A Reactive Trajectory Controller for Object Manipulation in Human Robot Interaction

Wuwei He[1,2], Daniel Sidobre[2,3] and Ran Zhao[1,2]
[1]*CNRS, LAAS, 7 Avenue du Colonel Roche, F-31400 Toulouse, France*
[2]*Univ. de Toulouse, LAAS, F-31400 Toulouse, France*
[3]*Univ. de Toulouse, UPS, LAAS, F-31400 Toulouse, France*

Keywords: Robotic, Manipulation, Trajectory Generation, Human-robot Interaction, Control.

Abstract: This paper presents a reactive trajectory controller for manipulating objects in Human Robot Interaction (HRI) context. The trajectories to be followed by the robot are provided by a human aware motion planner. The controller is based on an online trajectory generator, which is capable of calculating a trajectory from an arbitrary initial condition to a target within one control cycle. The controller is capable of switching to a new trajectory each time the motion planner provides a new trajectory, changing the frame in which the input trajectory is controlled and tracking a target or a trajectory in a frame, which moves with respect to the robot frame. The controller chooses different control modes for different situations. Visual servoing by trajectory generation is considered as one case of the control situations. Some results obtained with this controller are presented to illustrate the potential of the approach.

1 INTRODUCTION

In the context of Human Robot Interaction (HRI), intuitive and natural object exchange between human and robot is one of the basic necessary tasks for object manipulation. This paper presents a controller which enables the robot to realize a complete task of object exchange while respecting human's safety and other HRI specifications, such as monitoring the accessibility of human. This elementary manipulation task demands integration of different elements like geometrical and human-aware reasoning, position and external force monitoring, 3D vision and human perception information. The control system presented in this paper proposes to define a plan as a series of control primitives, each associated with a trajectory segment and a control mode.

Structure of the Paper: We introduce firstly the architecture of the system and present the related work. In section 2 we present briefly the trajectory generator and how cost values are associated to the trajectory based on cost maps. In section 3, we discuss the controller, which is based on the online trajectory generation. Some results and comparison could be found in section 4, followed by the conclusion.

1.1 Software Architecture for Human Robot Interaction

The robots capable of doing HRI must realize several tasks in parallel to manage various information sources and complete tasks of different levels. Figure 1 shows the proposed architecture where each component is implemented as a GENOM module. GENOM (Fleury et al., 1997) is a development environment for complex real time embedded software.

At the top level, a task planner and supervisor plans tasks and then supervises the execution. The module SPARK (**Spa**tial **R**easoning and **K**nowledge) maintains a 3D model of the whole environment, including objects, robots, posture and position of humans (Sisbot et al., 2007b). It manages also the related geometrical reasoning of the 3D models, such as collision risk between the robot parts and between robot and environment. An important element is that SPARK produces cost maps, which discribe a space distribution relatively to geometrical properties like human accesibility. The softwares for perception, from which SPARK updates the 3D model of the environment, are omitted here for simplicity. The module runs at a frenquency of 20Hz, limited mainly by the complexity of the 3D vision and of the perception of

He W., Sidobre D. and Zhao R..
A Reactive Trajectory Controller for Object Manipulation in Human Robot Interaction.
DOI: 10.5220/0004427000190028
In *Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2013), pages 19-28
ISBN: 978-989-8565-71-6

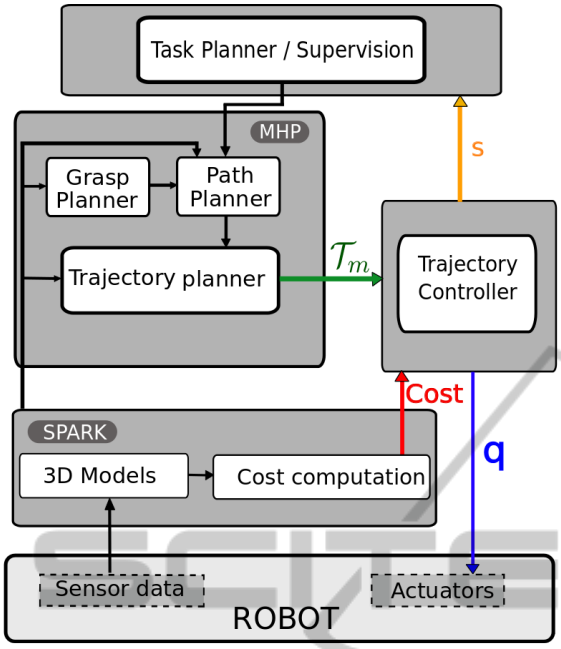

Figure 1: Software Architecture of the robot for HRI manipulation. $\mathcal{T}_m$ is the main trajectory calculated initially by MHP(**M**otion in **H**uman **P**resence). The controller takes also cost maps from SPARK. The controller sends control signals in joint(q in the figure) to the servo system, and during the execution, the controller sends the state of the controller (s) to the supervisor

human.

Another important module, MHP (**M**otion in **H**uman **P**resence), integrates path and grasp planner. RRT (Rapidly exploring Random Tree) and its variants (Mainprice et al.,) are used by the path planner. The paths could be in Cartesian or joint spaces depending on the task type. From the path, an output trajectory is computed to take the time into account. MHP calculates a new trajectory each time the task planner defines a new task or when the supervisor decides that a new trajectory is needed for the changes of the environment.

The system should adapt its behavior to the dynamic environment, mainly the human activities. But as the planning algorithms are time consuming, we introduce a trajectory controller that runs at 100 Hz, an intermediate frequency between the one of the MHP planner and the one of the fast robot servo system. This trajectory controller allows the system to adapt faster the trajectory to the environment changes.

