Global Path Planning of Quadrotor Using Reinforcement Learning

Mehdi Khakbaz a,*, Majid Anjidani b

a Department of Electrical Engineering, Sabzevar Branch, Islamic Azad University, Sabzevar, Iran
b Department of Computer Engineering and Information Technology, Payame Noor University, Tehran, Iran

Received 14 May 2022; Accepted 12 July 2022

Abstract

This paper aims to improve the trajectory by an extended reinforcement learning based method in which a new tracking algorithm is used for mobile robot applications with low-rate control command. There are some trajectories that underactuated robots, like quadrotors, are unable to track; hence a suitable trajectory should be designed with respect to the robot's dynamics. In this paper, the initial trajectory is generated by Rapidly-exploring Random Tree Star algorithm which is not suitable for quadrotor application. Then, the initial trajectory is improved by an extension of Path Integral Policy Improvement with Covariance Matrix Adaption (PI2-CMA) algorithm. The extension includes improving tracking algorithm and controller performance considering low-rate control command. Our proposed algorithm succeeded to reduce the cost of tracking by designing safer and shorter trajectories which are more suitable for real robots. Furthermore, the results show that the proposed tracking algorithm and controller improve the performance of tracking. The hardware requirements for implementing our proposed method are a webcam and a personal computer; therefore with a low-cost implementation of the proposed method, a suitable trajectory is designed.

Keywords: Quadrotor robot, Path planning, Trajectory tracking and Reinforcement learning

1. Introduction

During recent years, the quadrotors have been used for a variety of applications; from military applications [1] to commercial usages like toys [2]. The use of quadrotors are even evident in other areas like load transportation [3], inspection of buildings [4], mapping [5], identification and exploration of unknown regions [6], delivery of goods [7] and search and rescue [8]. Firstly, the quadrotors should be able to automatically map the environment and localize their positions by capturing data through their sensors and secondly plan a free obstacle trajectory according to a defined mission, and finally, to track the trajectory. Recently many research has been done in the field of path planning and obstacle avoidance.

The purpose of path planning is to find a trajectory in free obstacle space in a way that the quadrotor is able to track the trajectory. The path planning has two categories: local and global [9].

The Local Path Planning(LPP) tries to identify the obstacles of the environment and get the required knowledge through the sensors. The LPP generates a free obstacle trajectory along the movement [10]. In Global Path Planning(GPP), the environment is static and known so before the robot starts to move the algorithm can generate a complete trajectory between the start point and the destination point. The focus of GPP is to produce the smooth trajectories for the robot [11]. Therefore, another algorithm might be needed for trajectory smoothing and eliminating extra waypoints [12].

In some LPP methods, a combination of mathematical functions is used to represent the smooth trajectory. As an example, one of the most popular LPP methods is Dubin’s curves in which...
discontinuities arise at the junction of line and arc. The discontinuities cause errors while tracking [12]. Two types of curves have been used to resolve the discontinuities: the parametric curves whose curvature is a function of their arc length such as Clothoids, and the curves which coordinates have a closed-form expression such as Bézier curves and B-splines [13].

There are a number of GPP algorithms which are considered as the most common ones. They are namely: Probability RoadMap (PRM) [14], Visibility Graph (VG), the class of Rapidly-exploring Random Tree (RRT) algorithm [15] [16], Space Skeletonization [17] and cell decomposition [18] [19] methods. These methods are considered global when they are applied to the whole configuration space. They can be made local by restricting their application to a subset of configuration space around the current configuration of the robot [19]. In our research, we applied Rapidly-exploring Random Tree Star (RRT*) algorithm as a global path planner.

Also there are some GPP algorithm based on deep reinforcement learning which we explore them in the following:

In [20], a deep reinforcement learning based method is presented for quadrotor autonomous navigation in semi-known environments. Their method uses the dueling double deep recurrent Q-learning to implement global path planning with the obstacle map as input. Also, for effectively conducting real-time autonomous obstacle avoidance with monocular vision, a contrastive learning-based feature extraction is combined with their method.

Another deep Q-network (DQN) based method is proposed in [21] for motion planning of quadrotors. The method generates local motion plans in the form of motion primitives using raw depth images from a front-facing camera. DQN is trained with around 75,000 which takes a raw depth image and relative position information as its input, and yields a motion primitive selection as its output.

