Bayesian Optimisation with Prior Reuse for Motion Planning in Robot Soccer

Abhinav Agarwalla*
abhinavagarwalla@iitkgp.ac.in
Indian Institute of Technology
Kharagpur

Arnav Kumar Jain*
arnavkj95@iitkgp.ac.in
Indian Institute of Technology
Kharagpur

KV Manohar
kvmanohar22@iitkgp.ac.in
Indian Institute of Technology
Kharagpur

Arpit Tarang Saxena
arpit.tarang@gmail.com
Indian Institute of Technology
Kharagpur

Jayanta Mukhopadhyay
jay@cse.iitkgp.ac.in
Indian Institute of Technology
Kharagpur

ABSTRACT

We integrate learning and motion planning for soccer playing differential drive robots using Bayesian optimisation. Trajectories generated using end-slope cubic Bézier splines are first optimised globally through Bayesian optimisation for a set of candidate points with obstacles. The optimised trajectories along with robot and obstacle positions and velocities are stored in a database. The closest planning situation is identified from the database using k-Nearest Neighbour approach. It is further optimised online through reuse of prior information from previously optimised trajectory. Our approach reduces computation time of trajectory optimisation considerably. Velocity profiling generates velocities consistent with robot kinodynamoic constraints, and avoids collision and slipping. Extensive testing is done on developed simulator as well as on physical differential drive robots. Our method shows marked improvements in mitigating tracking error, and reducing traversal and computational time over competing techniques under the constraints of performing tasks in real time.

ACM Reference format:
Abhinav Agarwalla*, Arnav Kumar Jain*, KV Manohar, Arpit Tarang Saxena, and Jayanta Mukhopadhyay. 2018. Bayesian Optimisation with Prior Reuse for Motion Planning in Robot Soccer. In *Proceedings of , Goa, India, January 11–13, 2018 (CoDS-COMAD '18),* 10 pages.
DOI: https://doi.org/10.1145/3152494.3152502

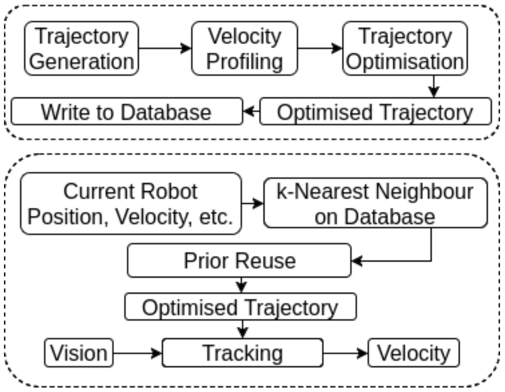

Figure 1: Planning Overview: Initial trajectory generated is optimised using Bayesian optimisation for a combination of various starting and end points, and saved to a database. In the online optimisation, the closest prior is queried using k-Nearest Neighbour algorithm and trajectory is further optimised. Using the optimised trajectory and vision input, tracker calculates and sends robot velocities.

1 INTRODUCTION

The environment of a robot soccer game is highly dynamic and adversarial with fast moving different drive robots and a competing opponent team. High-level complex manoeuvres like attacking and passing are performed through coordination of multiple robots making efficient planning even more challenging. Consider a case where a robot has to intercept a moving ball and shoot it to goal. The robot needs to reach the set interception point at the same time instant as the ball. High deviations from the generated trajectory would lead to missing the ball which is highly undesirable. Also, there is an opponent team whose bots are racing to be the first to reach the ball. This motivates the need for trajectory optimisation to intercept the ball in least possible time. Both trajectory traversal time and trajectory optimisation time contribute to interception time which must be reduced. In this paper, we address this issue and determine time-efficient computation of trajectories for differentiable drive robots under real time constraints.

Previous works in robotics have addressed the problem of trajectory generation and tracking in a dynamic environment. These approaches were built on joining points by lines to avoid obstacles [3, 25], and later extended by interpolating the points using curves to make the path differentiable and smooth at the waypoints [21, 32]. However, the path is not globally optimal with respect to time, and also may not be free from collisions. Recent works like Dynamic Window [35] and Potential Field [14] are more

* denotes equal contribution.

CoDS-COMAD '18, Goa, India
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM. 978-1-4503-6341-9/18/01...$15.00
DOI: https://doi.org/10.1145/3152494.3152502

focused on locally optimal trajectories to avoid collision with obstacles. Particle Swarm Optimization(PSO) [33] have been used to optimise trajectories, but take a lot of time to converge. This makes them unsuitable for real time optimisation such as our use case. We adopt Bayesian optimisation for trajectory optimisation since it is a global optimisation technique requiring very few evaluations of objective function. Fig. 1 gives an overview of our approach.

Another interesting property of Bayesian optimisation is the use of a surrogate function. Surrogate function forms the prior, and can be used to guide optimisation to regions containing minima points. We reuse this prior as obtained by Bayesian optimisation of trajectories and store it in an offline reference database. This reduces computation time since many similar situations arise in robot soccer. We use starting and ending positions of the robots, their velocities and obstacle positions as features for training k-Nearest Neighbour [7] model. The k-Nearest Neighbour technique identifies the closest prior from the reference database, which is reused to reduce optimisation time. We test our approach[1] on simulator developed by us. We also experimented with robots designed and developed by Kharagpur RoboSoccer Students' Group[2] (KRSSG), and report significant improvements in the computation of trajectories as compared to the state of the art techniques. The kinematics of the manufactured robot is depicted in Fig. 2.

Our main contributions summarized:

- We introduce the use of Bayesian optimisation for determining control points leading to time-efficient trajectories that avoid obstacles, and obey kinematic constraints of the differential drive robots.
- We reuse prior information of surrogate model in Bayesian optimisation from previously optimised trajectories to reduce trajectory optimisation time.
- We identify most suitable prior to be reused for a particular situation using k-Nearest Neighbours algorithm [7].
- We extensively test our approach on a simulator as well as on physical soccer playing robots.

2 RELATED WORK

Traditional approaches such as PolarBased [9], MergeSCurve [27] and Dynamic Window [5] [35] have several pitfalls, most important to us being: (1) sub-optimal traversal due to being reactive in nature, (2) no guarantees and estimates of travel time and distance leading to erroneous high level behaviours, (3) no incorporation of kinodynamic constraints of the bot (except Dynamic Window) and (4) frequent slipping of wheels at high speed. Trajectory based methods generally use Bézier curves [13] [31], B-splines [36] or Quintic Bézier Splines [17] for trajectory generation along with building velocity profiles that incorporate kinodynamic constraints. Building on work by Shiller and Gwo [36] and Lau *et al.* [17], we use cubic Bézier splines for trajectory generation. We optimise it globally using Bayesian Optimisation [22].

