Automated Tuning and Configuration of Path Planning Algorithms

Ruben Burger[†], Mukunda Bharatheesha[†], Marc van Eert[*] and Robert Babuška[†]

Abstract— **A large number of novel path planning methods for a wide range of problems have been described in literature over the past few decades. These algorithms can often be configured using a set of parameters that greatly influence their performance. In a typical use case, these parameters are only very slightly tuned or even left untouched. Systematic approaches to tune parameters of path planning algorithms have been largely unexplored. At the same time, there is a rising interest in the planning and robotics communities regarding the real world application of these theoretically developed and simulation-tested planning algorithms. In this work, we propose the use of Sequential Model-based Algorithm Configuration (SMAC) tools to address these concerns. We show that it is possible to improve the performance of a planning algorithm for a specific problem without the need of in-depth knowledge of the algorithm itself. We compare five planners that see a lot of practical usage on three typical industrial pick-and-place tasks to demonstrate the effectiveness of the method.**

I. INTRODUCTION

Recently there has been a growing interest in the use of robotics software. With the increasing popularity of tools like ROS [1], MoveIt! [2], Open Motion Planning Library (OMPL) [8] and OpenRave [3], even hobbyists have now ventured into robotics. These tools have also played a major role in enabling end-users to leverage state-of-the art for real world applications.

Development of novel planning methods with high theoretical merit makes up a large body of research within the motion planning community. Many of these methods are designed to solve a certain set of problems that have some specific characteristic. Consider Transition-based RRT (T-RRT) [4], which combines the exploration strength of RRTs with cost-map methods in order to guide the algorithms to paths that are low cost according to a specified cost metric. T-RRT manages to efficiently solve the posed problem, but how it translates to a different domain is unclear and difficult to estimate without a significant effort. A second example can be found in the set of informed planners like Informed-RRT and BIT* [5][6]. These planners work very well for scenarios in which the obstacles take a certain shape in the configuration space, but it is unclear how they can be leveraged for other classes of problems. They also have a number of configurable parameters that are difficult to understand without significant background knowledge. Another set of planners that use heuristics to guide the search are Kinodynamic Planning by Interior-Exterior Cell Exploration (KPIECE) methods [7]. These planners show good practical performance, but they use a discretization grid that makes it difficult to understand how the planning performance is affected by the configuration of the planning algorithm.

When presented with the problem of selecting a suitable planner for a given real-world problem, one faces a myriad of options. With several different available planning libraries and approaches like OMPL [8], SBPL [9] , CHOMP [10] and STOMP [11], it requires significant knowledge to determine which planning algorithms are suitable for each particular application. Choosing a good configuration for each planning algorithm is even more difficult. A configuration includes both *categorical* (for setting up the planner) and the *numerical* (for tuning) parameters of the planner. The number of parameters ranges for different planning algorithms from just one for RRTConnect [12] to twelve for RRT*, all of which may or may not have a significant impact on the performance. Especially the frequent use of heuristics make it very difficult to predict the behavior of an algorithm as the heuristics interact with each other in unpredictable ways. This makes manual configuration very difficult. In the case of OMPL, some of the parameters are calculated using the characteristics of the environment. While this is indeed beneficial, a larger improvement can be expected with a more rigorous tuning approach.

Typically, end users of motion planning software libraries adhere to ad-hoc heuristics to tune parameters of planning algorithms. The end result of such tuning would indeed result in a satisfactory solution, but there is no indication or feedback to the user if a better solution is possible. This limitation can be addressed with a structured and pragmatic approach to configure the planner. Ideally, it should be possible for an end-user to provide a geometric description of the manipulator and the scene, along with a set of typical problems to let an automated tuning algorithm provide the optimal configuration for each planner. Our work focuses on providing a basic framework to achieve this goal. We focus on the use of motion planning software [8], [2] for robotic manipulators. Despite using a specific class of motion planning software to present our results, our approach itself is generic and is not limited by our choice. The following section provides a brief review of relevant research focusing on parameter tuning for planning algorithms.

