Comparison of Optimal Path Planning Algorithms for an Autonomous Mobile Robot

Micho Radovnikovich, Ka C. Cheok and Pavan Vempaty
Department of Electrical and Computer Engineering
Oakland University, Rochester, MI 48309

***Abstract*—In this paper, an optimal control approach is used to solve a two-dimensional path planning problem for a differential drive mobile robot. Two optimal control algorithms are presented and analyzed, which consist of a novel implementation of a linear quadratic tracker (LQT), and a dynamic programming (DP) scheme. The algorithms are applied to the task of GPS navigation with obstacle avoidance. The methods aim to find an optimal path where the tracking error to the GPS target is minimized, while avoiding the obstacles present on the vehicle's map. The LQT algorithm minimizes the tracking error to the goal point, while simultaneously maximizing the distance to obstacles. It also makes use of a fuzzy logic system to adjust the optimization parameters according to different environmental scenarios.**

I. Introduction

With the steady advancement of computers and software development tools, modern control techniques such as optimal control are becoming much easier to develop, simulate and implement. As a result, greater opportunity exists to exploit these advanced algorithms to tackle control problems that in the past have been approached using less sophisticated classical control techniques.

The problem of GPS navigation with obstacle avoidance for an autonomous mobile robot is addressed in this paper. Several methods of optimal control are applied and compared. The strategy is to compute an optimal control sequence to drive the vehicle toward the target GPS coordinate while avoiding each obstacle on its map. The control sequence is then recomputed periodically through a receding horizon technique.

A linear quadratic tracker (LQT) [1] is investigated, whose cost function parameters and resulting optimal control law are adjusted according to different scenarios to achieve the desired results. Also, a dynamic programming (DP) approach is analyzed, which is found to be better suited for applying several constraints to the control problem, such as when the robot is in the presence of several obstacles without much room between them, or in the presence of very large obstacles. The derivation of the control schemes is presented, and the simulation results are discussed and compared.

II. Control Strategy

Fig.1 shows a block diagram of the optimal control approach. The plant is a non-linear, continuous-time model of a differential drive vehicle. The state space definition for this model is shown in (1). The sensors measure the true state vector $\mathbf{X}$ and produce a discrete, noisy measurement vector $\mathbf{z}_k$. The sensor array consists of a GPS unit to measure x and y, wheel encoders to provide more odometry complementary to the GPS, a digital compass to measure θ with a bias, and a lidar to measure the distance to obstacles.

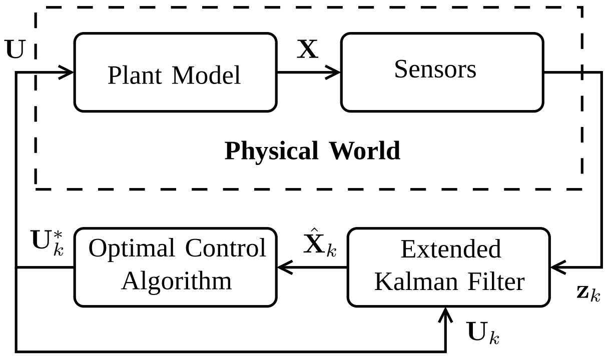


Fig. 1: Block diagram of the optimal control scheme.

An Extended Kalman Filter, using the measurement vector and a discretized and linearized model of (1), performs sensor fusion and generates a full state estimate $\hat{\mathbf{X}}_k$. In addition, it compares the course-over-ground heading of the GPS signal with the compass reading to filter out the bias. The optimal control algorithm, either the LQT (III) or DP (IV), then uses this state estimate to compute an optimal control sequence $\mathbf{U}_k^*$.

III. Linear Quadratic Tracker Approach

A. System Model

Fig.2 shows a top-view of the GPS navigation problem, defining the geometry upon which the system model for the LQT is based, as well as the nomenclature of the various parameters. The state vector $\mathbf{X}$ for the system consists of the vehicle's global position (x and y), its heading angle in the global frame (θ), and the distances to n number of obstacles on the map of the environment ($d_1, d_2, \cdots, d_n$). The coordinates of the obstacles are (x_1, y_1), $(x_2, y_2), \cdots (x_n, y_n)$, and are assumed to be constant. The forward and steering speeds of the vehicle (v_f and v_s, respectively), constitute the control vector $\mathbf{U}$ to the model, and are the signals that are generated by the LQT to drive the robot toward the target point (x_T, y_T).

