Phantom Assignment 2024
Biomedical Robotics

Valentina Condorelli, Annika Delucchi, Ramona Ferrari, Daniele Rialdi
Group 1

January 21, 2025

1 Reaching Task

1.1 Introduction

The main objective of this exercise is to create an interactive window where colored balls appear and the user must reach using the robot. For this reason is so called "Reaching Task". The software used is *Matlab Simulink*, where it is possible to use *VR toolbox*, a tool for ambient and robot simulations.
The robot used during the lab session is Phantom Omni Robot (Figure 1) and it can provide, by its serial structure, both force feedback and toch feeling during manipulation of virtual objects.

Figure 1: Phantom Omni Robot.

1.2 Description of the Exercise

The final assignment has to be tested with the real robot; actually, for proving the correctness of the work before going in the laboraotry, a simplified and practical version of the assignment is developed.

1.2.1 Reaching Task with the Mouse

What is described in this part is in "Reaching mouse" file. As specified in the instructions, a **list of 8 angles** is generated by dividing a 360° angle into quarters of 45°, resulting in 8 distinct angles. Since the very beginning, the robot block was commented out as there was not the possibility to test the simulation on our own; thus, another solution is provided by involving the mouse.
Mouse input is the subsystem used to retrieve mouse coordinates in the three dimensional space; to be more precise, only the x and y coordinates are relevant, while z is not fundamental as the cursor motion happens in a plane. For this reason the z coordinate is set to 1 statically. This is also necessary to visualize the cursor in the VVR window.
Then, both the angles and the mouse input enter in a **State Machine** block. This block is fundamental because it defines the "rules" of the simulation, depending on the user input.

The first step is to initialize some useful parameters, such as the radius of the spawned balls and some counters for knowing how many times a specific action is completed. The list of angles is created outside the state machine (as said) and then sorted by an ad hoc

implemented function ”randomAngle”. This is done to avoid having the angles repeated consecutively or always in the same order. The ball are spawned at a proper angle, specifying the color and the target position. The target position is computed by another ad hoc function, called ”computedTargetPose”, that calculates the mapping between the angle and the coordinates in two dimensional space (the plane of the simulation window).
If a target is reached and held by the user cursor for 1s, the target changes its colour and then disappears; at this point the user is requested to reach the target home again and hold the position for 2s.

This process is iterated for 10 times, for each all the 8 targets are reached and only then the simulation stops. The number of iterations is tracked by the initiliazed counter at the beginning of the state machine.
Each condition in the State Machine block, when verified, triggers a change of the simulation state.
From this block, the target position, the colours and thus the stop condition, are retrieved for having the correct visualization in the VR window; not to mention the need of the position of our mouse.
The first simulations were necessary to tune the parameters, such as the threshold for determining when the cursor can be considered inside the target or to validate the correctness of the solution.

After trying the solution with the mouse, it was possible to test with the real robot.

The Simulink block used to address this exercise is the following:

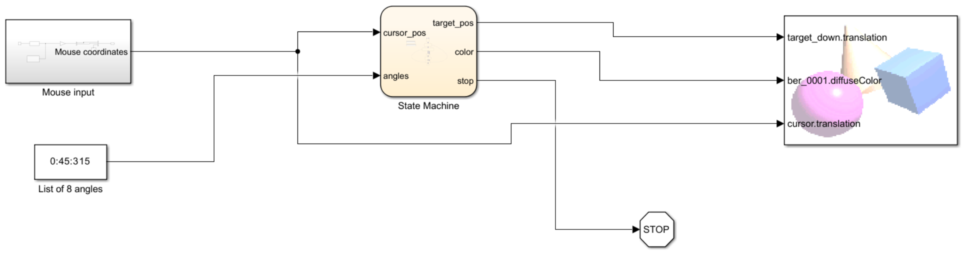


Figure 2: Simulink block for Reaching Task with mouse.

1.2.2 Reaching Task with Phantom Robot

This part of the exercise was developed in the ”Reaching” file. During the laboratory session, the robot was connected to the PC running the code. This file is quite similar to the previous one, but the part about retrieving mouse position is substituded by the **Model of the Robot** (Figure 3).

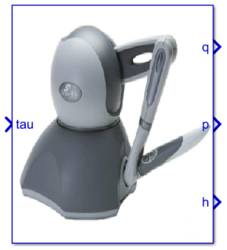


Figure 3: Block for Phantom Omni Robot (PhanTorque 3Dof).

The block takes as tau input a null 3D vector and has several outputs: Joint Position q, Cartesian Position p and the Homogeneous Transformation Matrix H. Since it is necessary to retrieve the end-effector position in the VR widow, p is necessary and it is given as input, multiplied by a gain, to the State Machine and the VR window, as shown in Figure (4).

The interesting part is the control of the position of the cursor on the screen: at the beginning it was necessary to calibrate the robot and then, the user can use the robot's terminal to change the cursor position in the window, to reach the target balls.