Potential Field [14] and Visibility Graphs [16] are widely used algorithms for obstacle avoidance. While the former minimizes a potential function with obstacles as repulsive and final goal as attractive poles, the optimisation procedure does not guarantee a globally optimal path. The latter generates paths that are very close to the obstacles frequently leading to collisions. Particle Swarm Optimisation [33, 44] optimises Ferguson cubic spline trajectories, but are slow in real time because of large search space. On the other hand, our approach generates obstacle free optimised trajectories in real time.

Bayesian optimisation has been employed for planning, sensing and exploration tasks in robotics. In [24], [23] and [38], authors propose an active learning algorithm for online robot path planning and reducing uncertainty in its state and environment under a time constraint. They essentially model the trade-off between exploration (uncertainty in environment) and exploitation (optimising planned trajectories) using a Gaussian process. In [20] and [6], authors address the problem of gait optimisation for speed and smoothness on a quadruped robot. In a more machine learning context, [1] proposes a generic method for efficient tuning of new problems using previously optimised tasks. They use a novel Bayesian optimisation technique through combining ranking and optimisation. [12] analyses the effect of hyper-parameters on performance through a functional ANOVA framework imposed on RandomForest [19] predictions. Similar technique is discussed in [4], where authors map features and optimal parameters using an Artificial Neural Network, and then optimise using Covariance Matrix Adaptation Evolution Strategy [11]. We extend these approaches in a kinodynamic planning framework using splines through reuse of prior information from Bayesian optimisation.

Rosman *et al.* [30] formalise Bayesian Policy Reuse (BPR) as solving a task within a limited number of trials by a decision making entity equipped with a library of policies. BPR introduces the problem where multiple policies are learnt for solving a single task in a reinforcement learning context. In contrast, we only store a single policy for different tasks of planning in a robot soccer domain tackling planning. We restrict the more general concept of BPR to policy reuse of Bayesian optimisation to suit our use case.

This paper is organised as follows: Section 3 details trajectory optimisation process using Bayesian optimisation and database formation for prior reuse. Section 4 introduces trajectory generation using splines and velocity profiling approach for kinematic constraints. Section 5 describes the experimental setup of the simulator and kinematics of the manufactured robots. Section 6 compares results of our approach with various other approaches, followed by concluding Section 7.

3 TRAJECTORY OPTIMISATION

Optimised trajectories improve winning chances by increasing ball possession and decreasing trajectory traversal time. However, optimising trajectories in a soccer match severely limits the reaction time of the robots due to computational overheads. We split trajectory optimisation into offline and online computation. First, trajectories are optimised offline and stored in a database. For online optimisation, the database is queried for similar situations to optimise it further. The proposed method is able to optimise trajectories under strict real time constraints.

We employ Bayesian optimisation technique for both offline and online optimisation of positions of control point. Setting control

[1]https://github.com/abhinavagarwalla/motion-simulation
[2]http://www.krssg.in

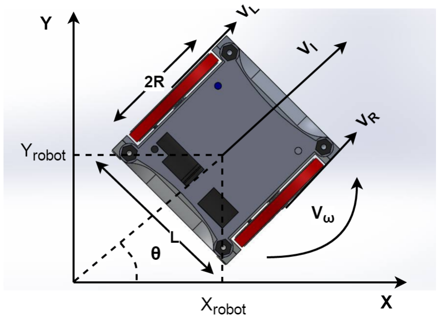

Figure 2: Robot Kinematic Model. (x, y, θ, v_l, v_w) **constitute robot state, which are governed by standard model for differential robots with** $x' = v_l cos(\theta), y' = v_l sin(\theta), \theta' = v_\omega$**.**

points at different locations leads to different trajectories, as described in Section 4. Traversal time for the generated trajectories is taken as the objective to be minimised. Apart from time, avoiding wheel slipping and high curvature paths is also important. These are handled implicitly in the trajectory generation process as described in Section 4. Formally, the objective F is set to minimise traversal time depending on the position of control points of the generated trajectory. Control points are positions through which a trajectory must pass, and can be interpreted as parameters for trajectory generation. These are denoted using $CP_{i,x}$ and $CP_{i,y}$ where i, x and y refer to i^{th} control point and coordinate axis. j denotes the total number of control points. The set of control points is denoted by P, where $P = \{CP_{1,x}, CP_{1,y} \dots CP_{j,x}, CP_{j,y}\}$. P forms the parameter set for optimisation.

$$F(P) = F(\{CP_{1,x}, CP_{1,y} \dots CP_{j,x}, CP_{j,y}\}) = \text{Traversal time of generated trajectory} \tag{1}$$

3.1 Bayesian Optimisation

Since F is a computationally expensive non-linear function for which closed form computation or gradient calculation is not possible, Bayesian optimisation [22] is employed to minimise F.

$$P^* = \arg\min_P F(P) \tag{2}$$

Bayesian optimisation is a surrogate based global optimisation method that tries to optimise with very few function evaluations. As opposed to other optimisation methods, it remembers the history of evaluations i.e. acquired samples on which objective has already been evaluated. Using this history and an acquisition function, it decides the next sampling point P to evaluate F on. It can efficiently handle exploration-exploitation dilemma by not only modelling uncertainty in surrogate model, but also predicting value of objective F at the next sampling point. The next sampling point is selected such that it minimises the uncertainty in the functional space of the objective.

Bayesian optimisation effectively shifts costly evaluations of objective function to cheap evaluations of the surrogate model G. Gaussian Process [28] is utilised as the surrogate model. The advantage of using Gaussian Process as surrogate is that a closed form solution for acquisition function is obtainable since Gaussian distribution is a self-conjugate distribution. Gaussian process models the function $f : P \to F(P)$, through a mean function and a covariance function $k(A, B)$, where A and B denote two parameter sets. We fix mean as a constant m, and use Automatic Relevance Determination (ARD) Matérn 5/2 kernel [37] as the covariance function. ARD Matérn 5/2 kernel, denoted by (3) avoids overly smooth functions, as generated by primitive kernels. σ_s denotes covariance amplitude, and is generally kept as 1. In (4), A_i, B_i and l_i denote ith parameter for set A and B, and characteristic length-scale factor respectively. n denotes the number of parameters, and d captures the distance between two parameter sets.

$$k(A, B) = \sigma_s^2(1 + \sqrt{5d} + \frac{5}{3}d)\exp(-\sqrt{5d}) \tag{3}$$

$$d = \sum_{i=1}^{n} \frac{(A_i - B_i)^2}{l_i^2} \tag{4}$$

Given a history of evaluated points, the next evaluation point is selected as the one which maximises the acquisition function. We use Expected Improvement [37] as acquisition function C, given by

$$C_{EI}(P) = \sigma(P)(u\Phi(P) + \phi(u)) \tag{5}$$

$$u = \frac{F(P_{best}) - \mu(P)}{\sigma(P)} \tag{6}$$

where μ and σ denote the mean and variance of objective function as estimated by Gaussian Process model at P. μ and σ map control point set P to a real number signifying the estimate of the value and variance of the objective function respectively. Φ and ϕ denote the cumulative standard normal function and standard normal function in one dimension respectively. Bayesian optimisation has its own set of hyperparameters, namely length-scaling factors l_i, covariance amplitude σ_s, observation noise, and kernel parameters u and σ. These are optimised by maximising the marginal likelihood of Gaussian process, as discussed in [2].