The main functionalities of the whole controller are:

- A decision process capable of integrating information form high-level software and building control primitives by segmenting the trajectory based on the HRI specifications.
- Algorithms for target or trajectory tracking, trajectory switching, and coordinates transformation.
- A low-level collision checker and a monitor of the external torques for the safety issues.

1.2 Related Works

Reactive controller for object manipulation is a research topic that is part of the fundamentals of robotic manipulation. Firstly, trajectory generation based approaches have been developed. In (Buttazzo et al., 1994), results from visual system pass firstly through a low-pass filter. The object movement is modeled as a trajectory with constant acceleration, based on which, catching position and time is estimated. Then a quintic trajectory is calculated to catch the object, before being sent to a PID controller. The maximum values of acceleration and velocity are not checked when the trajectory is planned, so the robot gives up when the object moves too fast and the maximum velocity or acceleration exceeds the capacity of the servo controller. In (Gosselin et al., 1993), inverse kinematic functions are studied, catching a moving object is implemented as one application, a quintic trajectory is used for the robot manipulator to joint the closest point on the predicted object movement trajectory. The systems used in those works are all quite simple and no human is present in the workspace. A more recent work can be found in (Kröger and Padial, 2012), in which a controller for visual servoing based on Online Trajectory Generation (OTG) is presented, and the results are promising.

Secondly, the research area of visual servoing provides also numerous results, a survey of which were presented by Chaumette and Hutchinson (Chaumette and Hutchinson, 2006), (Chaumette and Hutchinson, 2007) and a comparison of different methods can be found in (Farrokh et al., 2011). Classical visual servoing methods produce rather robust results and stability and robustness can be studied rigorously, but they are difficult to integrate with a path planner, and could have difficulties when the initial and final positions are distant.

Another approach to achieve reactive movements is through Learning from Demonstration(LfD). In (Calinon and Billard, 2004) and (Vakanski et al., 2012), points in the demonstrated trajectory are clustered, then a Hidden Markov Model(HMM) is built. Classification and reproduction of the trajectories are then based on the HMM. A survey for this approach is proposed in (Argall et al., 2009). Although LfD can produce the whole movement for objects manipulation, many problems may arise in a HRI context

as LfD demands large set of data to learn, and the learned control policies may have problem to cope with a dynamic and unpredictable environment where a service robot works.

Our approach to build the controller capable of controlling a complete manipulation tasks is based on Online Trajectory Generation. More results on trajectory generation for robot control can be found in (Liu, 2002), (Haschke et al., 2008), and (Kröger et al., 2006). The controller is capable of dealing with various data in HRI context. Compared to methods mentioned above, approaches based on OTG have the following advantages:

- The integration with a path planner is easy and allows to comply with kinematic limits like the one given by human safety and comfort.
- The path to grasp a complex moving object is defined in the object frame, making sure that the grasping movement is collision free.
- The trajectory based method allows to create a simple standard interface for different visual and servo systems, easy plug-in modules can be created.

The controller integrates various information from high-level software. More information about human-aware motion planning in the system can be found in (Mainprice et al., 2011). More about geometrical reasoning can be found in (Sisbot et al., 2011), (Mainprice et al.,), and (Sisbot et al., 2007a). Physical Human Robot Interaction is a dynamic research area including various aspects, such as safety, control architecture, planning, human intention recognition, and more. Readers may refer to (De Santis et al., 2008), (Sidobre et al., 2012), and (Strabala et al., 2013), among many others, for more information of this research field.

2 TRAJECTORY AND CONTROL PRIMITIVES

2.1 Online Trajectory Generation

Trajectories are time functions defined in geometrical spaces, mainly Cartesian space and joint space for robots. The books from Biagiotti (Biagiotti and Melchiorri, 2008) and the one from Kroger (Kröger, 2010) summarize background materials. For detailed discussion about the trajectory generator used in the system, the reader can refer to (Broquère et al., 2008) and (Broquère and Sidobre, 2010), here we describe some results without further discussion.

Given a system in which position is defined by a set of coordinate X, a trajectory $\mathcal{T}$ is a function of time defined as:

$$\begin{aligned} \mathcal{T} : [t_I, t_F] &\longrightarrow \mathbb{R}^N \\ t &\longmapsto \mathcal{T}(t) = X(t) \end{aligned} \tag{1}$$

The trajectory is defined from the time interval $[t_I, t_F]$ to $\mathbb{R}^N$ where N is the dimension of the motion space. The trajectory $\mathcal{T}(t)$ can be a direct function of time or the composition $\mathcal{C}(s(t))$ of a path $\mathcal{C}(s)$ and a function $s(t)$ describing the time evolution along this path. The time evolution could be used in the controller to slow down or to accelerate when necessary (Broquère, 2011).

Our trajectory generator is capable of generating type V trajectories defined by Kroger (Kröger, 2010) as satisfying:

$$\begin{array}{lll} X(t_I) \in \mathbb{R} & X(t_F) \in \mathbb{R} & |V(t)| \leqslant V_{max} \\ V(t_I) \in \mathbb{R} & V(t_F) \in \mathbb{R} & |A(t)| \leqslant A_{max} \\ A(t_I) \in \mathbb{R} & A(t_F) \in \mathbb{R} & |J(t)| \leqslant J_{max} \end{array} \tag{2}$$

Where X, V, A and J are respectively the position, the velocity, the acceleration and the jerk.

For a motion of dimension N, the algorithms find a trajectory $X(t)$, which satisfies:

1. The initial conditions (IC): $X(t_I) = X_I$, $V(t_I) = V_I$ and $A(t_I) = A_I$;
2. The final conditions (FC): $X(t_f) = X_F$, $V(t_f) = V_F$ and $A(t_f) = A_F$;
3. The continuity class of the trajectory is $\mathcal{C}^2$.
4. The kinematics limits V_{max}, A_{max} and J_{max}.

