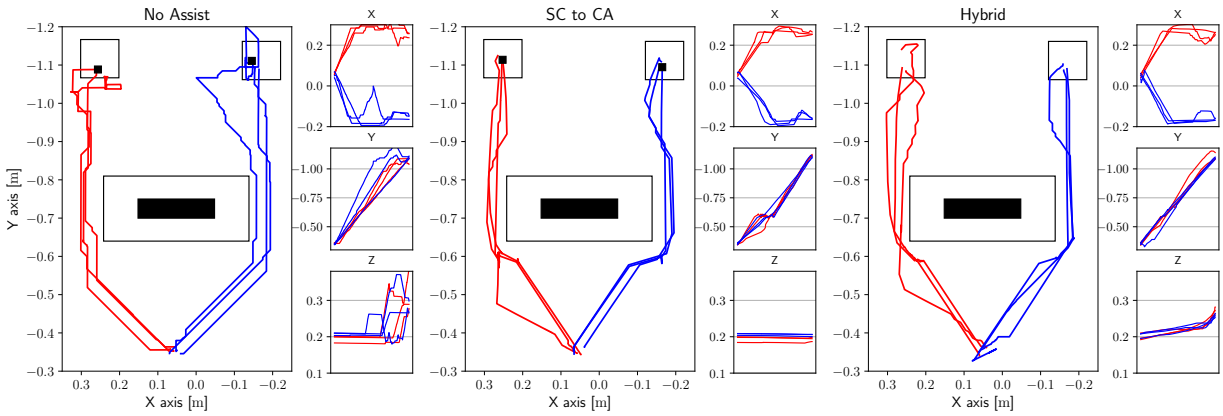


Figure 3.25: Trajectories of the gripper in the third MYO experiment. The trajectories which are marked with a black square collided with the object.

The third set of simulations with user control, shares the workspace of sim4 but it has been tested with a different user from the previous test. Again the control in elevation has been difficult and in the *No Assist* case required many corrections to reach the correct target as seen in Figure 3.25. With too much elevation the user also made wrong estimation of gripper position resulting in wrong movements and numerous corrections. The *Hybrid* architecture which also enabled movement on the vertical axis showed this behaviour but it managed to reach the target without collisions with the object, differently from the direct configuration which collided two times.

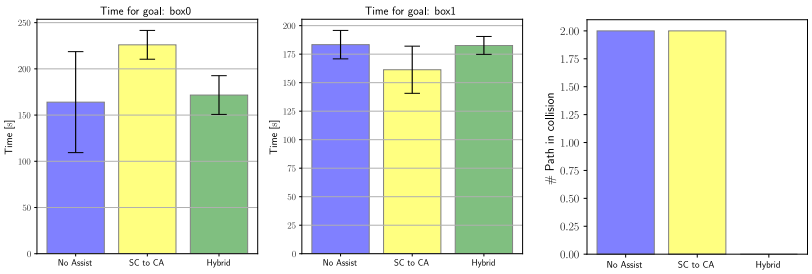


Figure 3.26: Average times for the third simulation with MYO control divided in two plot, one for each goal. The order of the bar is with: *No Assist* for direct control, *SC to CA* direct configuration, *Hybrid* architecture. On the right side the number of collision for each of them.

The times of *No Assist* and *Hybrid* configurations are similar, as shown in Figure 3.26. In this test the direct architecture shows to be the fastest for "box1" and the slowest for "boxo". This behaviour depends on the obstacle which is placed slightly on the left in front of "boxo" and therefore it is more dependant on the avoidance. The *No Assist* and *Hybrid* are less influenced by the obstacles having similar timings between different goals.

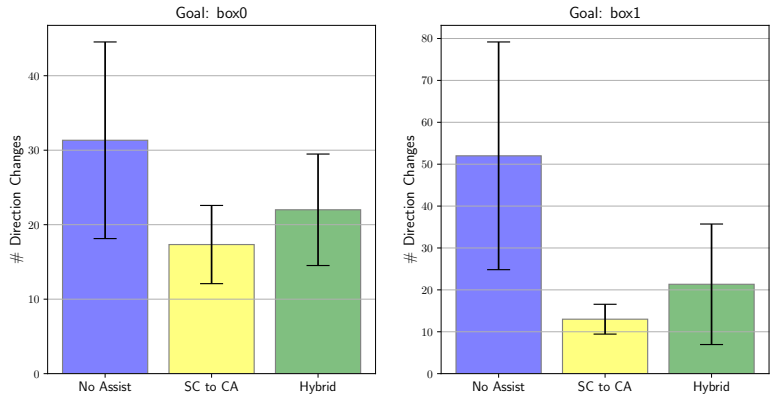


Figure 3.27: The average numbers of direction changes for the second simulation with MYO control, divided per goal in two plots.

The last plot for this test is the number of direction changes showed in Figure 3.27. This user required many commands to reach the goal as he showed to have incorrect estimate of elevation, seen in Figure 3.25. In this test the direct configuration might be better, but it had two collisions with the obstacle.

