Obstacle Avoidance using Dynamic Movement Primitives and Reinforcement Learning

Experimental Details

July 2025

1 Pick-and-Drop

The pick-and-drop experiment requires picking and carrying a small cube to the other side of an obstacle and into a cup. The position of the small cube and the cup are sampled randomly inside two workspaces with dimensions 0.4×0.4 m on opposite sides of the obstacle. The dimensions of the obstacle are defined by the arrangement of up to 20 blocks, each of size $0.1 \times 0.05 \times 0.12$ m. Between one and seven blocks are selected randomly along the Y axis ($n_Y \in [1, 7]$) and between one and four blocks are stacked along the Z axis ($n_Z \in [1, 4]$). The Y location of the blocks depends on n_Y, such that the obstacle arrangement is centered at $Y = 0.5$ m, while $X = 0$ is constant for all blocks.

1.1 Point cloud detection

The cup that is used to drop the cube into, has $X \times Y$ dimension of 0.07×0.07 m. After detecting the cup, the goal position is known. Together with the known start position of the end-effector, the center point of the trajectory is computed and an oriented bounding box is created (see Fig. **??**). Now, the point cloud is separated into four sections: left, right, top, bottom. In each section, the point with the maximum distance from the trajectory center can be computed. Since it is not possible to go below the obstacle, the bottom section is not considered. Only if there are no points in the top section. In that case, the minimum distance of the bottom section is captured as negative distance. This allows the detection node to be independent of robot gripper dimensions that are considered later in the application node. Fig. **??** a) shows the preliminary task parameter $\hat{s}_1$ for a point mass when avoiding to the left (L) or right (R). The third option is $\hat{s}_1^{up}$, avoiding the obstacle by going above. Given the gripper dimensions, the final task parameter can be computed as $s_1 = \hat{s}_1 + D_G + o$, including the offset o. D_G considers the gripper and grasped object dimensions. When attempting to avoid the obstacle to the left or right, half of the gripper width w_G is added such that $D_G = 0.5 w_G$. When avoiding upwards $D_G^{(up)} = 0$, as the grasped object is right at the considered tip of the end-effector. After normalization w.r.t. the distance between start and goal $s_1^{NN} = s_1 / L$, the neural network outputs the forcing term parameters that are set to the desired DMP dimensions.

2 Insertion

2.1 Data generation

The precise vertical descend is defined by a 1% tolerance relative to the trajectory length L. Except the S_{scope} that avoids collisions with the ground, a second is set with $\hat{v} = -L$, $m = 0.005L$ away from the goal location for the X positions of the trajectory. Additionally, s_2 is always set with $p_2 = 0.995L$ to constrain the other side of the descend. Five positions for $p_1 \in [0.067L, 0.87L]$ are set uniformly to simulate different arbitrary collision scenarios. The optimization target is set to $S^*_{shape} = -0.33L$.

2.2 Neural network training and performance

In less than 18 minutes one thousand trajectories are generated and the neural network is trained, now with an output dimension of $2 \times 20 = 40$. Testing the model accuracy shows a 100% avoidance success

rate with $o = 0.02$, applied as $s_1 + o$, $s_2 - o$.

2.3 Point cloud detection

The peg and the box with the hole must be detected in the point cloud. After removing all points that don't belong to either peg or box, the peg and box can be distinguished by there dimensions. The peg has dimensions of $d_p = 0.04 \times 0.04 \times 0.1$ m, the outer dimensions of the box are $d_b = 0.1 \times 0.1 \times 0.04$ m, the dimensions of the hole are $d_h = 0.045 \times 0.045 \times 0.04$ m. Using DBSCAN as before, two clusters are found and their centers are used to define the positions of the peg and the hole. The task parameters are then computed based on the known object dimensions, the grasp height $h_p = 0.02$ m and the distances D_{0p},D_{ph} between the initial end-effector position P_0 and the peg position P_p and between P_p and the hole position P_h. The task parameters for the pick trajectory are then $s_1^{pick} = h_p$ and

$$s_2^{pick} = D_{0p} - 0.5 d_p^{max} - 0.5 w_G,$$

and for the insertion, $s_1^{insert} = d_{b,Z}$ and

$$s_2^{insert} = D_{ph} - 0.5 d_p^{max} - 0.5 d_b^{max},$$

where $d^{max} = \sqrt{2} d_X$ for either the peg or box. This ensures the generation of a collision-free trajectory avoiding the maximum diagonal dimension for both square objects.

3 3D Avoidance

3.1 Data generation

The training scenarios are discretized uniformly, considering 5 different wall configurations $C_w \in [-0.6, 1.0]$ and 6 different gap locations $P_{gap} \in [0, -S^*_{shape}]$, leading to a total number of $5 \times 6 = 30$ PI^2 optimization runs. The wall configurations describe the fraction of the smaller wall where $C_w = 1.0$ is the case they have the same height. Which of the two walls is smaller is defined by the sign, $C_w > 0$ refer to $s_1 < s_2$ and vice versa. For the optimization the first wall is defined by $p_1 = 0.15L$ and $p_2 = 0.2L$, the other wall is defined by $p_3 = 0.8L$ and $p_4 = 0.85L$. In this case, the cost S_{shape} is computed for each point individually (p_1 at t_1 etc.) instead of the integral. Hence, $\mathbf{u}(t_1, t_2, t_3, t_4)$ is a vector with four elements where one wall pair is scaled with $|C_w|$ to keep the desired height difference for each generated trajectory. Besides the S^{bottom}_{scope}, four additional scope cost functions constrain the trajectory with two upper bounds and two lower bounds on the Y and Z coordinates of the trajectory inside the gap defined by $p_5 = 0.35L$ and $p_6 = 0.65L$, with $m = 0.013L$. The collection for the dataset starts once the scope constraints are satisfied. The optimization target is set to $S^*_{shape} = -0.33L$. Given the tight constraints for this model, the PI^2 optimization takes more time per run, compared to the previous models. For that reason, and because the sharp direction changes require higher acceleration and jerk, the $S_{acc,0}$ and S_{jerk} costs are omitted here.

3.2 Neural network training and performance

The PI^2 optimization of 30 runs generate 7.4k trajectories. The NN model outputs $3 \times 60 = 180$ forcing term parameters. Tests show more overfitting to the training data than the previous models. The interpolation between the ratios of the two wall heights C_w is not captured accurately enough to achieve a 100% avoidance success rate. When selecting scenarios among the same C_w used in the training, an offset of $o = 0.1$, applied to s_1, results a 100% success rate, interpolating only along the gap position P_{gap}. Unlike avoiding around or over an obstacle, a gap does not allow setting an offset, as collision can occur on both sides. The tests show that the learned model produces maximum errors in the gap of $e_Y = [-0.007L, 0.03L]$.

3.3 Point cloud detection

As the $X - Y$ positions of the walls are constant and known, their height in w_Z is measured by taking all points within a small area and taking their mean Z value. The location of the gap is known to be

along the Y axis. The distances between all points within a small area along the Y axis are computed. The maximum Y distance is between the points on either side of the gap. From this, the inputs for the neural network can be computed: $s_1^{NN} = max(w_{1,Z}, w_{2,Z})/L + o$ is the maximum relative height plus offset, $s_2^{NN} \in [-0.6, 1.0]$ maps the order of both walls and the ratio of their heights to fit the training configurations C_w. s_3^{NN} is computed as s_1^{NN} in the previous experiments, but without adding an offset.