Local Dynamic Window

A Local Dynamic Window (LDW) Approach was implemented in MATLAB with the score function:

$$G(v,\omega) = \alpha\text{heading}(v,\omega)... + \beta\text{dist}(v,\omega) + \gamma\text{velocity}(v,\omega) \quad (1)$$

The effect of these different parameters have been tested, the results are shown in figure 1.

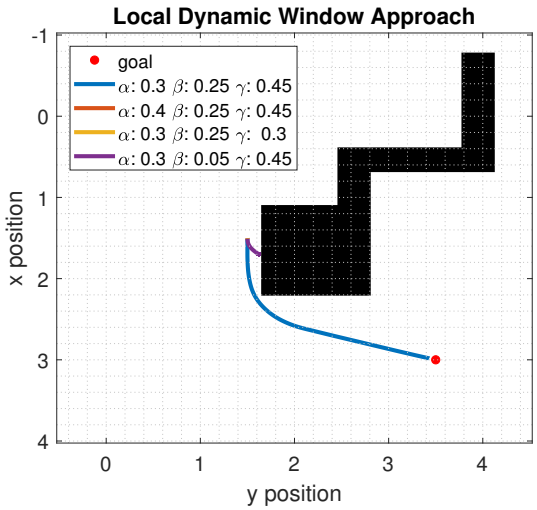

Figure 1: LDW: The effects of different weight factors

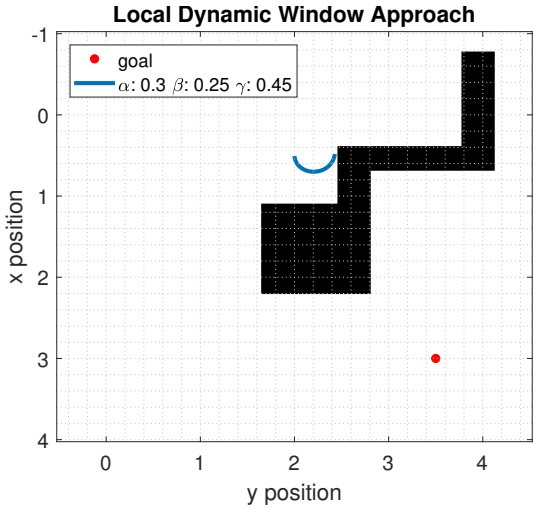

Figure 2: LDW: Bob can easily get stuck

With the 1^{st} set of weight factors Bob reaches its goal in a fairly efficient manner. In the 2^{nd} and 3^{th} cases however Bob gets stuck at its starting position. The second set of weight factors with an high α tries to align Bob towards to goal position, this seems to conflict with the obstacle distance term preventing Bob to move. With the 3^{th} set of weight factors Bob gets little reward for choosing fast solutions, therefore it decides not to move at all. In case 4 β is fairly low, which means Bob is no longer trying to avoid obstacles, causing him to bump into a wall. We can see from figure 2 that Bob can get stuck with the first set of weight factors when initialized at a different position.

Gloabl Dynamic window

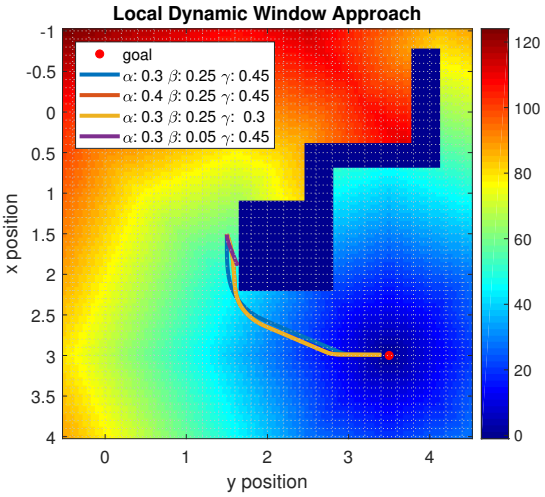

Figure 3: GDW: The effect of different weight factors

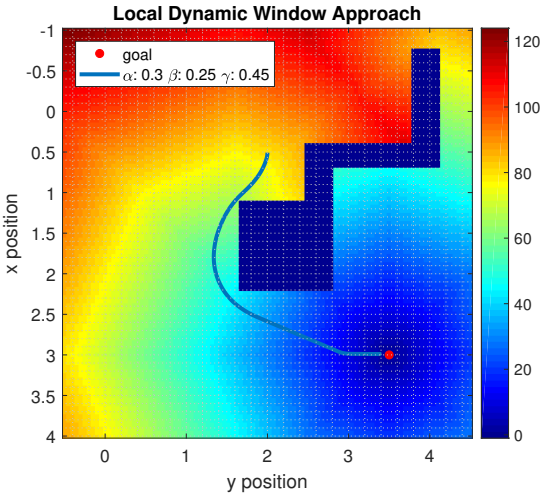

Figure 4: GDW: Bob no longer gets stuck

We noted that the LDW approach has several flaws, it is very dependent on the weight factors of the cost function, and it can get stuck fairly easily. Therefore the the method is enhanced with global guidance using Dijkstra's Algorithm. With the Global Dynamic Window

(GDW) Approach Bob aligns with the gradient of the navigation function instead of towards the goal position. Again a test was run with the same weight factors as before, the results are shown in figure 3.