Table 1: Bayesian Optimisation parameters

Parameter	Value
Number of dimensions: n	$\mid P \mid$
Range	[-1000, 1000]
Objective: $F(P)$	Equation (1)
Acquisition: $C(P)$	Expected Improvement
Kernel: $K(A, B)$	ARD Matern 5/2 kernel
Prior: $P(F)$	$\mathcal{GP}(0, K(\theta = \{l_i \dots, l_n, \sigma_s\}) + \sigma_n^2 I)$

An initial set X of points are sampled using Latin hypercube sampling [40] which generates random numbers from a stratified input distribution. The surrogate model G is incrementally updated to encode the observation history, from which the point maximising the acquisition function is selected as next evaluation point. The process is repeated until a specified number of evaluations or limited computation time. We experiment with various kernel functions, namely Matérn 5/2 kernel, Matérn 3/2 kernel and Squared Exponential kernel [37]. Various acquisition functions such as Expected Improvement [34], A-optimality criteria and Lower Confidence Bound [8]. The chosen design parameters for Bayesian optimisation is summarised in Table 1.

Algorithm 1: Trajectory optimisation and Database formation using Bayesian optimisation

```
Data: Start position $SP(x, y)$, Ending position $EP(x, y)$,
      Starting velocity $SV(x, y)$, Ending velocity $EV(x, y)$,
      Obstacle position $OP(x, y)$, Number of control points $j$
Result: $D$ is the database
1  $x = (SP, EP, SV, EV, OP)$;
2  $P = \{CP_{1,x}, CP_{1,y} \dots CP_{j,x}, CP_{j,y}\}$;
3  $K(P_1, P_2)$: Kernel Function;
4  $C(P)$: Acquisition Function;
5  $N$: Number of iterations;
6  $\{X, y\} = \phi$;
7  while $iter < N$ do
8      $\theta$ = Bayesian optimisation hyperparameters;
9      Posterior Model Update:
         $Pr(F|D) \propto \int Pr(D|F, \theta)Pr(F)Pr(\theta)d\theta$;
10     Select $P_i = \underset{P}{\operatorname{argmax}}\, C(P|Pr(F|D))$;
11     $\{X, y\} = \{X, y\} \cup \{P_i, F(P_i)\}$;
12 end
13 Best $P^* = \underset{P}{\operatorname{argmax}}\, y$;
14 $B^*$ = getTrajectory($P^*$, SP, EP, SV, EV, OP) ; // refer Section 4
15 Update database D with input $x$, $P^*$ and GP Model as prior
```

Algorithm 2: Run-time Bayesian optimisation

```
Data: Start position $SP(x, y)$, Ending position $EP(x, y)$,
      Starting velocity $SV(x, y)$, Ending velocity $EV(x, y)$,
      Obstacle position $OP(x, y)$, Database $D$
Result: $B^*$ is the optimal trajectory
1  $x = (SP, EP, SV, EV, OP)$;
2  for $i$=1 to $m$ do
3      Compute distance $d(D_i, x)$;
4  end
5  Compute set $I$ containing indices for the $k$ smallest distances;
6  Query D to get $k$ priors;
7  Reuse prior for initialising Bayesian optimisation;
8  Run Bayesian optimisation, and get optimised trajectory $B^*$;
9  Return $B^*$;
```

During a robot soccer game, the generated trajectories should be safe i.e. it should have a good distance with obstacles to avoid collisions. In our case, obstacle avoidance is simply achieved through adding a penalty to the objective function if the current trajectory intersects an opponent robot. This results in the optimisation step rejecting the trajectories that collide with the obstacles. The field boundaries are also taken as line obstacles, resulting in rejection of trajectories with any point outside the soccer field.

3.2 Prior Database Creation

In a robot soccer match, trajectories must be generated within a time window of a few hundred milliseconds. Despite being sample efficient, Bayesian optimisation is impractical for real time optimisation. Since we use Gaussian process as the surrogate model, both prior and posterior are Gaussian distributions. To reduce trajectory generation time, we reuse prior from offline optimised trajectories for online optimisation. Assuming that we have enough data points of optimised trajectories, prior is rich with areas of sample points that lead to minimisation of objective function. This results in a surrogate model which has lower uncertainty about the objective function surface. It efficiently trades off exploration and exploitation, leading to reduced function evaluations and trajectory generation time.

We form a database by optimising trajectories through Bayesian optimisation for various combinations of starting and ending positions of our robots and obstacles, summarised in Algorithm 1. We also vary planning scenario with the starting and ending velocity, number of obstacles and number of control points which greatly influence the generated trajectory. Trajectories generated for different planning scenarios are optimised offline using a simulator. For each trajectory, the database stores the optimal placement of control points as well as the prior parameters. Prior parameters are mainly constituted by surrogate model along with its mean function and kernel function. Other parameters such as variance in observation noise, and acquisition function are also utilised. Planning scenario comprising of starting and ending positions, velocities, number of obstacles, number of control points and obstacle positions are also stored. In this study, we simulate a total of $20,000$ random planning scenarios, and save them to the database.

3.3 Prior Reuse

With numerous starting and ending positions of five enemy obstacles and our bots, the space of possible configurations is really huge. This makes storing all situations computationally prohibitive facilitating the need to reuse previously optimised trajectories. Also, offline optimisation is done on a simulator and not on physical robots. A lot of factors such as lighting conditions, wheel mount, playing surface, etc. can alter playing conditions. Thus, online optimisation is necessary to adapt from simulator environment to current playing conditions.

After offline optimisation, we have a database which can be queried for identifying similar planning scenarios. A planning scenario is characterised through starting and ending positions, velocities, number of obstacles and obstacle positions. These characteristics are used as features for training a k-Nearest Neighbour model [7]. We use k-NN since it is a non-parametric technique that doesn't require learning of model parameters. Also, the model is not very sensitive to hyperparameters which eliminates the need to look for optimum hyperparameters.