Using the geometry of Fig.2, the state space model of the system is constructed as shown in (1). For simplicity of notation, the case where there are two obstacles on the map is demonstrated.

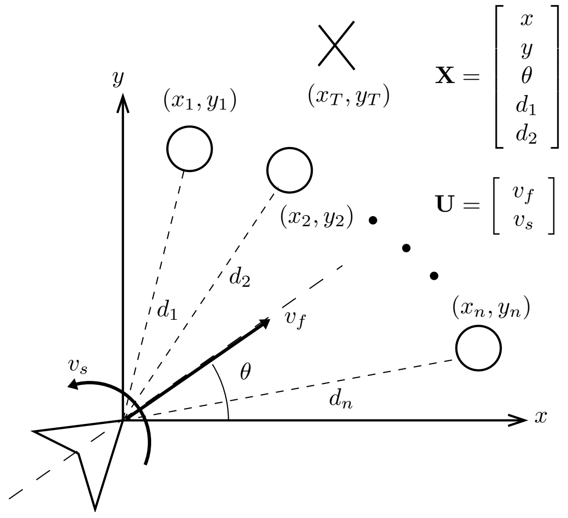

y
(x_1, y_1)
(x_T, y_T)
$\mathbf{X} = \begin{bmatrix} x \\ y \\ \theta \\ d_1 \\ d_2 \end{bmatrix}$
(x_2, y_2)
$\mathbf{U} = \begin{bmatrix} v_f \\ v_s \end{bmatrix}$
d_2
d_1
v_f
(x_n, y_n)
v_s
θ
d_n
x

Fig. 2: Illustration of the LQT control problem

$$
\dot{\mathbf{X}} = f(\mathbf{X}, \mathbf{U}, t)
= \begin{cases} \dot{x} = v_f \cos\theta \\ \dot{y} = v_f \sin\theta \\ \dot{\theta} = v_s \\ \dot{d}_1 = \frac{d}{dt}\left(\sqrt{(x_1 - x)^2 + (y_1 - y)^2}\right) \\ \dot{d}_2 = \frac{d}{dt}\left(\sqrt{(x_2 - x)^2 + (y_2 - y)^2}\right) \end{cases} \tag{1}
$$

After taking the time derivatives and discretizing the system at sample rate T_s, the discrete-time state equations are given in (2). By inspection, it can be seen that the system can be represented in linear state space form $x_{k+1} = Ax_k + Bu_k$, with the B matrix dependent on the current state, as seen in (3).

$$
\mathbf{X}_{k+1} = f(\mathbf{X}_k, \mathbf{U}_k)
= \begin{cases} x_{k+1} = x_k + T_s v_{f_k} \cos\theta_k \\ y_{k+1} = y_k + T_s v_{f_k} \sin\theta_k \\ \theta_{k+1} = \theta_k + T_s v_{s_k} \\ d_{1_{k+1}} = d_{1_k} + T_s d_{1_k}^{-1} v_{f_k} \alpha_1 \\ d_{2_{k+1}} = d_{2_k} + T_s d_{2_k}^{-1} v_{f_k} \alpha_2 \end{cases} \tag{2}
$$

$$
\mathbf{A} = \mathbf{I}_5, \qquad \mathbf{B}_k = T_s \begin{bmatrix} \cos\theta_k & 0 \\ \sin\theta_k & 0 \\ 0 & 1 \\ d_{1_k}^{-1}\alpha_1 & 0 \\ d_{2_k}^{-1}\alpha_2 & 0 \end{bmatrix} \tag{3}
$$

where $\alpha_i = [(x_k - x_i)\cos\theta + (y_k - y_i)\sin\theta]$, with $i = 1, 2, \cdots, n$.

B. Controller Design

The performance index of the controller is a standard LQR performance index, as shown in (4).

$$
J = \frac{1}{2}\mathbf{X'}_N^T \mathbf{P}_N \mathbf{X'}_N + \frac{1}{2}\sum_{k=0}^{N}(\mathbf{X'}_k^T \mathbf{Q} \mathbf{X'}_k + \mathbf{U}_k^T \mathbf{R} \mathbf{U}_k) \tag{4}
$$

To add tracking capability, the system state vector $\mathbf{X}_k$ is augmented with reference signal $\mathbf{r}_k = [x \;\; y]^T$ to form $\mathbf{X'}_k = [\mathbf{X}_k \; \mathbf{r}_k]^T$. Since it is desired to drive the robot toward the fixed target GPS coordinate (x_T, y_T), define $\mathbf{C}^r = \mathbf{I}_2$ and $\mathbf{r}_k = [x_T \; y_T]^T$.