In [22], a UAV path planning method based deep reinforcement learning is presented. The double deep Q-networks (DDQNs) in the method are trained by exploiting structured map information of the environment.

Neglecting the robot’s dynamic leads to trajectory incompatibility of the trajectory with the robot and in order to improve the trajectory, a complementary method needs to be applied. As evidence, Bézier curves are used in [23] and [24] and spline curves are applied in [25] and [26] to improve the trajectory. In [27] and [28] a GPP algorithm generates an initial trajectory as input to a closed loop structure including a robot’s dynamic, a controller and an optimization method. The output of the structure is the designed trajectory. A variety of optimization methods such as dynamic programming [29], Quadratic Programming (QP) [30] [31], Genetic Algorithm (GA) [32] [33], artificial network [34], Linear–Quadratic Regulator (LQR) [35], Particle Swarm Optimization (PSO) [33] can be used for trajectory improving. In these methods, the problem must be formulated in a particular form, and the optimal trajectory can be obtained by minimizing the cost function with respect to the desired constraints.

Studies show that various methods have been used by researchers to modify the trajectory. In the following we explore a few of these research:

In [30], an initial trajectory is generated with the RRT algorithm, and a controller based on Model Predictive Control (MPC) is designed to generate the feasible trajectories which satisfy given constraints. It is done by using an inner-simulator to track the trajectories. The MPC problem is transformed into QP form which can be numerically solved.

In [31], an initial trajectory in obstacle-free space is generated by the Informed Optimal Rapidly-exploring Random Trees-star (IRRT*) algorithm and the problem of optimizing the trajectory is stated in a QP form considering dynamic constraints. The constraints, which are not in QP forms cannot be used by this method.

Optimization methods are widely used to adjust the trajectory parameters as mentioned earlier. The environment might slightly change, and a real robot is incapable of adapting to new changes after the optimization. We are going to propose a learning method which can learn a smooth and continuous trajectory considering the robot’s dynamics and can be applied to the real robot. An alternative approach for adjusting the trajectory parameters is Reinforcement Learning (RL) methods. As an example, Path Integral Policy Improvement (PI2) [36] [37] [38] is a RL algorithm with interesting features like having an arbitrary state-dependent cost function part, exploration noise as the only open algorithmic tuning parameter, numerically robust performance in high-dimensional learning problems.
and better performance than gradient-based methods [39]. PI2 has been compared with some of the most efficient methods like REINFORCE [40] [41], GPOMDP [42], PG [43], ENAC [44] and PoWER [45] on multiple applications, and the results show that PI2 surpasses all of them [36]. The Path Integral Policy Improvement with Covariance Matrix Adaptation (PI2-CMA) learning algorithm is a modified version of the PI2 algorithm, which automatically and optimally adjusts the exploration noise. PI2-CMA optimizes the trajectory which is generated by Dynamic Movement Primitives (DMPs). The trajectory involves the position, velocity, and acceleration at any time, and is tracked using a PD controller by default.

In this paper, we propose a modified version of PI2-CMA which is applied for improving the initial trajectory, and we show that our modified version of this algorithm minimizes the cost considering the robot’s dynamics. The cost is calculated during tracking the trajectory by a proposed tracking algorithm which is suitable for mobile robots with low-rate control command. A modified controller, which considers the desired velocity and acceleration, is used in the proposed tracking algorithm. Since our proposed method benefits from the PI2-CMA algorithm, there is no need to state the problem in a specific form meaning that there is no limitation in the definition of the cost function. We have implemented a two-dimensional position measurement system based on image processing and a communication system between quadrotor and PC to evaluate the behavior of the proposed algorithm. In our practical experiment, trajectories are generated by DMPs and tracked by using our proposed tracking algorithm.

The rest of the paper is organized into four sections. The proposed path planning and trajectory tracking are discussed in section 2, and the implementation and results are presented in section 3 and 4, respectively. We conclude in section 5.

2. Quadrotor Path Planning And Trajectory Tracking

The quadrotor task is passing through all waypoints of the trajectory, which is placed in obstacle-free space. We already know that adjusting the task and improving the controller considering the robot's dynamic can reduce the cost. Two steps are needed to accomplish the task: the first step is generating a suitable path for moving from the starting point to the ending point and the second is tracking the trajectory by a strategy. We describe the strategy in the rest of this section.