The k-Nearest Neighbour (k-NN) technique is based on constructing a space partitioning tree on the training dataset. The constructed tree can be searched at testing time to identify data points similar to test query. It uses a distance function to identify the similarity between the test query point and the training dataset. Many distance functions can be utilised for measuring the similarity between two n-dimensional instances A, B. We utilise L1-distance d, as denoted by (7), as the distance function in this study. A_i and

B_i denotes the component of A and B along i^{th} dimension.

$$d(A,B) = \sum_{i=1}^{n} | A_i - B_i | \tag{7}$$

Our method is general in the sense that other sophisticated machine learning models that employ a notion of similarity can also be employed. The model identifies the most similar planning scenario and its corresponding prior using k-NN. The prior comprising of the mean function and kernel function of surrogate model is used to initialise Bayesian optimisation. It then optimises trajectory in very few steps until either convergence is achieved or computation time runs out. Algorithm 2 summarises prior reuse for real time Bayesian optimisation.

4 TRAJECTORY GENERATION

In this section, we explain how a trajectory is generated and traversal time is computed. Given the starting point, ending point and control points, trajectory are parametrized by splines. Velocity profiling algorithm ensures the kinematic constraints of the bot, and profiles velocity vector at each point on the trajectory. Trajectory traversal time is calculated by accumulating time and velocity attained between two successive points. A tracking algorithm ensures that the robot does not deviate from the generated path.

Cubic Bézier Splines [10] are chosen to represent trajectories because of their desirable smoothness properties. Two dimensional paths are obtained by parametrization of path along x and y directions through the parameter u along the curve. Initial trajectory is generated by stitching together n segments of Cubic Bézier Splines along $n-1$ control points. The equation for the Cubic Bézier Splines is given by (8), which follows the continuity and differentiability constraints at the knots in (9), (10) and the boundary conditions in (11). Initial robot velocity which is captured from the overhead camera is incorporated in (12) resulting in formation of end slope cubic splines. B_i denotes i^{th} segment of the spline with P_{i0}, P_{i1}, P_{i2} and P_{i3} as coefficients.

$$B_i(u) = \begin{bmatrix} (1-u)^3 \\ 3(1-u)^2u \\ 3(1-u)u^2 \\ u^3 \end{bmatrix}^T \begin{bmatrix} P_{i0} \\ P_{i1} \\ P_{i2} \\ P_{i3} \end{bmatrix} \tag{8}$$

$$B_i(1) = B_{i+1}(0) \tag{9}$$

$$B_i^{'}(1) = B_{i+1}^{'}(0), B_i^{''}(1) = B_{i+1}^{''}(0) \tag{10}$$

$$B_i(0) = P_{i0}, B_i(1) = P_{i3} \tag{11}$$

$$B_i^{'}(0) = v_l(0), B_i^{'}(1) = v_l(1) \tag{12}$$

Quintic Bézier Splines [17] which have the additional property of curvature continuity at end points, are compared with their cubic counterparts in terms of trajectory traversal time, tracking error and average velocity of the robot. Our method is inherently bidirectional by taking account of the direction of velocity at the time of trajectory generation.

4.1 Arc Length Re-parametrisation

Arc length re-parametrisation is required to compute the parameter u in (7), from the arc length S, which is equivalent to finding point on trajectory at which the arc length equals S. Re-parametrisation is implemented through Bézier splines [43] and inverting Bézier curves [42], of which former is finally chosen because it provides better performance. Equidistant points are sampled from $u \in [0,1]$, and arc length on each of these points is computed. Then, a 1D Cubic Bézier Spline is fitted on these points, approximating u as a function of S. Now, getting the corresponding u for a given arc length, is a simple evaluation of the cubic spline at u using equation (8).

4.2 Velocity Profiling

Velocity profile must be generated incorporating robot dynamics to minimise the error between observed and expected position at any moment of time. While keeping the errors to minimum, velocity at each point is maximised while respecting the kinodynamic constraints. On the trajectory $B(u)$, we fix a set of N_c equidistant planning points p_i along the arc length using arc length re-parametrisation. Planning points are just sampled from the generated trajectory. We have assumed that the acceleration between two planning points p_i and p_{i+1} is constant, which holds true for most trajectories in practice due to selection of sufficiently many points. Each translational velocity v_i corresponds to a planning point p_i. Initial velocity v_0 is set to initial velocity of robot, and final velocity v_{N_c} is set according to game-play. Forward consistency is established by accelerating with the maximum constrained acceleration from point p_i to reach p_{i+1} with maximum velocity. The process is repeated in the reverse direction. This ensures that the robot traverses between any two points obeying the constraints. The total time needed to complete the trajectory is also obtained, once the velocity v_i at each planning point p_i is determined.

Assuming that the acceleration is constant between the two planning points p_i and p_{i+1}, we compute the time taken to reach any of the equidistant planning points using (13), where v_i and v_{i+1} are determined from velocity profiling. Time taken to reach planning points p_i and p_{i+1} is denoted by t_i and t_{i+1} respectively. Δ_s denotes the distance between p_i and p_{i+1}. The time at the end point N_c is given by (14).

$$t_{i+1} = t_i + \frac{2\Delta_s}{v_i + v_{i+1}} \tag{13}$$

$$totalTime = t_{N_c} \tag{14}$$

4.3 Constraints

The kinodynamic constraints limit the maximum velocity v_i that can be set at each planning point p_i. The maximum translational velocity $v_{max|\omega}$ for angular velocity ω at a point p_i is given by (15), where ω_{max} and κ_i denote maximum rotational velocity and curvature at that point, respectively.

$$v_i \in [0, v_{max|\omega}], v_{max|\omega} = \frac{\omega_{max}}{|\kappa_i|} \tag{15}$$

The acceleration constraints at any planning point p_i depends on the velocities of the adjacent points p_{i-1} and p_{i+1}. The acceleration constraints are symmetric for the motion planning to be smooth

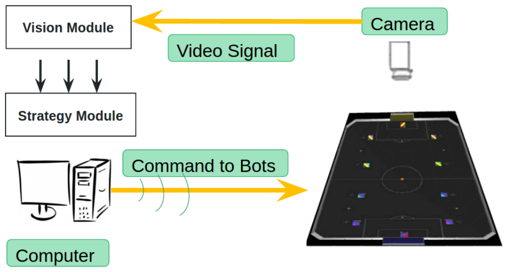

Figure 3: System Setup and Overview: Raw camera feed is used by vision module to identify robot positions and orientations. Strategy module initiates trajectory generation process, after which velocity commands are sent to the robots.

and to avoid slipping of the robot. At any planning point p_i, $v_i \in [v_{min|a_t}, v_{max|a_t}]$ for acceleration a_t can be attained using (16), and (17) where $a_{t_{max}}\Delta_t$ denotes maximum translation acceleration.

$$v_{min|a_t} = v_{i-1} - a_{t_{max}}\Delta_t \tag{16}$$

$$v_{max|a_t} = v_{i-1} + a_{t_{max}}\Delta_t \tag{17}$$

The boundary conditions can now be obtained for translational velocities obeying $a_{t_{max}}$, where distance between any two planning points p_i, p_{i+1} is Δ_s. The reader can refer to [39] for details.