II. RELATED RESEARCH

A general approach to algorithm tuning can be found in automatic algorithm configuration methods. These methods are optimization methods that are specifically designed to

[†]Authors are with the Faculty of Mechanical, Maritime and Materials Engineering, Delft University of Technology, 2628 CD Delft, The Netherlands {R.B.Burger,M.Bharatheesha,R.Babuska}@tudelft.nl
[*]Applied Scientist with Technolution BV, 2803 VW Gouda, The Netherlands marc.van.eert@technolution.nl

find the best performing algorithm configuration for a specific problem instance. They often do so by running the algorithm with a certain starting configuration and repeatedly selecting a better configuration based on the previous results. Historically, the problem of algorithm configuration can be abstracted to the Design of Experiments (DOE) approach [17], where a sequence of experiments are conducted to understand the relationship between (physical) experimental parameters and the associated outcomes. Typically, these relationships are established as regression models described by a probability distribution function mostly because, random errors, which are an integral part of physical experiments are well covered by probability distributions of different kinds. Further, related methods such as Design and Analysis of Computer Experiments (DACE) [18] are mentioned in [15] and the references therein. Here, the effort is more focused on computer-based (simulation) experiments which are mostly free of random physical errors. In this sense, methods such as in [18] are much closely related to our work in comparison to the DOE-based methods.

Algorithm configuration can be broadly classified in two main categories, local methods and model-based methods. Examples of methods that use a local search are ParamILS [13] and F-RACE [14]. For Sequential Model Based Optimization (SMBO) methods, a regression model is used to predict the performance of the algorithm in previously unseen regions. Gaussian process (GP) models are a common choice as a non-parametric regression model [15]. One disadvantage of GP models is that they do not deal with categorical input data such as boolean parameters. This is also a drawback with the aforementioned classical methods in [17], [18]. In [16] a method is introduced that uses a different model class for the regression named Sequential Model-based Algorithm Configuration (SMAC). SMAC uses random forests in order to handle categorical parameters. Random forests are collections of regression trees, which are known to offer good performance for categorical input data [16]. Furthermore, SMAC has been shown to deliver promising performance on a diverse set of expensive black-box problems [19].

These methods have been applied to a wide variety of problems, most notably to the tuning of solvers for computationally hard problems. These solvers are typically based on local or tree search, often have many tuning parameters that are very difficult to tune by hand. As an example; the mixed integer programming solver IBM CPLEX has 76 parameters relating to its search strategy [16].

More similar to our proposed application, automatic configuration methods have been applied to classical planning problems. An approach that is similar to our method can be found in [20], where SMAC was used to automatically tune the Fast Downward planning system. In [21], ParamILS was used to tune the topology, transition rules and parameter values of control software for a robot swarm. In [22], constrained Bayesian optimization is proposed as an optimization method that can be used to tune certain target parameters. Using constraints on the optimization, it is ensured that the tested configuration does not lead to safety-critical system failures. In [23] a model-free optimization is used to find optimal PID parameters.

Configuration and tuning of configuration space path planning algorithms in particular seems to be a less highlighted area of research. In [24], an infrastructure that can be used for benchmarking different configurations was introduced, but the problem of how to select competing configurations for each planner is left unexplored.

Typically, many parameters of planning algorithms that we consider in this work are categorical. Therefore, we choose SMAC as algorithm configuration and parameter tuning method.

III. PROBLEM STATEMENT

Let $X \in \mathbb{R}^n$ be the complete configuration space of a robotic manipulator. Denote by $X_{\text{obs}} \subset X$ the obstacle space, which consists of all states of the robot that are invalidated by collisions with either itself or the environment. Let $X_{\text{free}} \subset X$, the complement of X_{obs}, be the set of all valid configurations in the planning space.

Define $X_{\text{start}} \subseteq X_{\text{free}}$ and $X_{\text{goal}} \subseteq X_{\text{free}}$ to be a set of states representative of the planning problem complexity. For example, in a pick-and-place application, X_{start} would consist of pick states, and X_{goal} would be place states. A planning query ψ is then defined by a state pair, consisting of a start state and a goal state:

$$\psi = \{x_{start} \in X_{start}, x_{goal} \in X_{goal}\}$$

The outcome of a planning query is a path which is a sequence of states.