The performance index parameters $\mathbf{P}_N, \mathbf{Q}$ and $\mathbf{R}$ are then defined by

$$
\mathbf{P}_N = \begin{bmatrix} \mathbf{I}_5 & \\ -\mathbf{C}^r & \mathbf{0} \end{bmatrix} \begin{bmatrix} p & 0 & 0 & 0 & 0 \\ 0 & p & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{I}_5 & -\mathbf{C}^r \\ & \mathbf{0} \end{bmatrix}
$$

$$
\mathbf{Q} = \begin{bmatrix} \mathbf{I}_5 & \\ -\mathbf{C}^r & \mathbf{0} \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -q_{d_1} & 0 \\ 0 & 0 & 0 & 0 & -q_{d_2} \end{bmatrix} \begin{bmatrix} \mathbf{I}_5 & -\mathbf{C}^r \\ & \mathbf{0} \end{bmatrix}
$$

$$
\mathbf{R} = \begin{bmatrix} r_{v_f} & 0 \\ 0 & r_{v_s} \end{bmatrix}
$$

where p is a scalar that penalizes the algorithm for being far away from the final x and y position, q_{d_1} and q_{d_2} penalize the algorithm for being close to the obstacles, and r_{v_f} and r_{v_s} penalize the algorithm for using control energy. The assumption that $\mathbf{Q}$ is non-negative definite is thrown out from standard LQR theory, allowing the distance to each obstacle to be *maximized* during the optimization process. Traditional LQR theory expects $\mathbf{Q}$ to be non-negative definite because it assumes the states are all desired to be minimized. As a result, the negative elements in $\mathbf{Q}$ in conjunction with p simulates a repulsive force that guides the vehicle around the obstacles.

The Hamiltonian is then defined by (5), where λ are the Lagrange multipliers that impose the constraints of the system dynamics onto the optimization problem, and $L_k = \mathbf{X}_k^T \mathbf{Q} \mathbf{X}_k + \mathbf{U}_k^T \mathbf{R} \mathbf{U}_k$. The co-state equation and stationarity condition are then given by (6) and (7), respectively.

$$
H_k = L_k + \lambda_{k+1} f \tag{5}
$$

$$
\begin{gathered}
\lambda_k = \frac{\partial H_k}{\partial \mathbf{X}_k} = \frac{\partial L_k}{\partial \mathbf{X}_k} + \frac{\partial f}{\partial \mathbf{X}_k}^T \lambda_{k+1} \\
\rightarrow \lambda_k = \mathbf{Q}\mathbf{X}_k + \frac{\partial f}{\partial \mathbf{X}_k}^T \lambda_{k+1}
\end{gathered} \tag{6}
$$

$$
\begin{gathered}
0 = \frac{\partial H_k}{\partial \mathbf{U}_k} = \frac{\partial L_k}{\partial \mathbf{U}_k} + \frac{\partial f}{\partial \mathbf{U}_k}^T \lambda_{k+1} \\
= \mathbf{R}\mathbf{U}_k + \mathbf{B}_k^T \lambda_{k+1} \rightarrow \mathbf{U}_k = -\mathbf{R}^{-1}\mathbf{B}_k^T \lambda_{k+1}
\end{gathered} \tag{7}
$$

After detailed derivations [1], an algorithm to perform the optimization is developed. After choosing N, the number of steps to look ahead, and specifying $\mathbf{P}_N$, the optimal control sequence is then computed backward through time from $k = N-1$ to $k = 0$ by repeating the computations of (8) – (10).