$$v_{min|a_t} = \begin{cases} \sqrt{v_{i-1}^2 - 2a_{t_{max}}\Delta_s} & v_{i-1}^2 \geq 2a_{t_{max}}\Delta_s \\ 0 & else \end{cases} \tag{18}$$

$$v_{max|a_t} = \sqrt{v_{i-1}^2 + 2a_{t_{max}}\Delta_s} \tag{19}$$

4.4 Tracking

The tracking module ensures that the robot follows the trajectory by minimising the error between the current robot state and the expected state at that moment of time. The current tracker is very similar to [15]. Characteristic frequency $\omega_n(t)$ of the system is given by (20) where v_r, ω_r are tangential and angular velocity, respectively. Controller gains k_1, k_2, k_3 are given by (21) where ζ represents damping coefficient, and g is simply a parameter. Errors in position (x, y), orientation θ, and feed forward tangential and angular velocity are denoted by e_1, e_2, e_3, u_{r1} and u_{r2} respectively. The final tangential velocity v and angular velocity w sent to the bot are given by (22) and (23) respectively.

$$\omega_n(t) = \sqrt{\omega_r(t)^2 + gv_r(t)^2} \tag{20}$$

$$k_1 = k_3 = 2\zeta\omega_n(t), k_2 = g|v_r(t)| \tag{21}$$

$$v = u_{r1}cos(e_3) + k_1e_1 \tag{22}$$

$$w = u_{r2} + sgn(u_{r1})k_2e_2 + k_3e_3 \tag{23}$$

5 EXPERIMENTAL SETUP

5.1 Robot Dynamics

Federation of International Robot-Soccer Association (FIRA)[3] organises Mirosot league every year, in which a team of five mobile robots compete against an opponent team to win a soccer match.

[3]https://www.fira.net/

Table 2: A comparison of Bayesian Optimisation with five other algorithms in terms of average time(in sec), tracking error(calculated using (25)) and average velocity(in cm/s) by running them on our robots. The best scores are bold-faced.

Planner	Time(s)	te	Avg. Velocity(cm/s)
S-curve [27]	2.675	6.634	83.338
Polar-Bidirectional [9]	4.190	4.836	42.536
Dynamic Window [35]	2.800	7.067	85.049
PSO [33]	2.331	4.279	94.614
Quintic Spline [17]	2.472	4.631	90.470
Proposed approach	**2.231**	**4.024**	**96.903**

A Mirosot robot is a differential drive robot with two wheels on the sides and supportive caster wheels on the front and back. The robot has zero turn radius, and is fitted a cube of side length 7.5cm. The system setup is described in Fig. 3. Raw values obtained from camera are passed through Kalman [29] filter at 60fps, thereby updating ball and robot state. An NRF module then transmits the velocities to the micro-controller fitted on each robot. There is an internal Fuzzy-PID [41] loop running on the motors, which decides the velocity of motors.

5.2 Physical Constraints

Taking into account wheel dimensions, Fuzzy-PID control loop, circuit design and motors specifications, the maximum loaded velocity attainable by the robot is set to 200 cm/s. For calculating the radial acceleration, the robot is run on a circular path with a fixed radius as in [18]. The value of radial acceleration a_{rad} is increased in steps till the slipping point. Using linear regression, radial acceleration is obtained as a function of radius of curvature r, given by (24).

$$a_{rad} = -5.92r + 700 \tag{24}$$

5.3 Simulator

A simulator is also developed for testing the locomotion algorithms. It is specially designed to enable simulation with kinematic and physical model of our robots. The dimensions of the simulated robots and actuator limits are set as the same as that of actual robots. It discretises the transmission into steps of 16ms, accounts for packet delay and adds noise as encountered while working on real robots. The mobile robot does not receive the sent velocities instantaneously. There is a time lag between transmitting and receiving of packets, which we define as packet delay, and is calculated to be four packets i.e. 64ms. Due to the special care taken in designing the simulator, only minimal retuning of parameters is required and the same code works on both the simulator and the real robots. This has also proved useful in pre-tuning parameters in simulator, and then tuning them on real robots.

6 RESULTS AND DISCUSSION

We analyse the performance of our approach on multiple experiments and compare its performance with other competitive motion planners. In the first set of experiments, we utilise Cubic Bézier Splines [10] for trajectory generation and Bayesian optimisation