The set of all possible queries derived from X_{start} and X_{goal} is denoted by Ψ. Let c denote a user defined measure that indicates the performance of a planning algorithm on the entire set Ψ. For k start states and m goal states, Ψ will consist of $k \times m$ pairs of states. Denote by P, a target planning algorithm with i configurable parameters. A configuration for P is then denoted by a set of i-tuple:

$$\Phi = \{\phi_0, \dots, \phi_{i-1}\}$$

where ϕ_i is used to denote the parameter value.

Finally, a planner parameter tuning request where planner P, with configuration Φ, is required to solve a planning query ψ such that the resulting path stays in X_{free} can then be written as:

$$P(X, \Phi, \psi)$$

The problem is to find the configuration Φ that maximizes the performance c of planner P on the complete set of planning queries Ψ.

IV. METHOD

To translate the planner configuration problem into a format that SMAC can tune, we specify three ingredients: The algorithm *A*, the performance measure *c* and the problem instance *I*.

SMAC starts by evaluating the default configuration of *A* on *I*, and returns a performance measure after evaluating

c. Subsequently, the current configuration and corresponding performance measure are used to update SMAC's internal model using:

```
SMAC.FitModel()
```

Next, SMAC will select a new configuration to be tested. This operation is denoted by:

```
SMAC.SelectConfigurations()
```

SMAC uses the random forests model to formulate an *Expected Improvement* function and uses a local optimization to find the configuration that maximizes this function.

After the most promising configuration has been selected, it is evaluated on the problem instance I. Whenever a result has a cost lower than the current best performing configuration the corresponding configuration will be stored as the *incumbent* configuration. Finally, after the runtime exceeds the *AllowedRuntime*, the incumbent configuration and the corresponding parameter values are returned.

The main challenge with using SMAC to configure a configuration space planner is how to formulate the problem instance I and set up the performance measure c so that the performance of the resulting configuration also improves the performance over the complete set of problems Ψ. This will be discussed in the following sections.

A. Formulating the problem instance I

In order for SMAC to optimize towards a meaningful configuration, care should be taken in formulating the problem instance I as a function of the configuration space X and the complete problem set Ψ.

The full problem set Ψ was split into a tuning, Ψ_t, and validation set Ψ_v. Due to the random nature of the planners, it is not feasible to use the individual problems of the set as the problem instance. SMAC will not be able to fit an accurate model due to the high variance. By using the full tuning set as the problem instance I, the variance of the resulting cost is minimized. Furthermore, the whole set will be evaluated several times to further decrease the variance. In choosing the number of repetitions, a trade-off has to be made between decreasing the variance of the planning algorithm and allowing SMAC to quickly test new configurations.

B. Formulating the performance measure c

The performance measure c is another key element. The goal of the performance measure is to transform the output of a single problem query into a quantitative measure that can be minimized by SMAC. In many practical scenarios, the aim of tuning is to improve the solving percentage and decreasing the planning time. Focusing on these goals, a very effective performance measure was found to be the average computation time.

In order to limit SMAC spending too much time on poorly performing configurations, a planning timeout is configured for each planner. Whenever this timeout value is reached, a penalty value is used so that the regression model can still learn from the evaluation. The complete method is shown in Algorithm 1.

Algorithm 1 SMAC Tuning

```
    Input : Planner $P$, Tuning set $\Psi_t$ with $N$ queries,
    Configuration space $X$, Iterations $n$,
    default configuration $\Phi_0$.
    Output : Incumbent configuration $\Phi_{inc}$
 1: $\Phi = \Phi_0$
 2: $best = \infty$
 3: while $SMAC.Runtime < AllowedRuntime$ do
 4:     $result = 0$
 5:     for $i = 0, \dots, N-1$ do
 6:         $\psi = \Psi(i)$
 7:         for $0, \dots, n-1$ do
 8:             Timer.start()
 9:             $P(X, \Phi, \psi)$
10:             $result$ += Timer.stop()
11:     if $result < best$ then
12:         $\Phi_{inc} = \Phi$
13:         $best = result$
14:     SMAC.FitModel($result, \Phi$)
15:     $\Phi$ = SMAC.SelectConfigurations()
16: return Incumbent
```