2-1. Proposed path Planning Algorithm

It is preferred that the robot’s dynamic and the measurement errors be considered during path planning. Short, smooth and continuous trajectories with slow variation in velocity and acceleration, are more suitable to track. Also, it is necessary to keep a safe distance from obstacles to avoid a collision. Therefore, the path planning problem can be formulated as a reinforcement learning problem, in which, the initial trajectory has input role. The lowest cost trajectory which is experimented through learning is the output of the learning problem, in other words, the output is the most suitable trajectory that is experimented by the robot. The cost function is defined according to required features, and more rewards are given to the trajectories with lower cost, in this way the required features are obtained.

In our research, PI2-CMA reinforcement learning algorithm is used to optimize the trajectory. The advantages of using PI2-CMA are: having an arbitrary state-dependent cost function part, automatic adjustment of exploration noise, no open algorithmic tuning parameter, numerically robust performance in high-dimensional learning problems and better performance than gradient-based methods [39]. The PI2-CMA algorithm works based on the reward-weighted averaging methods [46]. It minimizes the cost function through an iterative process of exploration and parameter updating. The policy improvement algorithm [47] is extended for mobile robot applications with low-rate control command, and it is shown in Figure (1). The gray block in Figure (1) denotes our special tracking algorithm and controller in which is named “generate K rollouts” as discussed in the next section. In PI2-CMA, DMPs are usually used to produce the path as a special case of parameterized policies which its attractive feature in this research is to provide instantaneous values of velocity and acceleration of the trajectory. DMP equations are expressed as [39]:

67
\[
\begin{bmatrix}
\frac{1}{\tau} \dot{\eta}_t \\
\frac{1}{\tau} \ddot{x}_{d,t}
\end{bmatrix} = \begin{bmatrix}
-\alpha_x \eta_t \\
\alpha_z (\beta_z (\text{goal}_d - x_{d,t}) - \dot{x}_{d,t})
\end{bmatrix} + \begin{bmatrix}
0 \\
g^T_f
\end{bmatrix} (\theta_d + \epsilon_d)
\]

where \(\eta_t\) is internal state and \(\alpha_x, \beta_z, \tau\) are time constants. \(x_{d,t}, \dot{x}_{d,t}, \ddot{x}_{d,t}\) respectively determine \(d\)th component of the position, velocity, and acceleration at time \(t\). The generated trajectories are sampled over a uniform subdivision of time \(\{t_0, t_1, \ldots, t_N\}\).

The block diagram of our policy improvement algorithm component is shown in Fig. 1. The generated trajectories are sampled over a uniform subdivision of time \(\{t_0, t_1, \ldots, t_N\}\).

The basis functions \(g_k \in R^{n \times 1}\) are defined by a piecewise linear function approximator with Gaussian weighting kernels [36] as follows:

\[
| g_k | = \frac{w_k}{\Sigma_{k+1} w_k} (\text{goal}_d - x_{d,0}),
\]

\[
w_j = \exp \left( -0.5 h_j (\eta - c_j) \right)
\]

Where \(h_j\) is bandwidth and \(c_j\) is the center of the Gaussian kernels.

In Equation (1), the parameter \(\theta_d\) denotes the shape of the \(d\)th dimension of the absorption trajectory in which there is a scalar component in \(\theta^j_d\) for each component of the basis function \(| g_k | = \sum_{i=1}^{n} g^i_k \).

This formulation allows a DMP to generate arbitrary smooth trajectories (see [48] and [49]). The features of DMP include [50]:

1. Each DMP generates a dimension of the trajectory \([x_{d,t}, \dot{x}_{d,t}, \ddot{x}_{d,t}] \in (t_0, \ldots, t_N)\).

Therefore, \(m\) dimensional trajectory can be generated with \(m\) DMPs \([x_{d,t}, \dot{x}_{d,t}, \ddot{x}_{d,t}] \in (t_0, \ldots, t_N)\).