The problem is solved by a series of 3^{rd} degree polynomial trajectories. Such a trajectory is composed of a vector of one-dimensional trajectories: $\mathcal{T}(t) = ({}_1Q(t), {}_2Q(t), \ldots, {}_NQ(t))^T$ for joint motions or $\mathcal{T}(t) = ({}_1X(t), {}_2X(t), \ldots, {}_NX(t))^T$ in Cartesian space.

For the discussion of the next sections, we define Motion Condition as the position, velocity and acceleration at time t of the trajectory: $M(t) = (X(t), V(t), A(t))$. Once the trajectory is calculated, the function $M(t) = getMotion(t, \mathcal{T})$ returns the Motion Condition on trajectory $\mathcal{T}$ at time t.

2.2 Control Primitives

In HRI, the robot does various tasks like picking up an object, giving an object to human, taking an object from the human. For each of the task, a path is planned to realize it, and then the path is transformed into a trajectory. The controller designed here takes directly the trajectory as input and segments it based on the cost maps.

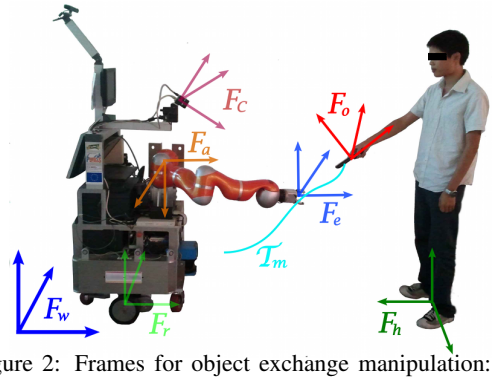

Figure 2: Frames for object exchange manipulation: F_w: world frame; F_r: robot frame; F_c: camera frame; F_e: end effector frame; F_o: object frame; F_h: human frame. The trajectory realizing a manipulation should be controlled in different task frames.

Figure 2 shows the basic frames needed to define a task. The trajectory $\mathcal{T}_m$ defines the move that allows the robot to do the task of grasping an object handed by the human.

Based on the cost values associated to each point of the trajectory, the trajectory is divided into segments associated to a control strategy. The 3D cost maps used are of different types: collision risk map calculated based on the minimum distance between trajectory and the obstacles; visibility and reachability map of a human (Sisbot et al., 2011) and safety and comfort 3D map of a human, Figure 3 shows two examples of cost maps. For example, when the risk of collision with the robot base is high, the trajectory can be controlled in robot the frame. Similarly, in the case where the human is handing an object to the robot, the grasping must be controlled in the object frame. (Sidobre et al., 2012) details other aspects of the use of cost maps to plan manipulation tasks.

To simplify the presentation, in the reminder of the paper we focus on the manipulation tasks where

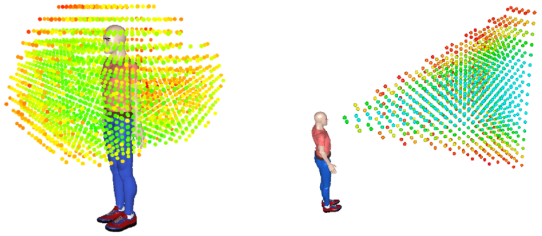

Figure 3: Left: 3D reachability map for a human. Green points have low cost, meaning that it is easier for the human to reach, while the red ones, having high cost, are difficult to reach. One application is that when the robot plans to give an object to a human, an exchange point must be planned in the green zone. Right: 3D visibility map. Based on visibility cost, the controller can suspend the execution if the human is not looking at the robot.

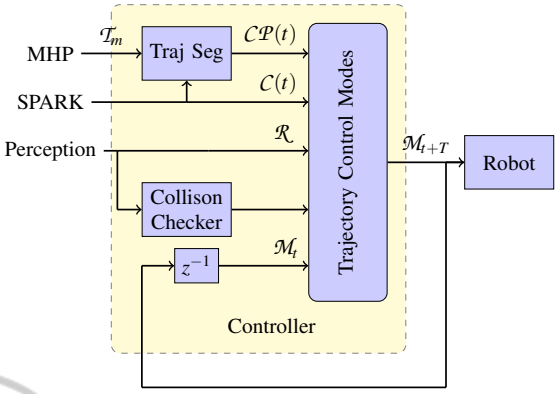

Figure 4: Input and output of the controller. $\mathcal{T}_m$ is the trajectory computed by MHP, it is then segmented into control primitives ($\mathcal{CP}(t)$). Traj Seg represents *trajectory segmentation*. $\mathcal{C}(t)$ are the cost values. $\mathcal{R}$ represents the tranformation matrices, giving the position of the target and of the robot. $\mathcal{M}_t$ is the current state of the robot, $\mathcal{M}_{t+T}$ is desired motion condition for the next control cycle. z^{-1} represents delay of a control cycle.

a human hands over an object to the robot. During the manipulations, the human moves and the different frames defining the task move accordingly. Based on the change of cost values, we divide the trajectory $\mathcal{T}_m$ in Figure 2 into three segments, as illustrated in the configuration space in the left part of Figure 7. In the figure, the points connecting the trajectory segments are depicted by red dots. The first segment $\mathcal{T}_1$, which is defined in the robot frame, has a high risk of auto-collision. When human or object moves, the cost value of collision risk stays the same. Segment $\mathcal{T}_2$ has a lower collision cost value, so modifying the trajectory inside this zone does not introduce high collision risk. The end part, segment for grasping movement $\mathcal{T}_g$, has a high collision cost value. To ensure the grasping succeeds without collision this segment of trajectory should be controlled in the moving object frame.