$$\mathbf{P}_k = \mathbf{Q} + \frac{\partial f}{\partial \mathbf{X}_k}^T \left[\mathbf{P}_{k+1} - \mathbf{P}_{k+1}\mathbf{B}_k\mathbf{G}^{-1}\mathbf{B}_k^T\mathbf{P}_{k+1}\right]\mathbf{A} \quad (8)$$

$$\mathbf{K}_k = \left(\mathbf{B}_k^T\mathbf{P}_{k+1}\mathbf{B}_k + \mathbf{R}\right)^{-1}\mathbf{B}_k^T\mathbf{P}_{k+1}\mathbf{A} \quad (9)$$

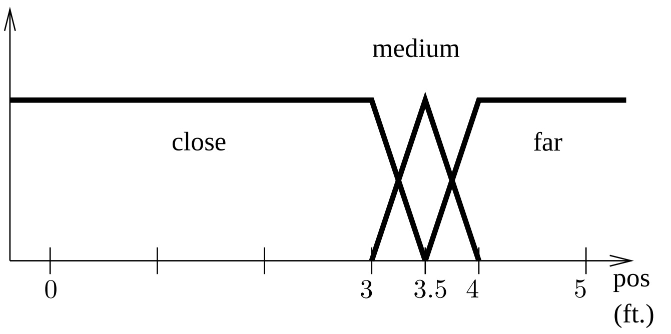
$$\mathbf{U}_k^* = -\mathbf{K}_k\mathbf{X}_k \quad (10)$$

where $\mathbf{G} = \mathbf{B}_k^T\mathbf{P}_{k+1}\mathbf{B}_k + \mathbf{R}$. This algorithm is used as part of a receding horizon strategy, applying only the first N_u control signals to the system before recomputing the optimal control sequence $\mathbf{U}_k^*$. The control window width $1 \leq N_u \leq N$ can be adjusted to influence the performance of the algorithm.

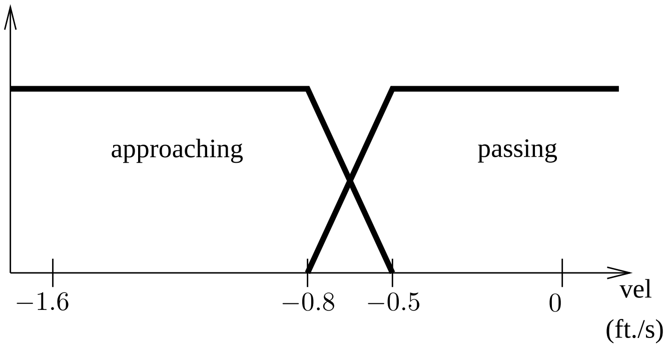
C. Fuzzy Q Matrix Values

To enhance the performance of the LQT algorithm, the negative elements of $\mathbf{Q}$ are adjusted dynamically according to the current control scenario. This is done using a fuzzy logic system, whose inputs are the distance and closing speed to each of the detected obstacles. Trapezoidal input and singleton output memberships are used for their simplicity and fast execution speed on a computer.

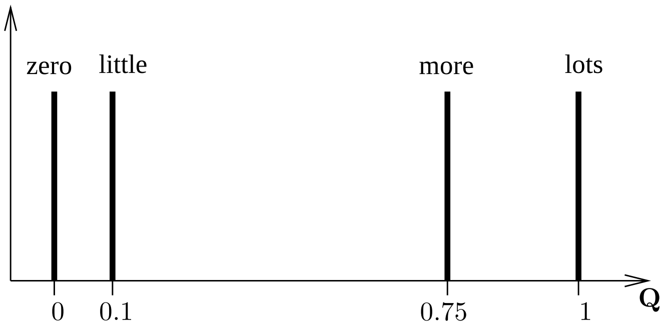
If the vehicle is far away from an obstacle, or if it is passing it, the element of $\mathbf{Q}$ corresponding to this obstacle is set to zero, causing it to not be considered in the optimal path planning algorithm. However, if the obstacle is near and is being approached, then the $\mathbf{Q}$ value should be increased to make the algorithm 'push harder' to avoid it. At mid-range, the obstacle should be reacted to slightly. These qualitative performance criteria are implemented using the fuzzy rules summarized in TABLE I. The definitions of the membership functions are shown in Fig. 3.