2. Each DMP converges from the starting point \(x_{d,0}\) to the goal point \(\text{goal}_d\).

3. The general shape of the trajectory is determined by the parameters \(\theta_d = [\theta^1_d, \theta^2_d, \ldots, \theta^m_d]\):
\[ S_{k,i} = \sum_{j=i}^{N-1} C_{k,j} \]  
\[ S_{\lambda_{k,i}} = -10 \frac{S_{k,i} - \min_{j=1,...,k} S_{j,i}}{\max_{j=1,...,k} S_{j,i} - \min_{j=1,...,k} S_{j,i}} \]  
\[ P_{k,i} = \frac{\exp(S_{\lambda_{k,i}})}{\sum_{j=1}^{N} \exp(S_{\lambda_{j,i}})} \]  

Where \( C_{k,i} \) is the instant cost of the \( i \)th waypoint of \( k \)th rollout, and it is equal to the sum of all the costs that the robot has spent to reach the \( i \)th point of the trajectory from \((i - 1)\)th point. \( S_{\lambda_{k,i}} \) is the normalized version of \( S_{k,i} \). \( P_{k,i} \) denotes the importance of the \( k \)th rollout from \( i \)th waypoint to \((N - 1)\)th waypoint. Each of the \( K \) rollouts will have a specific contribution in updating the parameters and as well as determining the size and direction of the noise according to the weights. In other words, the new values of \( \theta_d \) and \( \Sigma_d \) for the time \( t_i \) (i.e., \( \theta_d^{new} \) and \( \Sigma_d^{new} \)) are calculated as follow [47]:

\[ \theta_d^{new} = \sum_{k=1}^{K} P_{k,i} \theta_{d,k} \]  
\[ \Sigma_d^{new} = \sum_{k=1}^{K} P_{k,i} (\theta_{d,k} - \theta_d) (\theta_{d,k} - \theta_d)^T, \]  
i.e. \[ \Sigma_d^{new} = \sum_{k=1}^{K} P_{k,i} \varepsilon_{d,k} \varepsilon_{d,k}^T \]  

Then, the parameter \( \theta_d \) and the noise variance \( \Sigma_d \) is updated as [47]:

\[ \theta_d^{new} = \left[ \sum_{i=0}^{N-1} (N - i) \theta_d^{new} \right] \]  
\[ \Sigma_d^{new} = \sum_{i=0}^{N-1} (N - i) \Sigma_d^{new} + \lambda_{\text{min}} I \]  

Where \( I \) is identity matrix and \( \lambda_{\text{min}} \) is a positive scalar constant. In Equation (10), early convergence is avoided by adding \( \lambda_{\text{min}} I \) to the covariance matrix at each updating step [51]. With the updating method, the learning tends towards less costly trajectories. Equation (8) shows, the contribution of sample \( \theta_{d,k} \) in \( \Sigma_d^{new} \) is \( P_{k,i} \varepsilon_{d,k} \varepsilon_{d,k}^T \). In fact, the effect of sample \( \theta_{d,k} \) on the expansion of the distribution in \( \varepsilon_{d,k} \) direction is proportional to \( P_{k,i} \) and \( \varepsilon_{d,k} \). It means that the distribution will be expanded towards less costly samples. This process is repeated until the algorithm converges to a suitable trajectory for moving the robot.

### 2-2. Proposed Path Tracking Algorithm

The proposed path planning algorithm generates a trajectory which includes the waypoints namely \( \text{wayPoint}_{i=0,...,N-1} \). The quadrotor can pass along the trajectory with the sequential chasing of these waypoints. Each waypoint is a vector including three components: instantaneously required position, velocity and acceleration, i.e. \( (p_{t_i}, \dot{p}_{t_i}, \ddot{p}_{t_i}) \), where \( p_{t_i} = (x_1, x_2, x_3) \) shows the position of the robot in three-dimensional space. Trajectory tracking is improved using the desired velocity and acceleration in the controller. Also for every moment, a suitable target point should be determined by the tracking algorithm due to low-rate control command and position updating. In the next part, we present the details of the proposed trajectory tracking algorithm and its controller.