We name *task frame* the frame in which the trajectory must be controlled. We define a *control primitive* $\mathcal{CP}$ by the combination of five elements: a segment of trajectory, a cost function, a task frame, a control mode, and a stop condition.

$$\mathcal{CP}(t) = (\mathcal{T}_{seg}(t),\ \mathcal{C}(t),\ \mathcal{F},\ O,\ \mathcal{S})^T \qquad (3)$$

In which, $\mathcal{T}_{seg}(t)$ is the trajectory segment, $\mathcal{C}(t)$ is the cost value associated to the trajectory which is monitored during the execution of a control primitive, $\mathcal{F}$ is the task frame, O is the control mode which we will define in next section, and $\mathcal{S}$ is the stop condition of the control primitive. For example, the grasping movement includes five elements: the trajectory segment $\mathcal{T}_g$, the high collision risk cost value $\mathcal{C}(t)$, the

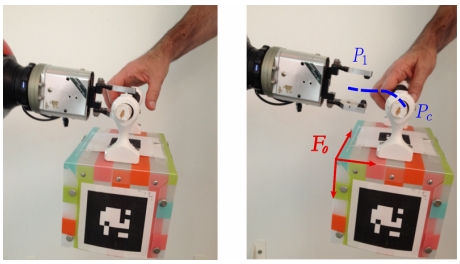

Figure 5: A simple case of grasp. Left: a planned grasp defines contact points between the end effector and the object. Right: To finish the grasping, the manipulator must follow the blue trajectory P_1 - P_c, and then close the gripper. This movement must be controlled in the object frame F_o.

task frame $\mathcal{F}_o$, the control mode as trajectery tracking, and the stop condition $\mathcal{S}$ as a predefined threshold for the distance between the robot end effector and the end point of $\mathcal{T}_g$. In the literature, Manipulation Primitives or Skill Primitives are often the concept for the intermediate level between planning and control and have been discussed in numerous works, as in (Kröger et al., 2011).

Using the definition of control primitives ($\mathcal{CP}(t)$) and Motion Condition: $M(t) = (X(t), V(t), A(t))$, the different components of the trajectory controller and the input and output are presented in Figure 4. The initial trajectory $\mathcal{T}_m$ is segmented into a series of $\mathcal{CP}(t)$. The cost values $\mathcal{C}(t)$ are used during the segmentation, they are also monitored by the controller during execution of a control primitive. The collision checker integrates data from vision, human perception and encoder of the robot. It prevents collision risk by slowing down or suspending the task execution. With all the data and the current Motion Condition $\mathcal{M}_t$ of the robot, different control modes can compute Motion Condition for the next control cycle, which are the input for the robot sorvo system.

Figure 5 shows the last control primitive of *grasping an object*. It is similar to the end part, $\mathcal{T}_g$, of the trajectory in Figure 2. The grasp position, the contact points and the final trajectory are planned by the grasp planner. More details on the grasp planner are given in (Bounab et al., 2008) and (Saut and Sidobre, 2012). When the object moves, the object frame F_o and the path of the trajectory moves also. So to avoid collision, the trajectory of these control primitives must be controlled in the object frame F_o.

3 REACTIVE TRAJECTORY CONTROL

At the control level, a task is defined by a series of control primitives, each defined by a quintuplet. The first level of the proposed trajectory controller is a state machine, which controls the succession of the control modes, the collision managing and other special situations. Target tracking and trajectory tracking are parts of the control modes presented after the state machine.

3.1 Execution Monitoring

A state machine controls the switching between the different control modes associated to each control primitive and monitors the execution. Due to human presence, the robot environment is moving and the control task must be adapted accordingly. The state machine can also suspend or stop the control of a control primitive like depicted in Figure 6.

Suspend Events: When the visual system fails or the target becomes unavailable, or because of some specific human activities based on the monitoring of cost value $\mathcal{C}(t)$, the trajectory controller should suspend the task.

Stop Events: Whatever the control mode chosen, unpredictable collisions can occur and they must stop the robot. Our controller uses two modules to detect these situations. The first one based on (De Luca and Ferrajoli, 2008) monitors the external torques. The second is a geometric collision checker based on results from (Larsen et al., 1999), it updates a 3D model of the workspace of the robot, and runs at the same frequency as the trajectory controller.

Slow Down On Trajectory: Based on the input cost function, the controller can slow down on the main trajectory by changing the time function $s(t)$. Imagine that a fast movement could cause some people anxiety when the robot is close to them, for example. Details about this specific situation can be found in (Broquère, 2011). In this situation, the controller is still executing the task but only at a slower speed.

Each elementary controller based on online trajectory controller is implemented with a simple state machine inside.

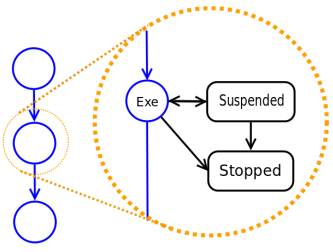

Figure 6: In the left, each circle represents the controller of a control primitive. The system can suspend or stop the execution of a control primitive.