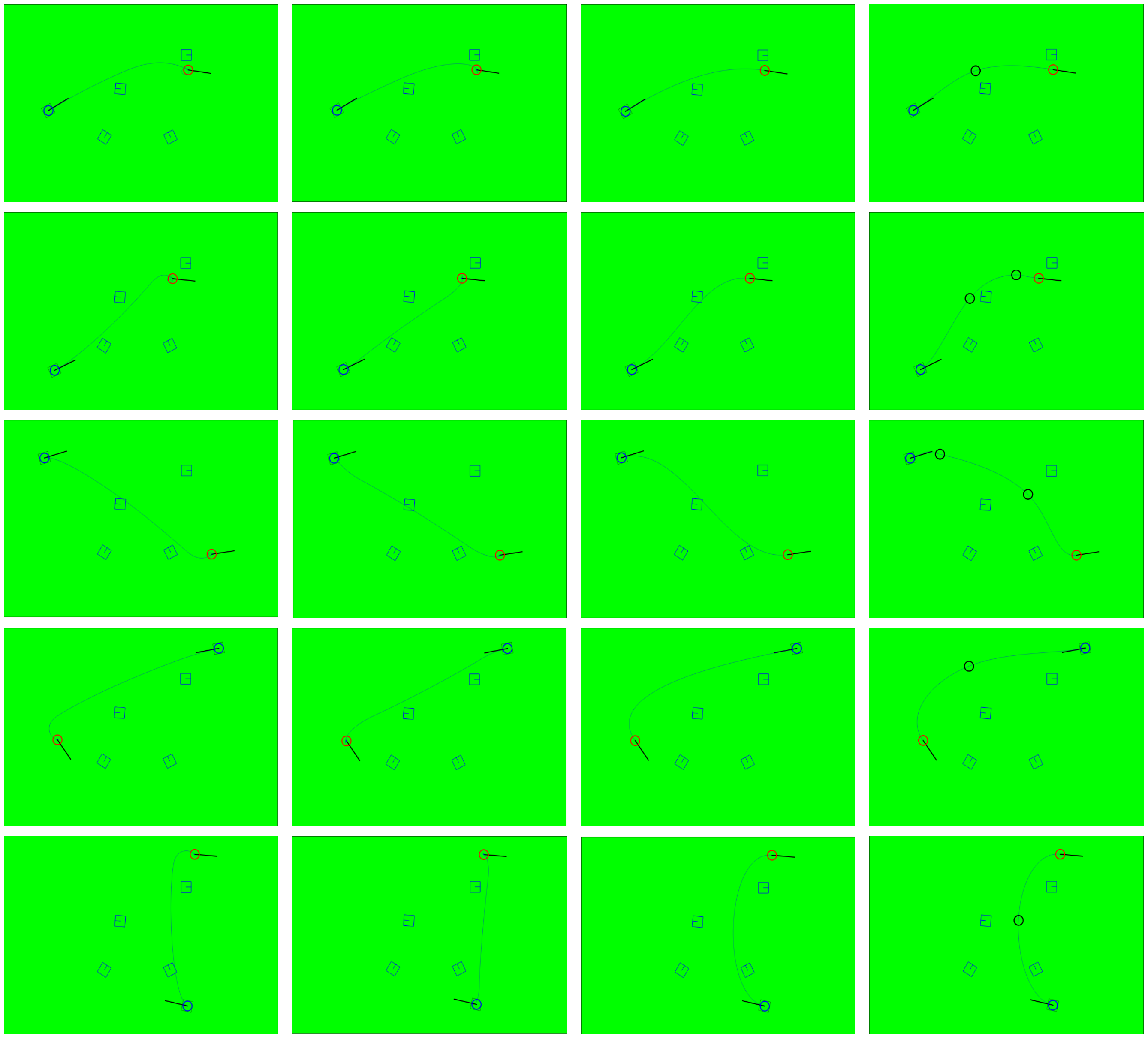

Figure 4: Trajectory generated by a) S-curve, b) Dynamic Window, c) PSO and d) Our Method, for a set of start and end positions on our developed simulator. The blue circle and red circle represent the start and end positions respectively. Black line from the centre of the circles displays the orientation of the robot at those points, and obstacles are shown as blue squares. Trajectory is denoted by a blue curve between start and end points. In case of Our Method, the position of generated control points are shown by black circles.

with prior reuse for trajectory optimisation. Fig. 4 shows the generated trajectory for various planners for different starting and ending points. The field setup is similar to a robot soccer match with over-crowding of obstacles. S-Curve [26] algorithm generates smooth path but the trajectory has very low radius of curvature at some points. Due to kinematic constraints of the robot, the robots slow down at these points leading to increase in total traversal time. The path generated by Dynamic Window [35] has got very sharp turns at the end points which cause slipping. Since the generated trajectory is very close to the obstacles, the robot collides with the obstacles while following the path. The path generated by PSO [33] looks promising but it is computationally expensive with few high curvature points. There are some cases in Fig. 4 where the trajectory is very close to the obstacles. In our experiments, the

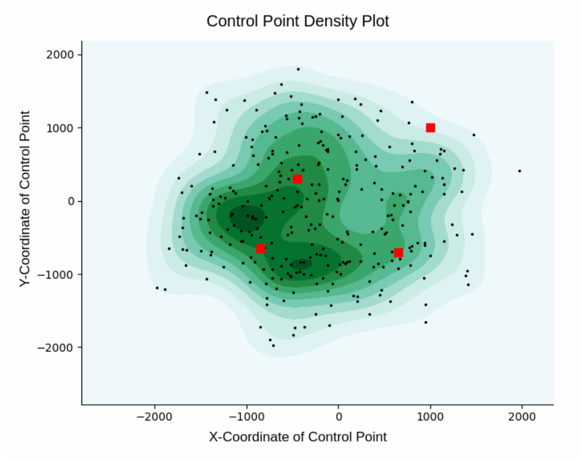

Figure 5: Obtained control points as overlayed on a kernel density estimation plot for 300 randomly initialised starting and ending points. Obstacles are depicted using red squares, and control points using black dots. In density plot, darker regions depict denser regions.

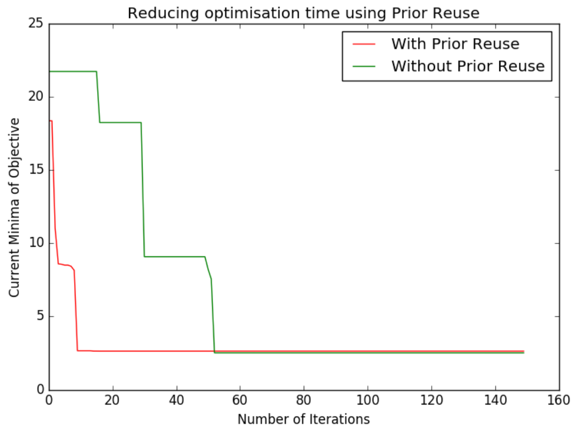

Figure 6: Performance comparison of Bayesian Optimisation with (in red) and without (in green) reuse of prior information from database.

population of PSO contains 15 particles, and takes around 100 iterations to converge. Trajectory generated by splines after Bayesian Optimisation has smooth turns and maintains good distance with obstacles at all points. Thus, our method leads to trajectories that only avoid obstacles, but also reduce trajectory traversal time.

We plan a fixed set of manoeuvres for the robot comprising of a set of 10 starting and ending positions. The task of the robot is to reach from starting point to end point in minimum time possible while avoiding slipping. The path must also result in low tracking errors for high-level attacking and ball interception abilities. Tracking error te is taken as the log of mean squared error given by (25), where $x_{i,p}$, $y_{i,p}$, $x_{i,a}$, $y_{i,a}$ denote the planned and actual positions of the robot for x and y coordinate respectively. n denotes the total length of the trajectory. The tracking error te might become large either due to slipping of wheels or due to collision with another robot.

$$te = \log \frac{\sum_0^1 \sqrt{(x_{i,p} - x_{i,a})^2 + (y_{i,p} - y_{i,a})^2}}{n} \tag{25}$$

Table 2 summarises the performances on the above task. We evaluate planners on average time taken to reach the destination, tracking error te and average velocity of the robot while following the path. Our method performs better than other approaches on all the metrics. Online methods like Dynamic Window [35] and S-curve [26] show promising results in average time taken and average velocity, but have high tracking errors te due to wheel slipping. The tracking error te of Polar-Bidirectional [9] planner is small compared to other online methods, but the velocity of the robot at each point is less leading to high traversal time. The Quintic Spline [17] method results in sharp turns in some cases, which increase the average traversal time of the robot. Our method generates trajectories with high radius of curvature at all points, which helps the robot to move with higher speeds and reduces the risk of slipping. Thus, the average speed is quite higher leading to less traversal time and tracking error te, when compared to other methods.

It was also observed that on planning trajectories using S-curve, Polar-Bidirectional and Dynamic Window, the robot first reaches the end point at a high velocity, over shoots and keep oscillating around the end point. This property can also be seen by zooming into generated trajectories in Fig. 4. Also, these approaches didn't consider the orientation and velocity of the bot at end point, and reached the end point with an arbitrary velocity and orientation. This led to failure in correctly intercepting the ball, and poor ball handling performance.