In the trajectory tracking algorithm, the starting waypoint is considered as the first target point \((p_{T_i}, \dot{p}_{T_i}, \ddot{p}_{T_i})\), and the controller brings the quadrotor to the target point neighborhood\(^2\). Then the target point is replaced by the next waypoint, and the controller steers the quadrotor toward it. The process of replacing the target points will be continued until the quadrotor reaches to the end of the trajectory. By this approach, if the quadrotor passes the target point with a distance more than \( r_a \), the target point does not change; therefore reciprocating movement is unavoidable. For this issue, another algorithm is proposed to determine the suitable target points after lift-off and before landing. The pseudocode is given in Table 1. The execution of the algorithm is done after each position and angle measurement. The algorithm starts tracking by selecting \( \text{wayPoint}_0 \) as the target point; therefore the quadrotor moves in the neighborhood of \( \text{wayPoint}_0 \). If the quadrotor stays of successive time steps at the neighborhood of \( \text{wayPoint}_0 \), the target point changes to \( \text{wayPoint}_4 \) (rows 3 to 8 Table 1). In the next time steps, if the quadrotor is in the neighborhood of a target point,
the target point is replaced by the first waypoint outside of its neighborhood. In other words, the waypoints might be dense and close together in the vicinity of the target point; hence some of the next waypoints are also in the neighborhood. In this case, the target point will be replaced by the first waypoint in the outside of the neighborhood (rows 14 to 18 in Table 1). The rows 10 to 13 in Table 1 are considered for reciprocating avoidance during trajectory tracking. In other words, if the quadrotor is getting away from the current target point, which is assumed to be \( \text{wayPoint}_i \), the distances of quadrotor from \( 2n_p + 1 \) waypoints, i.e. \( \text{wayPoint}_{i-n_p}, \ldots, \text{wayPoint}_{i+n_p} \) are calculated. The target point is replaced by the waypoint with minimum distance, \( \text{minDist} \). \( \text{ra} \) is set to \( \text{minDist} + \Delta \) where \( \Delta \) is a small value. In this way, the quadrotor returns to the most appropriate waypoint and continues passing the trajectory. The target point is not changed anymore after setting to \( \text{wayPoint}_{N-1} \), in which the desired velocity and acceleration are zero.

Waypoints contain velocities and accelerations of the trajectory and using the velocities and accelerations can improve the trajectory tracking as

Various controllers such as PID [4] [52], PD [53], PI [52], Takagi-Sugeno fuzzy controller [54] and sliding mode controller [55] are used by researchers to navigate the quadrotor along the trajectory. In this paper, a two degrees of freedom PID controller based on the controller in [53], is used as shown in Equation (10). The advantages of the PID controller are: simple adjustment of its coefficients, robustness and capability of industrial implementation.

\[
\begin{align*}
u &= K_p (p_T - p) + K_i \int (p_T - p) dt \\
&\quad + K_d (b \dot{p}_T - \dot{p}) + c \ddot{p}_T \\
\dot{p} &= \frac{M}{1 + M \frac{T_s}{Z-1}} p
\end{align*}
\]

Where \( u = (u_{x_1}, u_{x_2}) \) is control signal in which \( u_{x_1} \) and \( u_{x_2} \) are control signals for \( p.x_1 \) and \( p.x_2 \) axes. \( p_T, \dot{p}_T \) and \( \ddot{p}_T \) are respectively, the position, velocity, and acceleration of the target point which is determined by the proposed algorithm, \( p \) and \( \dot{p} \) are respectively, the quadrotor position and velocity. \( T_s \) is time step, \( M \) is a constant coefficient that determines the bandwidth of the highpass filter which acts as an differentiator, \( Z \) is \( z \) transfer variable, \( K_p, K_i, K_d, b \) and \( c \) are PID controller coefficients. The first and second term of the controller reduce the position error. The quadrotor velocity converges to the desired velocity due to the third component. The third term, i.e., velocity error is used instead of the derivative of the error signal. The error is equal the difference between the target point and the quadrotor position. Since the target...
point is constant until the quadrotor reaches to its neighborhood, we have \( \dot{p}_T = 0 \) in the error derivative that is not true. With this approach, the velocity is also controlled. Quadrotor velocity is calculated by the time derivative of its position. For this purpose, a high-pass filter in Equation (11) has been used. The filter also reduces the effect of the noise in the computation of the derivative. The term \( \dot{c}p_T \) is the prediction of the desired acceleration, and it can significantly improve performance for trajectories with large accelerations or controllers with soft gains [53].

The yaw angle affects the quadrotor direction according to Equation (12).

\[
\begin{align*}
\dot{u}_\theta &= u_{x_1} \cos \psi + u_{x_2} \sin \psi \\
\dot{u}_\phi &= -u_{x_1} \sin \psi + u_{x_2} \cos \psi
\end{align*}
\]  

(12)

Where \( u_\theta \) and \( u_\phi \) are control signals of pitch angle and roll angle. \( \psi \) is quadrotor yaw angle. To control the yaw angle, PID controller is used as:

\[
\begin{align*}
\dot{u}_\psi &= K_{P3} \times (\psi_T - \psi) + K_{I3} \int (\psi_T - \psi) \, dt \\
&\quad + K_{D3} \frac{Z-1}{Z} (\psi_T - \psi)
\end{align*}
\]  

(13)

Where \( u_\psi \) is yaw angle control signal. \( \psi_T \) is desired yaw angle. \( K_{P3}, \ K_{I3} \) and \( K_{D3} \) are PID controller coefficients. Block diagram of the system controller is shown in Figure (2).