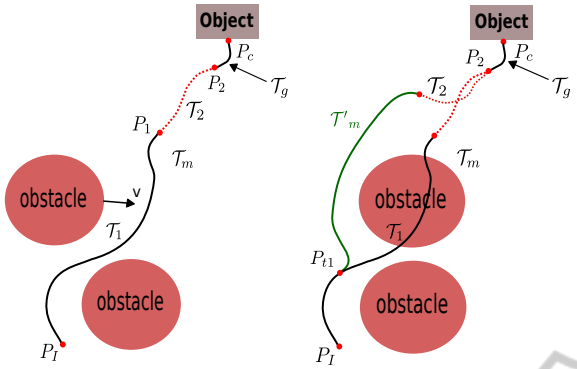

Figure 7: Left: trajectories of the control primitives. Right: trajectory switching for the controller due to the movement of an obstacle.

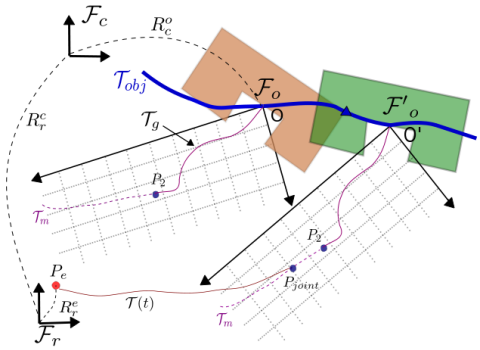

Figure 8: *Control Mode 1.* The robot tracks a point. The object moves to the right, it is drawn at two times: firstly in brown for time t_1 and then in green at time t_2. In both cases, the entry point P_2 of the trajectory $\mathcal{T}_g$ is drawn relatively to the object frame $\mathcal{F}_o$.

3.2 Trajectory Control Modes

Depending on the context defined by the control primitives, different control strategies must be developed. Online trajectory generator gives a flexible solution to build these controllers, which can easily react to unforeseen sensor events and adapt the kinematic parameters, mainly velocity, to the environment context. Switching to a new trajectory or a new frame in which the trajectory is controlled is also possible.

The main idea of the controller is to compute and follow a trajectory while joining up the target trajectory or a target point from the current state. Several control modes are defined to solve the reactive HRI manipulation problem.

Control Mode 1: Target Tracking. If we suppose the robot is in an area without risk of collision, the system can track the end point of the trajectory. In this case, the controller generates iteratively a trajectory to reach the end point and send the first step of this trajectory to a low-level controller. In the special case where the controller does target tracking with visual system, it does visual servoing.

Figure 8 shows the details of the trajectory control mode for *Target Tracking*. The object is at position O at current time, and moves following the curve $\mathcal{T}_{obj}$. This curve is obtained by a simple Kalman filter, building a movement model from the results of 3D vision system. $\mathcal{F}_r$ is the robot base frame, $\mathcal{F}_c$ and $\mathcal{F}_o$ are camera frame and object frame, respectively. Also, R_r^c is the 4×4 transformation matrix from $\mathcal{F}_r$ to $\mathcal{F}_c$ and R_c^o the transformation matrix from $\mathcal{F}_c$ to $\mathcal{F}_o$. They are all in dashed line and they change with time when the humans or objects move. Initially, the robot is at point P_e, since there is no risk of collision, the controller can simply track point P_2, which is the end point of the segment. It is also possible for the robot to join up the trajectory at another point P_{joint} defined in the object frame which is the task frame. The details of the algorithm is given in Algorithm 1 where:

T: duration of one control cycle.
$\mathbf{M_r}$: current motion condition of the robot, so $M_r = (X_r, V_r, A_r)$.
δ: distance threshold to stop the tracking process.
$\mathbf{M_g(t)}$: motion conditions at time t on trajectory $\mathcal{T}_g$.
$\mathbf{M_{P_2}}$: motion conditions of the target P_2 on the main trajectory, which is calculated by the planner.
MaxLoop: the maximum times the loop repeats for the controller to track the target or the trajectory. Once the time exceeds the value, the trajectory controller is suspended and a signal is sent to the supervisor, requiring the replanning of new task or a new path to realize the task.
X, Q: input signal in Cartesian space and in joint space for low-level controller.

Algorithm 1: Control for target tracking (*Control Mode 1*).

input : Target point P_2;
while
$(distance(P_2, M_r) > \delta) \wedge (Loop < MaxLoop)$
do
  system time $t = t + T$, $Loop = Loop + 1$;
  Update perception data;
  if *Collision Detected* **then** Emergency stop;
  if *Suspend Events Occur* **then** Suspend task;
  Coordinates transformations;
  Generate Type V control trajectory $\mathcal{T}(t)$, for which: $IC = M_r$, $FC = M_{P_2}$;
  $X = getMotion(t + T, \mathcal{T}(t))$;
  Inverse kinematics: $X \rightarrow Q$;
  Q to the position servo system;
end

Control Mode 2: Trajectory tracking in Task Frame. Once robot reaches point P_2, it starts the grasping movement, which corresponds to $\mathcal{T}_g$ in Figure 7. The object is still moving, but as the robot is in the high cost zone, it should track the main trajectory in the task frame. The details of the control mode is given in Algorithm 2.

Figure 9 shows the details of the control mode, all the frames and object movements are the same as in Figure 8, but the robot is at point P'_e. The robot tracks $\mathcal{T}_g$ in the frame $\mathcal{F}_o$, and will end up executing $\mathcal{T}'(t)$ in the robot frame.

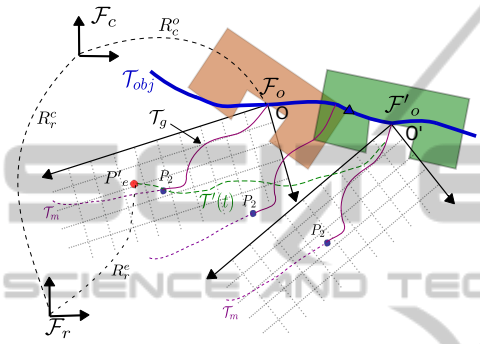

Figure 9: *Control Mode 2.* Control mode. Object at time t_1 is colored in light brown, and green at time t_2. It follows a movement model given as the blue trajectory $\mathcal{T}_{obj}$. The purple trajectory for grasping $\mathcal{T}_g$ stays unchanged in the object frame. The robot tracks the trajectory $\mathcal{T}_g$ as it does the grasping movement.