(a) Position input membership functions



(b) Velocity input membership functions



(c) Q value output membership functions

Fig. 3: Fuzzy membership functions

IV. Dynamic Programming Approach

A. System Model

The model used for the dynamic programming (DP) algorithm does not include the distances to each obstacle in its state vector. Rather, the model just keeps track of the position and heading of the vehicle, and the optimal control algorithm assigns infinite cost to points that contain obstacles. The discrete-time state model (11) is derived in almost the exact same way as the LQT, following Fig.4.

$$\mathbf{X}_{k+1}^d = \mathbf{A}^d\mathbf{X}_k^d + \mathbf{B}_k^d\mathbf{U}_k \quad (11)$$

TABLE I: Fuzzy Rules to Determine $\mathbf{Q}$ Values

	close	**medium**	**far**
approaching	lots	more	zero
passing	little	zero	zero

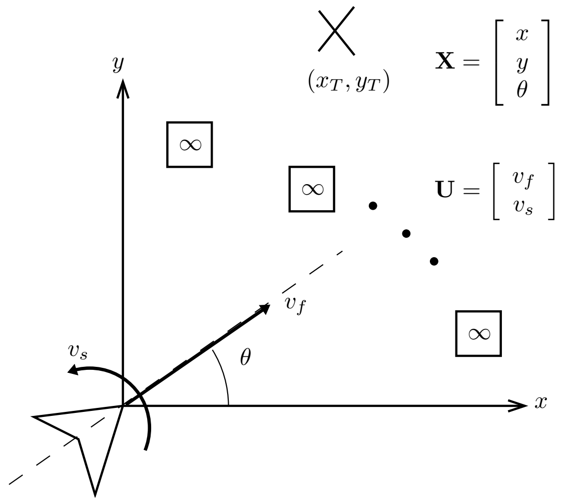
where $\mathbf{X}_k^d = [x \quad y \quad \theta]^T$, and

$$\mathbf{A}^d = \mathbf{I}_3, \qquad \mathbf{B}_k^d = T_s \begin{bmatrix} \cos\theta_k & 0 \\ \sin\theta_k & 0 \\ 0 & 1 \end{bmatrix}$$

B. Controller Design

The cost function for the DP algorithm is shown in (12). The cost function is similar to the LQT performance index, except the reference trajectory is included explicitly, as opposed to augmenting the state vector with the reference signal.

$$J = \left(\mathbf{X}_N^d - \mathbf{r}_N^d\right)^T \mathbf{P}_N^d \left(\mathbf{X}_N^d - \mathbf{r}_N^d\right) + \sum_{k=0}^{N-1}\left(\mathbf{X}_k^{d\,T}\mathbf{Q}^d\mathbf{X}_k^d + \mathbf{U}_k^T\mathbf{R}^d\mathbf{U}_k\right) \quad (12)$$



(x_T, y_T)
$\mathbf{X} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$
$\mathbf{U} = \begin{bmatrix} v_f \\ v_s \end{bmatrix}$
y
x
∞
∞
∞
v_f
v_s
θ

Fig. 4: Illustration of the dynamic programming control problem

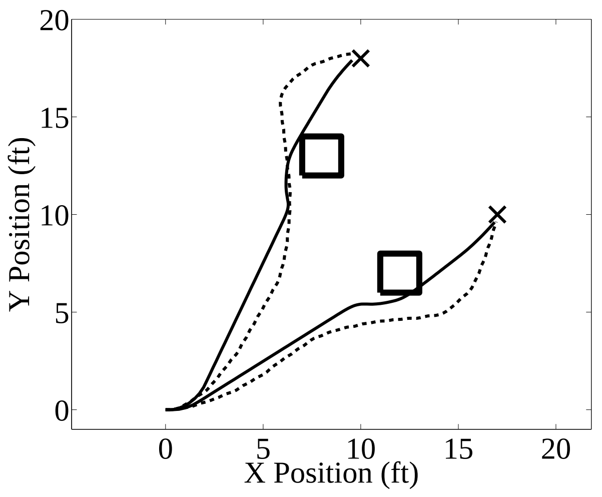


Fig. 5: Optimal trajectory comparison plot. Solid line corresponds to LQT, dashed line to DP.

The cost function parameters are defined as follows:

$$\mathbf{P}_N^d = \begin{bmatrix} p & 0 & 0 \\ 0 & p & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{Q}^d = \begin{bmatrix} q & 0 & 0 \\ 0 & q & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{R}^d = \begin{bmatrix} r_{v_f} & 0 \\ 0 & r_{v_s} \end{bmatrix}$$

The lower right element of $\mathbf{P}^d$ and $\mathbf{Q}^d$ are zero, which makes the heading variable disappear from the cost function, making the optimization problem strictly act upon the current position of the vehicle. The heading angle comes into play in the system dynamics, which the algorithm uses to predict future positions of the vehicle while computing optimal cost paths.