The Simulink block of this exercise is the following:

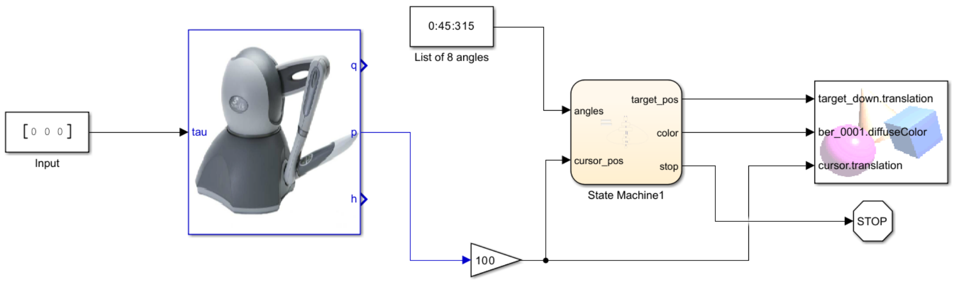


Figure 4: Simulink block for Reaching Task with real robot.

2 Force Fields Task

The goal of the second part of the assignment is to use the robot to simulate **force fields** when the user is moving the robot's terminal. The force fields to be implemented are the following two: one based on the position and the other on the velocity of the Phantom Omni's end-effector. The position-based field will generate an attractive force that guides the cursor toward the target, while the velocity-based field will produce a viscous force that resists the end-effector's motion.

2.1 Description of the Exercise

The file accomplishing this exercise is "forceField" and it involves the robot model of Figure 3. For simulating a force field it is necessary re-use output information from the block as input; precisely, the Joints positions are needed to calculate the Jacobian matrix, subsequently transposed.

The Cartesian Position of the end-effector is retrieved directly from the PhanTorque block and it is used on the two branches of Simulink scheme:

1. the cartesian position is substracted in a sum node with an initial offset. Their sum is then multiplied by a gain matrix: in this way, the force field based on position is computed.

2. the cartesian position is used also for retrieving the velocity, using an ad hoc subsystem. Then, the velocity is multiplied by another gain matrix.

Both the previous quantities are sum together and the resultant matrix is multiplied by the previous Jacobian. The final result of this last multiplication is τ vector, representing the torque requested as input from the Phantom Robot.

The following figure shows what explained before:

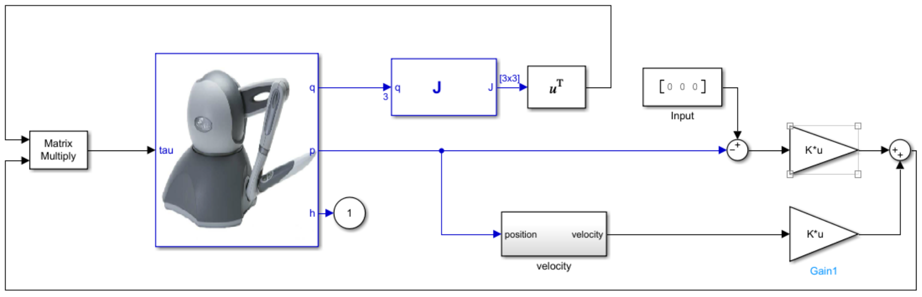
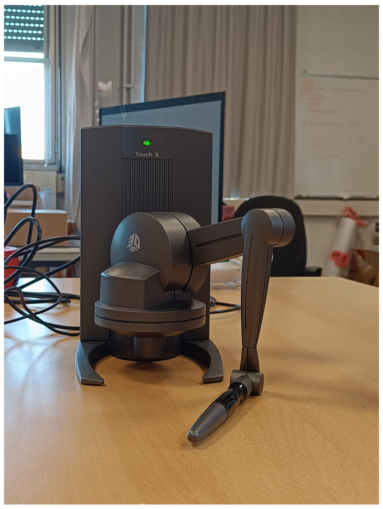


Figure 5: Simulink blocks for Force Fields.

Define how to compute (and comment) a field that attracts the end effector toward the target? How to have a field that rejects from the target instead? How to compute (and comment) a viscous field that is opposed to the velocity of the end effector?

- **Attractive Field Implementation**: This field is created by calculating the positional error between the current Cartesian position of the end-effector (provided by the "Phan Torque 3dof" block) and the target position. The error is then scaled by a 3x3 diagonal matrix with positive diagonal entries, representing the strength of the attraction in each direction. The z-component of the matrix is set to 0 to restrict the movement to the 2D plane, ensuring the robot moves to the target without unnecessary movement along the z-axis.
- **Viscous Field Implementation**: This field is based on the joint position vector, which is used to calculate joint velocities via a "Describe Derivative" block. The velocity is processed through a Discrete Transfer Function to simulate a resistive damping force proportional to the joint velocities. The resulting force is again scaled by a 3x3 matrix with adjusted damping coefficients. As before, the z-component is set to zero to apply the viscous resistance only in the 2D plane, preventing any damping effect along the z-axis.

3 Images taken during laboratory session.



(a) Real robot.

(b) Testing simulation.

Figure 6: Images taken during laboratory session.