We evaluate the computation time for online Bayesian optimisation for both cases, one involving prior reuse and other without prior reuse. Fig. 6 shows the number of iterations on the x-axis, and the current best minima of objective function on y-axis. For prior reuse, we utilise the closest prior as obtained from k-Nearest Neighbour to initialise the optimisation. It can be seen that prior reuse greatly reduces the number of iterations required to minimize the objective function. The optimisation process converges in around 15 iterations. Without prior use, it takes around 60 iterations to converge to the minima. Although the minima achieved without reusing prior is slightly better, but is insignificant considering that trajectory optimisation time is limited. This shows that our method is able to generate near optimal trajectories in a limited time setting.

Figure 5 shows the obtained control points for 300 different combinations of starting and ending point with fixed obstacle positions. We see that the obtained control points avoid obstacles for all combinations, as no black point lies on the red square. The control points are concentrated in certain safe regions that are denoted by darker green regions. The generated trajectories not only avoid obstacles by traversing through the region in-between obstacles, but also reduce traversal time.

Table 3: Trajectory traversal time (in seconds) for different combinations of kernel and acquisition function in Bayesian optimisation

Kernel / Acquisition	MaternARD3	MaternARD5	SEARD
EI	1.437	**1.436**	1.452
LCB	1.436	**1.436**	1.458
Aopt	1.449	1.472	1.526

We also experiment with different hyperparameter settings for Bayesian optimisation. We evaluate the performance of different acquisition functions, namely Expected Improvement [34] (EI), Lower Confidence Bound [8] (LCB) and A-optimality criteria (Aopt). We also try different kernel function, namely Matérn 5/2 (MaternARD5), Matérn 3/2 (MaternARD5) and Square Exponential (SEARD) [28]. We use Automatic Relevance Determination (ARD) for all these kernels. The trajectory travel times for all different combinations of kernel function and acquisition function are analysed for 20 different starting and ending points. As shown in Table 3, we see that different combinations lead to very similar results. The best results are obtained using either EI or LCB as acquisition function, along with MaternARD5 or MaternARD3 as kernel function. However, the number of iterations required to reach minima is different for these combinations, as shown in Fig. 7. We see that using MaternARD5 as kernel function, and EI as acquisition function takes the minimum number of iterations.

In the next experiment, we fix the starting and ending positions of our robot and the obstacles. Setting $k = 6$, we query the database using k-NN, and initialise the prior for Bayesian optimisation. The prior is initialised by averaging the prior parameters across all the neighbours. We evaluate the performance of control points obtained after optimisation by measuring the trajectory traversal time for a dense grid surrounding the region. Figure 8 shows the obtained contour plot, where bluer region denote regions minimising trajectory traversal time. We validate the use of k-NN for prior reuse as the control point obtained after optimisation minimises the objective.

7 CONCLUSION

Performing online optimisation under real time constraints is the key limitation of common optimisation methods, especially in complex robotics tasks. We combine learning and planning to address this problem in the context of trajectory optimisation in robot soccer. Our work utilises a database of previously optimised trajectories to effectively reduce optimisation time, and lead to time-efficient trajectories. Specifically, we store resultant prior by running Bayesian optimisation offline for different input configurations. At test time, the closest planning scenario is queried using k-Nearest Neighbours approach for reuse of prior. Bayesian optimisation is initialised with this prior, and run for few function evaluations. We test our method on multiple planning scenarios, where it outperforms traditional optimisation techniques.

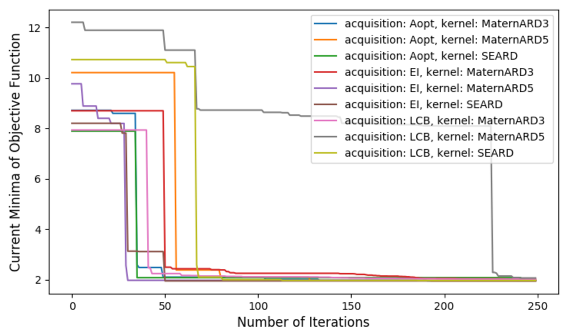

Figure 7: Performance comparison in terms of number of iterations required to converge to minima for different combinations of kernel and acquisition function

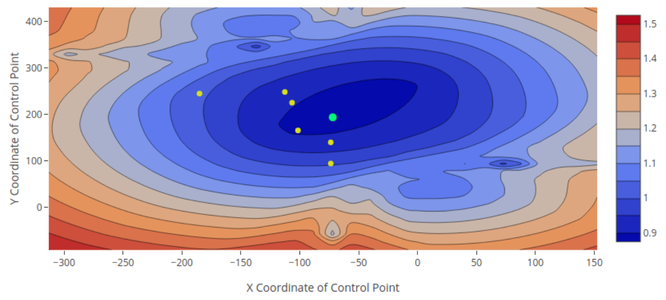

Figure 8: Contour plot for trajectory traversal time (in seconds) generated using control points optimised using prior reuse. The bluer the region, lower the traversal time. Green point denotes control point obtained after Bayesian optimisation. The similar planning scenarios identified k-NN are shown in yellow.

We suggest that current optimisation technique are inadequate for real time optimisation for complicated tasks such as planning. If we are to make significant advances in this field, similar efforts targeting both theory and experimentation for combining learning and planning must be explored. Using more sophisticated machine learning algorithms that can better capture the relation between planning scenario and control point locations is left to future work. Querying multiple prior from the database, and combining them appropriately for a rich prior is also an interesting future direction.

ACKNOWLEDGMENTS

We thank Prof. Sudeshna Sarkar, Prof. Alok Kanti Deb and Prof. Dilip Kumar Pratihar for their constant guidance and support as co-Principal Investigators of Kharagpur RoboSoccer Students' Group (KRSSG). We also thank all the alumni and present members of KRSSG especially Kumar Abhinav, Dhananjay Yadav and Madhav Mantri for addressing game play, mechanical and embedded challenges in robot soccer. This work is supported by Sponsored Research and Industrial Consultancy (SRIC), Indian Institute of Technology, Kharagpur.

REFERENCES

[1] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. 2013. Collaborative hyperparameter tuning.. In *ICML (2)*. 199–207.

[2] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*. 2546–2554.

[3] Johann Borenstein and Yoram Koren. 1991. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation* 7, 3 (1991), 278–288.

[4] Mátyás Brendel and Marc Schoenauer. 2011. Instance-Based Parameter Tuning and Learning for Evolutionary AI Planning. In *PAL 2011 3rd Workshop on Planning and Learning*. 5.

[5] Oliver Brock and Oussama Khatib. 1999. High-speed navigation using the global dynamic window approach. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, Vol. 1. IEEE, 341–346.