Algorithm 2: Control for trajectory tracking in a moving work frame (*Control Mode 2*).

input : Trajectory segment $\mathcal{T}_g$;
while
$(distance(P_c, M_r) > \delta) \wedge (Loop < MaxLoop))$
do
  system time $t = t + T$, $Loop = Loop + 1$;
  Update perception data and object movement model;
  if *Collision Detected* **then** Emergency stop;
  if *Suspend Events Occur* **then** Suspend task;
  Coordinates transformations;
  $M_{\mathcal{T}_g} = getMotion(t + T, \mathcal{T}_g)$;
  $M_{object} = getMotion((t + T, \mathcal{T}_{obj})$;
  $X = M_{\mathcal{T}_g} + M_{object}$;
  Inverse kinematics: $X \rightarrow Q$;
  Q to the position servo system;
end

Control Mode 3: Path re-planning and trajectory switch: during the execution, a path can be re-planned, for example when an obstacle moves (see Fig. 7). A new trajectory is computed and given to the controller that switches to the new trajectory. While the controller is following the trajectory $\mathcal{T}_m$, an obstacle moves and invalidates the initial trajectory. Then the system provides the controller with a new trajectory $\mathcal{T}'_m$ beginning at time t_1 in the future. The controller anticipates the switch, and when the robot reaches P_{t_1} at time t_1, the robot switches to the new trajectory $\mathcal{T}'_m$. Because the new trajectory $\mathcal{T}'_m$ is calculated using the state of the robot at time t_1 as its initial condition, the trajectory is switched without problem. The controller keeps the possibility of path re-planning. In some cases, a new path is needed to accomplish the task.

In this paper, we essentially solved the problem of the task of grasping a moving object held by the human counterpart. For other tasks, like picking an object, giving an object to human or putting an object on the table, the same functionalities can also be used. For example, putting an object on a moving platform would require the end segment of the main trajectory to be controlled in the frame of the platform, which moves in the robot frame. Likewise giving an object to a moving human hand will require the manipulator to track the exchange point, normally planned by a human-aware motion planner till the detection that human grasps the object successfully. Although a general algorithm to decompose the tasks into control primitives is still to develop, the basic HRI tasks can all be controlled by the control modes discussed above.

4 MANIPULATION RESULTS

We focus on some results of how the controller is integrated in a HRI manipulator. For the performance of the trajectory generator, readers may refer to (Broquère et al., 2008) and (Broquère, 2011).

The controller has been implemented on the robot Jido at *LAAS-CNRS*. Jido is a mobile manipulator , built up with a *Neobotix* mobile platform *MP-L655* and a *Kuka LWR-IV* arm. Jido is equipped with one pair of stereo cameras and an *ASUS Xtion* depth sensor. The software architecture that we used for Jido is presented in 1.1. Note that the pan-tilt stereo head may move during manipulations, then the transformation matrix from robot frame to camera frame is updated at the same frequency as the controller. The stereovision system uses marks glued on manipulated objects for localization. Unfortunately, the localization of these marks is highly sensible to lighting conditions and the estimated position of the object O in the robot frame $\mathcal{F}_r$ is very noisy and unstable. Figure

10 shows the results returned by the 3D-vision system in poor lighting conditions and the result given by a Kalman filter. Even when raw data oscillates, this filter is capable to reduce the offset at the price of an acceptable delay. This reduces the oscillations of the robot arm too.

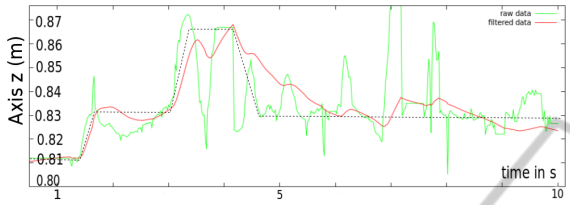

Figure 10: Instability of the 3D vision in poor lighting condition. The green curve shows axis z of the localization result of an object in the robot frame over 10 seconds and the red one shows the filtered result. The object was held by a human that intended to move the object accordingly to the black dashed curve.

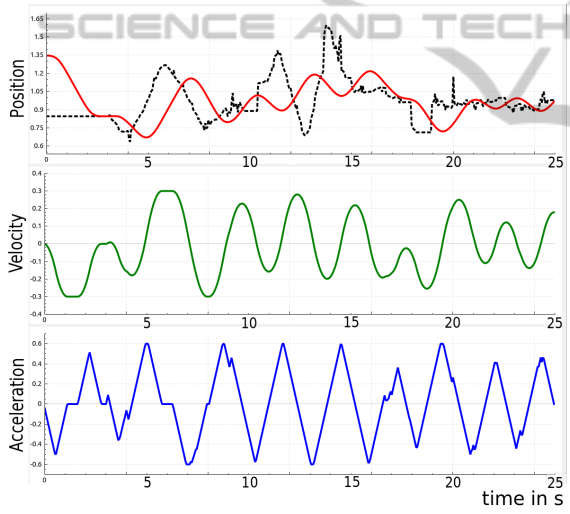

Figure 11: Results of robot tracking a target: position (in m), velocity (in m/s) and acceleration (in m/s^2) during the tracking for 25 seconds. The black dashed line is the target position, with noise of the 3D vision, and the red line is the position of the robot, which tracks the target with a delay. The positions of the robot are calculated from measured joint values and the kinematic model, while velocity and acceleration are estimated. The velocity, acceleration and jerk are always limited, maintaining a smooth tracking process.