The achieved results are much better than before, three sets of weight factors were able to reach the final goal. Furthermore Bob is now able to break out of more difficult starting positions as can be seen in figure 4.

2.1 Tuning Score Function

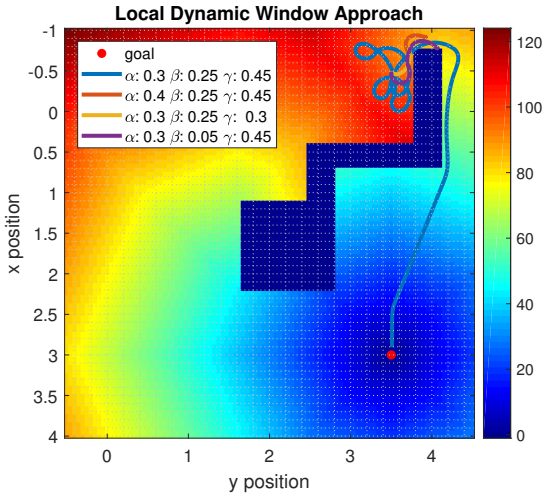

Figure 5: GDW: Chaotic behaviour for certain initial condition

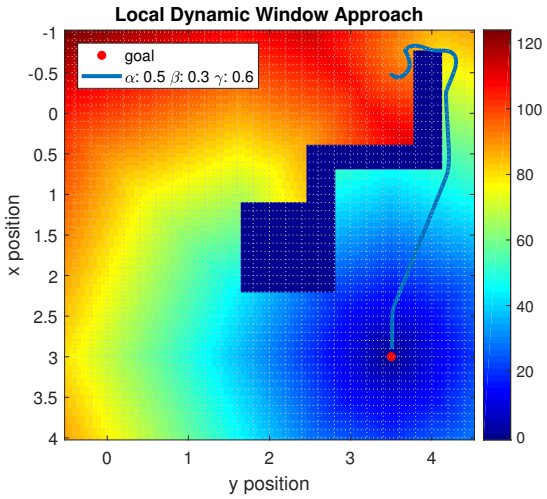

Figure 6: GDW: Result of tuned weight factors

We can see from figure 5 that with the previously used weight factors we can get very chaotic behaviour for certain initial conditions. Only the first set of weight factors is able to reach the goal, the others get stuck or bump into a wall. We can see that the second set of weight factors behaves fairly nice, however it gets stuck after a while. This was solved by increasing α a bit, causing Bob to try to align with the gradient even more. Furthermore the obstacle distance penalty β was slightly raised to prevent Bob from bumping into any walls. Also the velocity reward γ was slightly raised to prevent Bob from getting stuck. The final result can be seen in figure 6.

V-Rep Simulation

Finally the GDW Approach was used in a V-REP simulation, with the default weight factors for this and the previous simulation, the results are shown in figure 7.

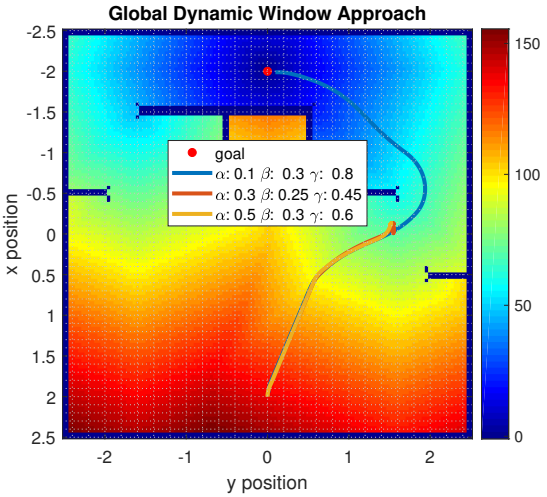

Figure 7: Results from V-REP simulation

We get good performance for the first weighting factors, however the weighting factors from the previous simulations gets stuck at the location where Bob has to move between the two walls. This is caused by the velocity constant γ being fairly low compared to the obstacle distance penalty β. Therefore Bob decides it is better to stop moving instead of moving in-between the walls. The V-REP set of weighting factors have a much higher γ to prevent this from happening.

The above explanation is strongly related to why Bob still can get stuck. This behaviour is inherent from maximizing the score function mentioned in equation 1. In certain cases Bob has to move in the direction of an obstacle, like the parallel walls in figure 7. However this would make the score decrease and an optimum might be found at not moving at all.