Dynamic programming is based on Bellman's Principle of Optimality [2], wherein all future controls must form an optimal sequence, regardless of the controls that came before. Realizing this, dynamic programming algorithms solve backward from the target point and find the optimal control sequence at every possible state making use of previous minimum cost points.

The dynamic programming algorithm presented here is descibed in the pseudo-code below:

- Initialize cost matrix $\mathbf{J}_N$, explicitly setting costs corresponding to detected obstacle points to ∞.
- Loop k from $N-1$ to 0:
  - Set x and y state.
  - For every possible control input, predict next state $\mathbf{X}_{k+1}^d$. Compute cost of current state $\mathbf{X}_k^d$ and add it to optimal cost from the predicted next state.
  - Record the minimum of these cost values as $\mathbf{J}_k$ and its corresponding control signals as $\mathbf{U}_k^*$.
  - Sweep through the rest of the possible x and y states
- Apply $1 \leq N_u \leq N$ of the computed optimal control signals.
- Recompute.

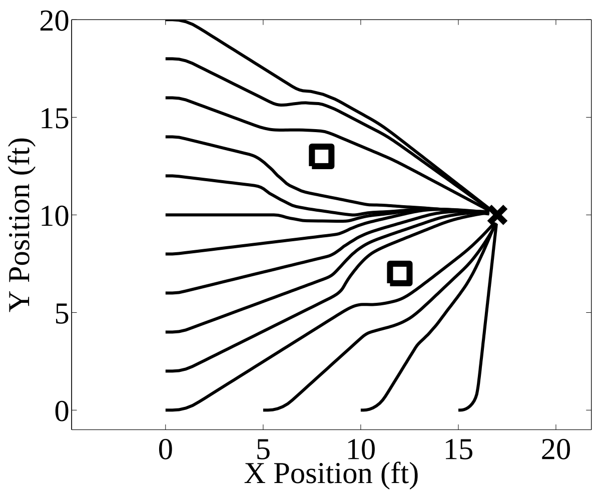
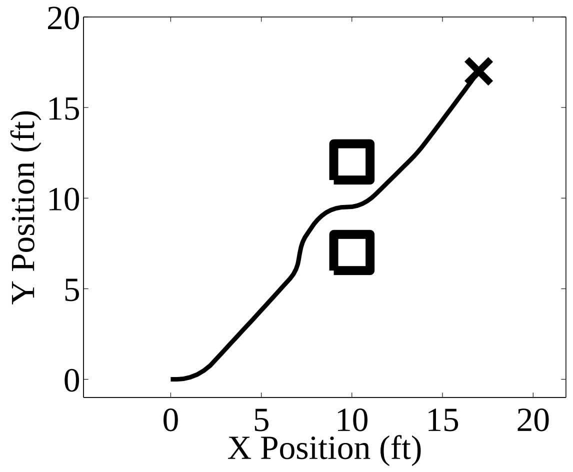


Fig. 6: Plot of optimal trajectories from different starting points, generated by the LQT controller.

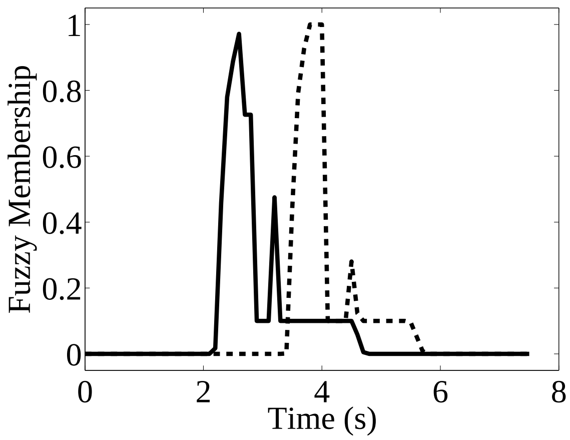
V. Simulation Results

The LQT and DP algorithms were implemented and compared in Matlab. A sample run comparing the two algorithms is shown in Fig.5. In this case, the LQT and the DP algorithms are both using $N = 15$ and $N_u = 3$.