Figure 11 shows the results of the target tracking by the trajectory controller, as in *Case 2*, over 25 seconds. For simplicity, only axis X is shown. The black dashed line is the position of the target, generated by the 3D vision system. The red line is the position of the robot. The two bottom diagrams show the velocity and acceleration of the robot in the same period.

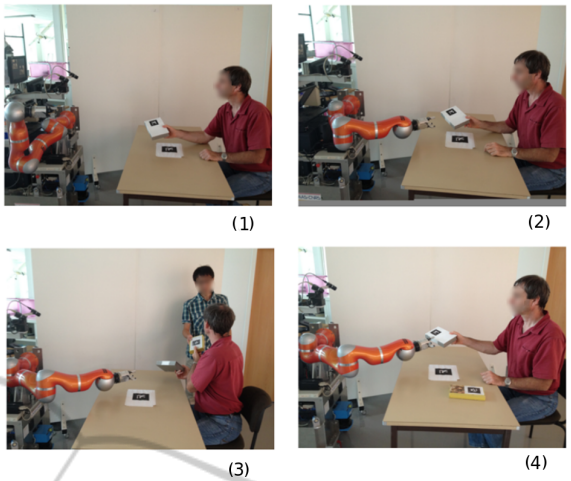

Figure 12: (1): The controller is given the task of receiving the object from human. It tracks the first segment in the robot frame. (2): The object is moving and the robot tracks a target. (3): Human is distracted by another human and the task is suspended. (4): Human returns to the task, and the robot resumes the task and grasps the object.

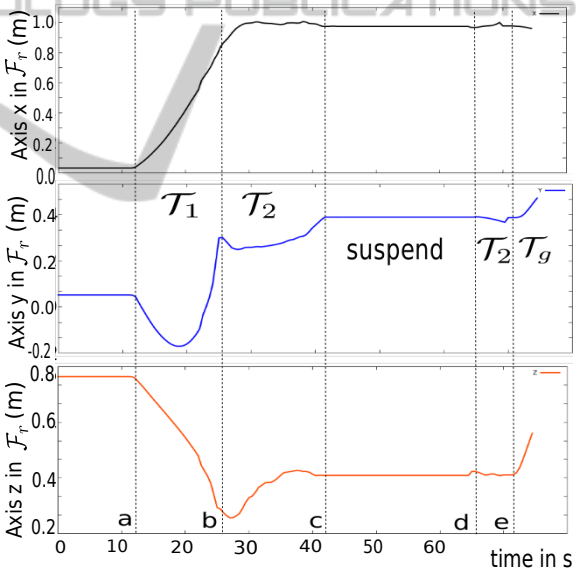

Figure 13: Real position of the robot arm end effector in the robot frame. The motion starts at time a; Between a and b: the controller tracks the first trajectory segment $\mathcal{T}_1$ in $\mathcal{F}_r$; From b to c and d to e: target tracking; From c to d: the task is suspended; From e to end: the grasping movement controlled in $\mathcal{F}_o$.

Firstly, we can see that the controller produces robust behavior to the noise in the visual system. Secondly, the velocity and acceleration of the robot are saturated as type V trajectories and calculated.

Finally, we show the behavior of the controller for a complete manipulation task. Figure 12 shows the scenario of the manipulation and Figure 13 shows the

real position of the robot end effector in the robot frame (see figure 2 for the axes assignment of the robot base). The high-level task planner plans a task to receive the object. When the robot sees the object held by the human, the grasp planner calculates a valid grasp and the path planner with the trajectory generator plans the main trajectory for the robot to take the object.

The trajectory is divided into three segments by the controller, and different control modes are chosen. As we as seen above, each control primitive is associated to a trajectory segment. In this case, we obtain three segments, the first one is controlled in the robot frame, the second is defined as the tracking of the entry point of the third segment and the third segment is a trajectory defined in the object frame.

During the target tracking, human is distracted because a second human arrives and gives an object to him. High-level software detects this event by monitoring the visibility cost map of the human. Because of the event, the controller suspends the task. It resumes the tracking when the human look again at the robot and the object to exchange come back in the reachable zone. Then, the grasping movement is finished. Note the performance of the target tracking process in the time intervals: between *b* to *c*, and between *d* to *e*. The controller finished the task reactively without the need of task or path replanning. The results shows that a reactive controller can be built based on Online Trajectory Generation, and as it is more responsive for the human, the robot is easier to interact with. Before the implementation of the reactive controller, the human needs to hold still the object for the robot to grasp successfully.

5 CONCLUSIONS

A reactive trajectory controller has been presented with some results relative to a robot grasping an object held by a human. The first results presented illustrate the versatility of the controllers based on online trajectory generation. In the example shown here, the controller switch between frames and suspend the control task during the time the human is distracted.

The trajectory controller proposed uses an online trajectory generator to build a trajectory to join up the trajectory to follow. It is very simple to use and implement and gives an efficient solution to follow trajectories and track moving objects in the HRI context. More precisely, it can adapt kinematic limits to the changing state of the scene and switch between trajectories and control modes.

The challenge is now to extend this type of trajectory controller and the concept of control primitives to manage forces, to handle events based on force sensing and to control dual arm manipulators.

ACKNOWLEDGEMENTS

This work has been supported by the European Community's Seventh Framework Program FP7/2007-2013 "SAPHARI" under grant agreement no. 287513 and by the French National Research Agency project ANR-07-ROBO-0011 "ASSIST" and ANR-10-CORD-0025 "ICARO".