The control signals \( u_\phi, u_\theta \) and \( u_\psi \) respectively determine the desired roll angle(feed to input A of flight controller\(^3\)), the desired pitch(feed to input E) and the desired yaw angle(feed to input R) for the flight controller as Figure (2) shows.

3. Implementation

We did a low-cost implementation of our proposed algorithm and evaluated its performance within a real configuration containing a quadrotor, a webcam, and a computer. Furthermore, a two-dimensional trajectory is considered in an arena that could be seen via the webcam. On the arena, the robot’s attitude \((p, x_1, p, x_2, \psi)\) and obstacles are obtained by processing the webcam frames.

3-1. Specifications of the Robot and the Arena

\[\text{Table 2 shows the specifications of the arena as well as the webcam's view. The color of the floor is white, and the color of the obstacles are set to black. Our webcam, which is a TSCO TW 100K, sends the captured images to a computer via its USB port. The rate of data transmission is 30 frames per second (30 fps). The size of quadrotor with its propeller guards is (approximately) equivalent to a square with a side length of 35cm. Table 2 shows the quadrotor parts list. NAZA M-Lite flight controller has some parameters which called gains and we adjust them as shown in}

\[\text{Table 3. We have made a radio control unit which can communicate with the computer and quadrotor via two modules namely HLK-RM04 and MRF24J40. The radio control unit runs in two modes: either manual or automatic. Running in manual mode allows the operator to control the quadrotor manually, but the automatic mode enables the quadrotor to receive the commands from a computer automatically.}\]
3.2. Our Software Program

Using Visual C++ 2012, we developed a processing unit by which the robot is controlled and the calculations are performed. The processing unit has three modules:

The first module is responsible for performing the following stages:

1- Capturing an image from the webcam.
2- Processing the image using OpenCV library to identify the obstacle positions
3- Constructing the spherical Bounding Volume Hierarchy (BVH) tree, by BVH algorithm to enable checking the collision occurrence in the next stages [56]
4- Generating an initial trajectory [57] by RRT* algorithm.

In the second module, the initial trajectory is represented using the DMPs and improved by the proposed method which is done offline.

The third module:

3.3. Identification of the System

Figure (2) shows the open loop system whose inputs are applied to the flight controller inputs and its outputs are position and angle, \((p.x_1, p.x_2)\) and \(\psi\), of the quadrotor in the arena. The inputs of the flight controller (E,R,A) are 50 Hz PWM signals whose pulse-width is determined as shown in Table 4. The control signal \(u_\theta\), \(u_\phi\) and \(u_\psi\) are pulse-width in the range \([1000, 2000]\) which are transformed to the corresponding PWM pulses in radio controller unit. The radio controller unit transmits the pulses to the flight controller as inputs.

According to Equation (12), position control is simplified as \(u_\theta = u_{x_1}\) and \(u_\phi = u_{x_2}\) by setting
\( \psi_p = 0 \) in the yaw controller. We used Matlab identification toolbox to identify the transfer functions of the open-loop system including position transfer functions, i.e. \( H_{p.x_1} \) and \( H_{p.x_2} \), and yaw transfer function, i.e. \( H_{\psi} \). For this purpose, \( u_\theta \), \( u_\varphi \) and \( u_\psi \) signals are fed to the inputs \( E, A, R \) of the flight controller and \( p.x_1, p.x_2 \) and \( \psi \) are measured as outputs which are shown in Figure (5). \( H_{p.x_1} \), \( H_{p.x_2} \) and \( H_{\psi} \) are identified as:

\[
H_{p.x_1} = \frac{-0.009862Z + 0.011745481275}{Z^2 - 2.0075Z + 1.007509555} \\
H_{p.x_2} = \frac{-0.0012214Z + 0.00270073686}{Z^2 - 2.034 + 1.035011534} \\
H_{\psi} = \frac{0.001539Z}{Z^2 - 1.9Z + 0.9002}
\]

and the response of the identified systems to the inputs are plotted in Figure (5).