[6] Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. 2016. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence* 76, 1-2 (2016), 5–23.

[7] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27.

[8] Dennis D Cox and Susan John. 1992. A statistical method for global optimization. In *Systems, Man and Cybernetics, 1992., IEEE International Conference on*. IEEE, 1241–1246.

[9] Alessandro De Luca, Giuseppe Oriolo, and Marilena Vendittelli. 2001. Control of wheeled mobile robots: An experimental overview. In *Ramsete*. Springer, 181–226.

[10] Gerald Farin. 2014. *Curves and surfaces for computer-aided geometric design: a practical guide*. Elsevier.

[11] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation* 11, 1 (2003), 1–18.

[12] Holger Hoos and Kevin Leyton-Brown. 2014. An efficient approach for assessing hyperparameter importance. (2014).

[13] KG Jolly, R Sreerama Kumar, and R Vijayakumar. 2009. A Bezier curve based path planning in a multi-agent robot soccer system without violating the acceleration limits. *Robotics and Autonomous Systems* 57, 1 (2009), 23–33.

[14] Oussama Khatib. 1986. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*. Springer, 396–404.

[15] Gregor Klančar and Igor Škrjanc. 2007. Tracking-error model-based predictive control for mobile robots in real time. *Robotics and Autonomous Systems* 55, 6 (2007), 460–469.

[16] R Kunigahalli and JS Russell. 1994. Visibility graph approach to detailed path planning in cnc concrete placement. *Automation and robotics in construction XI* (1994).

[17] Boris Lau, Christoph Sprunk, and Wolfram Burgard. 2009. Kinodynamic motion planning for mobile robots using splines. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2427–2433.

[18] Marko Lepetič, Gregor Klančar, Igor Škrjanc, Drago Matko, and Boštjan Potočnik. 2003. Time optimal path planning considering acceleration limits. *Robotics and Autonomous Systems* 45, 3 (2003), 199–210.

[19] Andy Liaw and Matthew Wiener. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.

[20] Daniel J Lizotte, Tao Wang, Michael H Bowling, and Dale Schuurmans. 2007. Automatic Gait Optimization with Gaussian Process Regression.. In *IJCAI*, Vol. 7. 944–949.

[21] Rajko Mahkovic and Tomaž Slivnik. 1997. Smooth Path Planning on the Basis of Reduced Visibility Graph. *MODELLING IDENTIFICATION AND CONTROL* (1997), 267–270.

[22] Ruben Martinez-Cantin. 2014. BayesOpt: a Bayesian optimization library for nonlinear optimization, experimental design and bandits. *Journal of Machine Learning Research* 15, 1 (2014), 3735–3739.

[23] Ruben Martinez-Cantin, Nando de Freitas, Eric Brochu, José Castellanos, and Arnaud Doucet. 2009. A Bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. *Autonomous Robots* 27, 2 (2009), 93–103.

[24] Ruben Martinez-Cantin, Nando de Freitas, Arnaud Doucet, and José A Castellanos. 2007. Active Policy Learning for Robot Planning and Exploration under Uncertainty.. In *Robotics: Science and Systems*. 321–328.

[25] H Meng and PD Picton. 1992. A neural network for collision-free path planning. *Artificial neural networks* 2, 1 (1992), 591–4.

[26] Kim Doang Nguyen, I-Ming Chen, and Teck-Chew Ng. 2007. Planning algorithms for s-curve trajectories. In *2007 IEEE/ASME international conference on advanced intelligent mechatronics*. IEEE, 1–6.

[27] Kim Doang Nguyen, Teck-Chew Ng, and I-Ming Chen. 2008. On algorithms for planning S-curve motion profiles. *International Journal of Advanced Robotic Systems* 5, 1 (2008), 99–106.

[28] Carl Edward Rasmussen. 2006. Gaussian processes for machine learning. (2006).

[29] Maria Isabel Ribeiro. 2004. Kalman and extended kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics* 43 (2004).

[30] Benjamin Rosman, Majd Hawasly, and Subramanian Ramamoorthy. 2016. Bayesian policy reuse. *Machine Learning* 104, 1 (2016), 99–127.

[31] Alireza Sahraei, Mohammad Taghi Manzuri, Mohammad Reza Razvan, Masoud Tajfard, and Saman Khoshbakht. 2007. Real-time trajectory generation for mobile robots. In *Congress of the Italian Association for Artificial Intelligence*. Springer, 459–470.

[32] M Saska, M Kulich, G Klancar, and J Faigl. 2006. Transformed net-collision avoidance algorithm for robotic soccer. In *Proceedings 5th MATHMOD Vienna-5th Vienna Symposium on Mathematical Modelling*.

[33] Martin Saska, Martin Macas, Libor Preucil, and Lenka Lhotska. 2006. Robot path planning using particle swarm optimization of Ferguson splines. In *Emerging Technologies and Factory Automation, 2006. ETFA'06. IEEE Conference on*. IEEE, 833–839.

[34] Matthias Schonlau, William J Welch, and Donald R Jones. 1998. Global versus local search in constrained optimization of computer models. *Lecture Notes-Monograph Series* (1998), 11–25.

[35] Marija Seder and Ivan Petrovic. 2007. Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 1986–1991.

[36] Zvi Shiller and Y-R Gwo. 1991. Dynamic motion planning of autonomous vehicles. *IEEE Transactions on Robotics and Automation* 7, 2 (1991), 241–249.

[37] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.

[38] Jefferson R Souza, Roman Marchant, Lionel Ott, Denis F Wolf, and Fabio Ramos. 2014. Bayesian optimisation for active perception and smooth navigation. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 4081–4087.

[39] Christoph Sprunk. 2008. Planning motion trajectories for mobile robots using splines. *University of Freiburg* (2008).

[40] Michael Stein. 1987. Large sample properties of simulations using Latin hypercube sampling. *Technometrics* 29, 2 (1987), 143–151.

[41] Antonio Visioli. 2001. Tuning of PID controllers with fuzzy logic. *IEE Proceedings-Control Theory and Applications* 148, 1 (2001), 1–8.

[42] Marcelo Walter and Alain Fournier. 1996. Approximate arc length parameterization. In *Proceedings of the 9th Brazilian symposium on computer graphics and image processing*. Citeseer, 143–150.

[43] Hongling Wang, Joseph Kearney, and Kendall Atkinson. 2002. Arc-length parameterized spline curves for real-time simulation. In *5th international conference on Curves and Surfaces*.

[44] Li Wang, Yushu Liu, Hongbin Deng, and Yuanqing Xu. 2006. Obstacle-avoidance path planning for soccer robots using particle swarm optimization. In *Robotics and Biomimetics, 2006. ROBIO'06. IEEE International Conference on*. IEEE, 1233–1238.