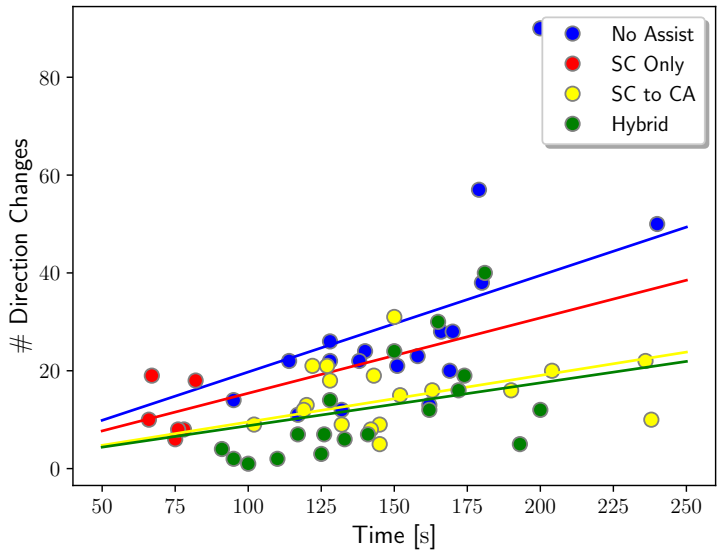


Figure 3.28: All of the test with user control plotted together. Each point is a single experiment with color based on its configuration. For each configuration the regression line is plotted.

With all the data from user control tests, a single plot which places times on the horizontal axis and the number of direction changes on the vertical axis is shown in Figure 3.28. The regression line is showed for each of the control configuration. This result confirms the observations made before. The *No Assist* is always the worst scenario, requiring more commands as well as more time respect to the others. The two configurations with the collision avoidance system (direct and hybrid) have shown to be better than assistance only with the same timings, confirming that integrating the avoidance to the system provides a better shared control. Between direct and hybrid it is possible to observe that the difference between the intercept of direct and hybrid is small, with the latter configuration being slightly better. The most notable difference is that *Hybrid* architecture have most of the test with smaller timings and number of commands, therefore they have similar ratio between time and number of commands, but the hybrid shows generally to be faster and to use a smaller number of commands.

4 Conclusions

Several experiments have been made to cover different possible situation and their results are showed in the previous chapter. The tests used user and simulated input, and the latter also covered different type of user (always in control like the user input and only one command at start). The experiments also used different configuration and from the observations of the figures shown previously, some conclusion are made.
The simple configuration with assistance only, *SC Only*, provides maximum performance in reaching the target whether the input is simulated or not. The performance comes at the cost of higher risk of collision, as the user has to overcome the assistance to avoid the obstacle and in simulated input collisions are not avoided. Therefore the integration of the collision avoidance is required in order to achieve a better system. The Direct Architecture, *SC to CA*, where the avoidance uses the output of the assistance is the first step but it shows some limitation. If the robot find itself close to the target, the direct configuration may result in opposing behaviors from the two systems locking the robot in the current position. Another limitation is found after the obstacle has been avoided. In this situation the probabilities associated with the goals are changed based on the position, which in turn is affected by the obstacle avoidance resulting in the system assisting wrongly the user. While these two problems might be addressed by a user with good control capabilities, they require more command and usually even more time than the non assisted control.

The *Hybrid Architecture* has proven to overcome the limitation of the other architectures. It provides an additional policy to coordinate movements towards target

and away from obstacles as required by the current state of the system. It also provides collision avoidance without affecting the performance of assistance target and the user has to provide less command making the control easier. However it has some limitation which are related to the collision avoidance. The avoidance system is limited to bi-dimensional control and it shows varying behaviour based on how it is approaching the obstacle leading to different result even with the same condition, therefore an alternative avoidance system might be considered. The new avoidance could also cover movement in the three-dimensional environment and consider all the robot collision instead of only the end-effector. Another limitation is the input which is limited to speed control, which might be extended to position for example the position of the user hand.

Bibliography

[1] M. Desai and H. A. Yanco, "Blending human and robot inputs for sliding scale autonomy," in *ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005.*, pp. 537–542, Aug 2005.

[2] J. W. Crandall and M. A. Goodrich, "Characterizing efficiency of human robot interaction: A case study of shared-control teleoperation," in *IEEE/RSJ international conference on intelligent robots and systems*, vol. 2, pp. 1290–1295, IEEE, 2002.

[3] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & information systems engineering*, vol. 6, no. 4, pp. 239–242, 2014.

[4] D. Feil-Seifer and M. J. Mataric, "Defining socially assistive robotics," in *9th International Conference on Rehabilitation Robotics, 2005. ICORR 2005.*, pp. 465–468, IEEE, 2005.

[5] M. Attaran *et al.*, "The automated factory: justification and implementation," *Business Horizons*, vol. 32, no. 3, pp. 80–5, 1989.

[6] D. Feil-Seifer and M. J. Matarić, "Socially assistive robotics," *IEEE Robotics & Automation Magazine*, vol. 18, no. 1, pp. 24–31, 2011.

[7] R. C. Goertz, "Manipulators used for handling radioactive materials," *Human factors in technology*, pp. 425–443, 1963.