**Result** In order to evaluate the performance of our proposed algorithm, two arenas with different obstacles have been created. First, the obstacles regions are specified, and the BVH tree is generated to check the collision occurrence. In our experiment, we use two DMPs for the variables \( p.x_1 \) and \( p.x_2 \) in the two-dimensional arena. The initial parameters \( \theta_{d=1,2}^{init} \) of DMPs are calculated from the initial trajectory.

The PI2-CMA algorithm adjusts the parameter vectors \( \theta_{d=1,2} \) such that the cost function is minimized. We defined the cost function according to the following criteria:

- Unsafe distances from obstacles during trajectory are punished.
- The trajectories with shorter length are preferred.
- Reaching to the goal point is rewarded.

According to the above-mentioned criteria, the cost function for the piece of \( k \)th rollout starting from wayPoint\(_t\) is defined as:

\[
C_{k,i} = \xi_1 \times TLC_{k,i} + \xi_2 \times CUDO_{k,i}, \quad \text{for } i > \frac{N-1}{2}
\]

\[
S_{k,i} = \sum_{j=1}^{N-1} C_{k,j}
\]

Table 4

<table>
<thead>
<tr>
<th>Input</th>
<th>Minimum pulse width</th>
<th>Middle pulse width</th>
<th>Maximum pulse width</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(roll)</td>
<td>1000µs</td>
<td>1500µs</td>
<td>2000µs</td>
</tr>
<tr>
<td>E(pitch)</td>
<td>1000µs</td>
<td>1500µs</td>
<td>2000µs</td>
</tr>
<tr>
<td>R(yaw)</td>
<td>1000µs</td>
<td>1500µs</td>
<td>2000µs</td>
</tr>
</tbody>
</table>

Fig. 5. a) The response of \( H_{p.x_1} \) and the position system \( p.x_1 \) to \( u_\varphi \). b) The response of \( H_{p.x_2} \) and the position system \( p.x_2 \) to \( u_\varphi \). c) The response of \( H_\psi \) and the yaw system to \( u_\varphi \). d) The input signals \( u_\theta, u_\varphi \) and \( u_\psi \).
where $TLC_{k,i}$ is the distance between current and previous robot positions (note that the kth rollout length is equal to $\sum_{j=0}^{N-1} TLC_{k,i}$) and is defined as:

$$TLC_{k,i} = (p.x_1 - p_{prev}.x_1)^2 + (p.x_2 - p_{prev}.x_2)^2$$

where the current and the previous position of the quadrotor are denoted by $p$ and $p_{prev}$. $CRGP_{k,i}$ is the distance between current robot position and the goal point and is defined as:

$$CRGP_{k,i} = (p.x_1 - p_{goal}.x_1)^2 + (p.x_2 - p_{goal}.x_2)^2$$

Where $p_{goal}$ denotes the last waypoint which also called goal point. $CUDO_{k,i}$ denotes the cost of the robot distance from obstacles and is defined as follows:

$$CUDO_{k,i} = \sum_{i=1}^{Q} e^{\frac{0}{\text{mindist} - \text{obsdist}_i}}$$

Where Q is the number of obstacles. Minimum safe distance from the obstacles is $\text{mindist}$. The distance between the quadrotor and the ith obstacle is denoted by $\text{obsdist}_i$, hence the distance $\text{obsdist}_i$ is safe if it is greater than $\text{mindist}$. $\xi_1$, $\xi_2$ and $\xi_3$ are determined according to the importance of each criterion and changing the value of the coefficients affect the result.

The control signal for a target point is applied to the $H_{p,x_1}$ and $H_{p,x_2}$ models and the output, i.e., the quadrotor positions during tracking, are calculated. Hence after tracking, we have all the values which are needed for calculating the cost. Table 5, Table 6, Table 7, Table 8 and Table 9 respectively show the parameter values for DMP, yaw and position controller tracking algorithm and cost function.