The LQT proves to be more robust and stable because of the added fuzzy logic that adjusts the performance index parameters dynamically. In addition, the LQT requires much fewer computations and is therefore much faster to run on a computer.



(a) Optimal trajectory generated by the LQT system



(b) Corresponding $\mathbf{Q}$ values

Fig. 7: LQT simulation showing how the $\mathbf{Q}$ values change dynamically. The solid line corresponds to the bottom obstacle, the dashed line to the top obstacle.

However, it can be seen from the plot that the LQT tends to shave corners more than the DP algorithm. This is because the LQT only considers the distance to the center of the obstacles and does not take into account the breadth of them, whereas the DP algorithm is explicitly prohibited from going to any point containing an obstacle. Dynamic programming is in general more computationally intensive than other optimal control methods. However, as contstraints are added to the problem, it actually speeds up while other methods like LQT become increasingly complex and computationally intensive.

Therefore, it can be concluded that dynamic programming techniques are ill-suited for wide open navigation situations where there are overwhelmingly many possible routes, but get increasingly better as more and more constraints are added to the optimization problem.

Fig.6 shows trajectories generated by the LQT and fuzzy system from several starting points, and illustrates the robustness of the algorithm. Fig.7 shows how the values of the $\mathbf{Q}$ matrix adjust according to the situation. These sets of runs were performed with $N = 20$ and $N_u = 5$.

VI. Conclusion

In this paper, optimal control techniques were applied to the task of GPS navigation on a mobile robot. The pros and cons of using linear quadratic tracking algorithms versus dynamic programming were observed, and it is concluded that a hybrid system using a combination of the two algorithms could prove fruitful. As the robot traverses more open ground, rely more on the efficient linear quadratic optimizer, but in cramped situations with large or numerous obstacles, adjust more towards dynamic programming.

References

[1] F. L. Lewis and V. L. Syrmos, *Optimal Control.* New York: John Wiley & Sons, 2^{nd} ed., 1995.

[2] R. E. Bellman, *Dynamic Programming.* Princeton University Press, 1957.

[3] R. R. Bitmead, M. Gevers, and V. Wertz, *Adaptive Optimal Control: The Thinking Man's GPC.* Brunswick, Australia: Prentice Hall, 1990.

[4] B. C. Kuo, *Digital Control Systems.* New York: Oxford University Press, 2^{nd} ed., 1992.

[5] Y. Hu, "A-Snake: Integration of path planning with control for mobile robots with dynamic contraints," in *The 2nd International Conference on Computer and Automation Engineering* (anonymous, ed.), pp. 127–134, 2010.

[6] K. C. Cheok, N. Loh, and H. Hu, "Self-training cognitive preview control for autonomous path navigation," in *Proc. 27th IEEE Conf. Decision and Control,* (Austin, TX), pp. 2286–2291, 1988.

[7] K. Hu, "Self-training cognitive preview control for autonomous vehicle path navigation," in *27th Conference on Decision and Control* (anonymous, ed.), pp. 2286–2291, 1988.

[8] N. Crossley, "Discrete-time synergetic optimal control of nonlinear systems," *Journal of Guidance, Control, and Dynamics*, vol. 31, no. 6, pp. 1561–1574, 2008.

[9] S. DeCarlo, "Optimal control of robotic systems with logical constraints: Application to UAV path planning," in *IEEE International Conference on Robotics and Automation* (anonymous, ed.), pp. 176–181, 2008.

[10] M. Lewis, "Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network HJB approach," *Automatica*, vol. 41, no. 5, pp. 779–791, 2004.

[11] Y. Jin, "Trajectory planning for an unmanned ground vehicle group using augmented particle swarm optimization in a dynamic environment," in *IEEE International Conference on Systems, Man, and Cybernetics* (anonymous, ed.), pp. 4341–4346, 2009.

[12] L. Williams, "Optimal, robust predictive control of nonlinear systems under probabilistic uncertainty using particles," in *American Control Conference* (anonymous, ed.), pp. 1759–1761, 2007.

[13] H. Kim, "Nonlinear model predictive controller design with obstacle avoidance for a mobile robot," in *Mechatronic and Embedded Systems and Applications* (anonymous, ed.), pp. 494–499, 2008.