[8] T. Fong and C. Thorpe, "Vehicle teleoperation interfaces," *Autonomous robots*, vol. 11, no. 1, pp. 9–18, 2001.

[9] T. Fong, C. Thorpe, and C. Baur, "A safeguarded teleoperation controller," in *IEEE International Conference on Advanced Robotics (ICAR)*, no. CONF, 2001.

[10] S. J. Anderson, S. C. Peters, T. E. Pilutti, and K. Iagnemma, "An optimal-control-based framework for trajectory planning, threat assessment, and semi-autonomous control of passenger vehicles in hazard avoidance scenarios," *International Journal of Vehicle Autonomous Systems*, vol. 8, no. 2-4, pp. 190–216, 2010.

[11] S. A. Green, M. Billinghurst, X. Chen, and J. G. Chase, "Human-robot collaboration: A literature review and augmented reality approach in design," *International journal of advanced robotic systems*, vol. 5, no. 1, p. 1, 2008.

[12] J. DeGol, A. Akhtar, B. Manja, and T. Bretl, "Automatic grasp selection using a camera in a hand prosthesis," in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 431–434, IEEE, 2016.

[13] L. V. Herlant, R. M. Holladay, and S. S. Srinivasa, "Assistive teleoperation of robot arms via automatic time-optimal mode switching," in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 35–42, March 2016.

[14] D. P. McMullen, G. Hotson, K. D. Katyal, B. A. Wester, M. S. Fifer, T. G. McGee, A. Harris, M. S. Johannes, R. J. Vogelstein, A. D. Ravitz, *et al.*, "Demonstration of a semi-autonomous hybrid brain–machine interface using human intracranial eeg, eye tracking, and computer vision to control a robotic upper limb prosthetic," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 22, no. 4, pp. 784–796, 2014.

[15] D. J. Kim, R. Hazlett Knudsen, H. Culver Godfrey, G. Rucks, T. Cunningham, D. Portee, J. Bricout, Z. Wang, and A. Behal, "How autonomy impacts performance and satisfaction: Results from a study with spinal cord injured subjects using an assistive robot," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 42, no. 1, pp. 2–14, 2012.

[16] A. Leeper, K. Hsiao, M. Ciocarlie, L. Takayama, and D. Gossow, "Strategies for human-in-the-loop robotic grasping," in *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 1–8, IEEE, 2012.

[17] A. D. Dragan and S. S. Srinivasa, "A policy-blending formalism for shared control," *The International Journal of Robotics Research*, vol. 32, no. 7, pp. 790–805, 2013.

[18] S. Javdani, H. Admoni, S. Pellegrinelli, S. S. Srinivasa, and J. A. Bagnell, "Shared autonomy via hindsight optimization for teleoperation and teaming," *The International Journal of Robotics Research*, vol. 37, no. 7, pp. 717–742, 2018.

[19] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[20] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning.," in *Aaai*, vol. 8, pp. 1433–1438, Chicago, IL, USA, 2008.

[21] A. Fern and P. Tadepalli, "A computational decision theory for interactive assistants," in *Advances in Neural Information Processing Systems*, pp. 577–585, 2010.

[22] M. Chen, E. Frazzoli, D. Hsu, and W. S. Lee, "Pomdp-lite for robust robot planning under uncertainty," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5427–5433, IEEE, 2016.

[23] B. D. Ziebart, "Modeling purposeful adaptive behavior with the principle of maximum causal entropy," 2010.

[24] B. D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa, "Planning-based prediction for pedestrians," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3931–3936, IEEE, 2009.

[25] B. Ziebart, A. Dey, and J. A. Bagnell, "Probabilistic pointing target prediction via inverse optimal control," in *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, pp. 1–10, ACM, 2012.

[26] A. Campeau-Lecours, H. Lamontagne, S. Latour, P. Fauteux, V. Maheu, F. Boucher, C. Deguire, and L.-J. Caron L'Ecuyer, "Kinova modular robot arms for service robotics applications," *International Journal of Robotics Applications and Technologies*, vol. 5, pp. 49–71, 07 2017.

[27] C. Borst, M. Fischer, and G. Hirzinger, "A fast and robust grasp planner for arbitrary 3d objects," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, vol. 3, pp. 1890–1896, IEEE, 1999.

[28] T. Lozano-Perez, "Automatic planning of manipulator transfer movements," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, pp. 681–698, Oct 1981.

[29] J. Crowley, "Navigation for an intelligent mobile robot," *IEEE Journal on Robotics and Automation*, vol. 1, no. 1, pp. 31–41, 1985.

[30] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.

[31] G. Beraldo, N. Castaman, R. Bortoletto, E. Pagello, J. d. R. Millán, L. Tonin, and E. Menegatti, "Ros-health: An open-source framework for neurorobotics,"

in *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pp. 174–179, IEEE, 2018.

[32] S. Tortora, M. Moro, and E. Menegatti, "Dual-myo real-time control of a humanoid arm for teleoperation," in *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 624–625, IEEE, 2019.