Table 6: The values of the position controller coefficients

<table>
<thead>
<tr>
<th>Position controller coefficients</th>
<th>$K_p$</th>
<th>$K_i$</th>
<th>$K_d$</th>
<th>$b$</th>
<th>$c$</th>
<th>$M$</th>
<th>$T_s(s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0.5</td>
<td>0.3</td>
<td>1.2</td>
<td>0.5</td>
<td>0.1</td>
<td>20</td>
<td>0.033</td>
</tr>
</tbody>
</table>

Table 7: The values of the yaw controller coefficients

<table>
<thead>
<tr>
<th>Yaw controller coefficients</th>
<th>$K_{P3}$</th>
<th>$K_{I3}$</th>
<th>$K_{D3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>3</td>
<td>0.3</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 8: The values of the tracking algorithm Parameters

<table>
<thead>
<tr>
<th>Tracking algorithm Parameters</th>
<th>$n_c$</th>
<th>$n_p$</th>
<th>$\Delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 9: The values of the cost function parameters

<table>
<thead>
<tr>
<th>Cost function parameters</th>
<th>$\xi_1$</th>
<th>$\xi_2$</th>
<th>$\xi_3$</th>
<th>$\text{mindist}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0.1</td>
<td>75</td>
<td>10</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure (6) and Figure (7) show initial and improved trajectory of an example leaning run in two different arenas, A and B. Figure (8) and Figure (9) show the average learning curves of 10 different learning runs and illustrate how the cost is reduced during the learning process.

By a real experiment in arena A, we compare the performance of the proposed tracking algorithm with/without reciprocating avoidance code (rows 10 to 13 in Table 1) in Figure (10) and Figure (11). In Figure (10) the improved trajectory in arena A is tracked based on our proposed tracking algorithm three times that the quadrotor movement during one of them is shown in Figure (11).

Figure (12) shows that the algorithm with reciprocating avoidance code has a higher performance. Figure (13) shows the effects of the desired speed and acceleration terms in the controller behavior of the proposed tracking algorithm.

At the beginning of a real experiment, the quadrotor is manually steered into the neighborhood of the starting point and automatic control is activated. During the real experiment, the neighborhood radius, $r_a$, for the first waypoint is set to 10 and for next waypoints, it is set to 15. Also, the yaw angle controller is used to hold $\psi_T = 0$. 

---

Table 5: The values of the DMP Parameters

<table>
<thead>
<tr>
<th>DMP Parameters</th>
<th>$a_x$</th>
<th>$a_y$</th>
<th>$a_z$</th>
<th>$\tau(s)$</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0.333</td>
<td>6.25</td>
<td>25</td>
<td>0.05</td>
<td>25</td>
</tr>
</tbody>
</table>
Fig. 6. Initial and improved trajectory of an example leaning in the arena A.

Fig. 7. Initial and improved trajectory of an example leaning in the arena B.

Fig. 8. Average learning curve during learning in the arena A.

Fig. 9. Average learning curve during learning in the arena B.

Fig. 10. The dotted line denotes the learned trajectory in arena A that is tracked three times by the proposed tracking algorithm in real experiment.

Fig. 11. The quadrotor movement during the tracking experiment which is labeled Test1 in Figure (10).
At the end, we want to design a path by the proposed algorithm for a more complex problem which is presented in paper [22]. The problem in the paper is designing a path which should be passed through the vicinity of some certain points. The designed path in [22] is shown in Figure (14).

To track such a path, the robot have to move slowly in the path corners, otherwise it may not be able to navigate the path successfully. If the dynamics of the robot is taken into account in the design of the path, it will be possible to navigate the path at higher speeds. We designed the same path with our proposed algorithm regarding to the robot dynamics, as shown in Figure (15).

The figure shows that the designed path has passed through some obstacles and during tracking the path, the robot can pass through safe areas at higher speed as shown in the figure.

4. Conclusion

In this paper, a learning-based GPP algorithm is proposed that are applicable to mobile robots. Advantages of using our proposed algorithm are: 1- Using the identified model, the effects of the robot’s dynamic and error/delay in the position measurement system are considered, 2- DMP is used...
to generate desired velocities and accelerations as well as desired positions that improve tracking.

To evaluate the performance of the proposed algorithm, a processing unit is developed for processing images, performing our algorithm calculations and controlling the robot movement. Also, a radio control unit is made for communication between the computer and the quadrotor (or operator in the manual mode).

Results show that the proposed algorithm improves the initial trajectory concerning the robot's dynamics and PI²-CMA with the designed cost function tend to shorter and safer trajectory and improves the initial trajectory concerning the robot's dynamics and PI²-CMA with the designed cost function tend to shorter and safer trajectory. Also, a radio control unit is made for communication between the computer and the quadrotor (or operator in the manual mode).

5. Acknowledgment
The authors would like to thank Omid Maldar, Dr. Moosa Ayati and Dr. Javad Mashayekhi Fard.

References