REFERENCES

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 – 483.

Biagiotti, L. and Melchiorri, C. (2008). *Trajectory Planning for Automatic Machines and Robots*. Springer.

Bounab, B., Sidobre, D., and Zaatri, A. (2008). Central axis approach for computing n-finger force-closure grasps. *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1169–1174.

Broquère, X. (2011). *Planification de trajectoire pour la manipulation d'objets et l'interaction Homme-robot*. PhD thesis, LAAS-CNRS and Université de Toulouse, Paul Sabatier.

Broquère, X. and Sidobre, D. (2010). From motion planning to trajectory control with bounded jerk for service manipulator robots. In *IEEE Int. Conf. Robot. And Autom.*

Broquère, X., Sidobre, D., and Herrera-Aguilar, I. (2008). Soft motion trajectory planner for service manipulator robot. *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2808–2813.

Buttazzo, G., Allotta, B., and Fanizza, F. (1994). Mousebuster: A robot for real-time catching. *IEEE Control Systems Magazine*, 14(1).

Calinon, S. and Billard, A. (2004). Stochastic gesture production and recognition model for a humanoid robot. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2769 – 2774 vol.3.

Chaumette, F. and Hutchinson, S. (2006). Visual servo control. part i: Basic approaches. *IEEE Robotics and Automation Magazine*, 4(13).

Chaumette, F. and Hutchinson, S. (2007). Visual servo control. part ii: Advanced approaches. *IEEE Robotics and Automation Magazine*, 1(14).

De Luca, A. and Ferrajoli, L. (2008). Exploiting robot redundancy in collision detection and reaction. In *Intelligent Robots and Systems, 2008. IROS 2008.*

IEEE/RSJ International Conference on, pages 3299 – 3305.

De Santis, A., Siciliano, B., De Luca, A., and Bicchi, A. (2008). An atlas of physical human–robot interaction. *Mechanism and Machine Theory*, 43(3):253–270.

Farrokh, J.-S., Lingfeng, D., and William, J. (2011). Comparison of basic visual servoing methods. *IEEE/ASME Transactions on Mechatronics*, 16(5).

Fleury, S., Herrb, M., and Chatila, R. (1997). Genom: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *IEEE/RSJ Int. Conf. on Intel. Rob. And Sys.*

Gosselin, G., Cote, J., and Laurendeau, D. (1993). Inverse kinematic functions for approach and catching operations. *IEEE Trans. Systems, Man, and Cybernetics*, 23(3).

Haschke, R., Weitnauer, E., and Ritter, H. (2008). On-Line Planning of Time-Optimal, Jerk-Limited Trajectories. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008*, pages 3248–3253.

Kröger, T. (2010). *On-Line Trajectory Generation in Robotic Systems*, volume 58 of *Springer Tracts in Advanced Robotics*. Springer, Berlin, Heidelberg, Germany, first edition.

Kröger, T., Finkemeyer, B., and Wahl, F. (2011). *Manipulation Primitives A Universal Interface between Sensor-Based Motion Control and Robot Programming*, volume 67 of *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg.

Kröger, T. and Padial, J. (2012). Simple and Robust Visual Servo Control of Robot Arms Using an On-Line Trajectory Generator. *2012 IEEE International Conference on Robotics and Automation*.

Kröger, T., Tomiczek, A., and Wahl, F. (2006). Towards on-line trajectory computation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, China*. Citeseer.

Larsen, E., Gottschalk, S., Lin, M., and Manocha, D. (1999). Fast proximity queries with swept sphere volumes.

Liu, S. (2002). An on-line reference-trajectory generator for smooth motion of impulse-controlled industrial manipulators. In *7th International Workshop on Advanced Motion Control*, pages 365–370.

Mainprice, J., Sisbot, E., Jaillet, L., Cortés, J., Siméon, T., and Alami, R. (2011). Planning Human-aware motions using a sampling-based costmap planner. In *IEEE Int. Conf. Robot. And Autom.*

Mainprice, J., Sisbot, E., Siméon, T., and Alami, R. In *IARP Workshop on Tech. Challenges for Dependable Robots in Human Environments*.

Saut, J.-P. and Sidobre, D. (2012). Efficient models for grasp planning with a multi-fingered hand. *Robotics and Autonomous Systems*, 60.

Sidobre, D., Broquère, X., Mainprice, J., Burattini, E., Finzi, A., Rossi, S., and Staffa, M. (2012). Human–robot interaction. *Advanced Bimanual Manipulation*, pages 123–172.

Sisbot, E., Ros, R., and Alami, R. (2011). Situation assessment for human-robot interactive object manipulation. *20th IEEE International Symposium on Robot and Human Interactive Communication*.

Sisbot, E. A., Marin-Urias, L. F., Alami, R., and Siméon, T. (2007a). Spatial reasoning for human-robot interaction. In *IEEE/RSJ Int. Conf. on Intel. Rob. And Sys.*, San Diego, CA, USA.

Sisbot, E. A., Urias, L. F. M., Alami, R., and Siméon, T. (2007b). Spatial reasoning for human-robot interaction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, San Diego, CA, USA.

Strabala, K. W., Lee, M. K., Dragan, A. D., Forlizzi, J. L., Srinivasa, S., Cakmak, M., and Micelli, V. (2013). Towards seamless human-robot handovers. *Journal of Human-Robot Interaction*, 2(1):112–132.

Vakanski, A., Mantegh, I., Irish, A., and Janabi-Sharifi, F. (2012). Trajectory learning for robot programming by demonstration using hidden markov model and dynamic time warping. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 42(4):1039 –1052.