V. RESULTS

In order to validate our tuning method, it was tested on three different environments with two different robots: two environments for the 6-DoF Universal-Robots UR5, and one for the 7-DoF KUKA LBR iiwa 7 R800. The following planners were selected to be tuned:

- **KPIECE** This algorithm uses several heuristics in order to guide the search. These heuristics make it hard to predict its performance and make it an ideal candidate for automated configuration.
- **BKPIECE** For much of the same reasons, the bi-directional variant of KPIECE will also be considered.
- **BIT** While BIT* is actually an optimal planner, it can easily be configured to only calculate the first solution. As this eliminates all optimal properties of BIT*, this planner will be denoted as BIT (without the asterisk). With several configurable parameters, BIT is another interesting candidate for using the SMAC tools for tuning the parameters to a given problem requirement.
- **RRTConnect** As RRTConnect only has a single parameter, it is not expected to gain much from tuning. However, it is worthwhile including RRTConnect as it is a widely adopted planner in more practical applications, known for short computation times and high solving percentages.
- **BiTRRT** Although very similar to RRTConnect, it is interesting to see whether the extra parameters that can be tuned for BiTRRT have a greater impact on its performance.

The experiments were conducted on a PC with a Intel i5-3470 CPU at 3.20GHz and 8GB of RAM running Ubuntu

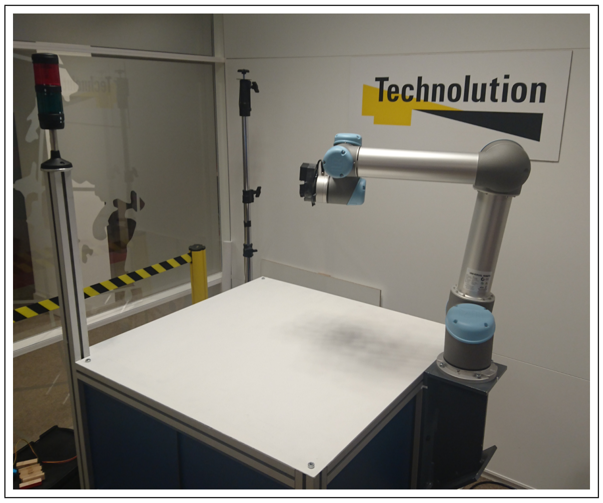

Fig. 1. Photo of the first target problem environment for the UR5 manipulator.

TABLE I

SMAC RESULTS FOR THE FIRST UR5 PROBLEM

Planner	Runtime [ms]	Solved [%]	Path length [2-norm]
RRTConnect	36.3	100	7.3
RRTConnect - SMAC	36.0	100	7.2
BiTRRT	46.8	100	7.1
BiTRRT - SMAC	45.8	99	7.2
BKPIECE	254	99	7.8
BKPIECE - SMAC	108	99	7.4
KPIECE	95.7	100	7.8
KPIECE - SMAC	40.3	100	7.5
BIT	102	92	9.0
BIT - SMAC	52.2	99	8.9

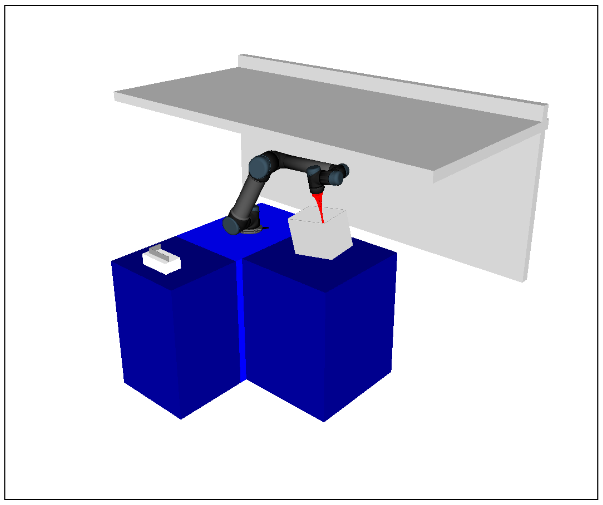

Fig. 2. Model of the second, more complex problem environment for the UR5 manipulator.

TABLE II

SMAC RESULTS FOR THE SECOND UR5 PROBLEM

Planner	Runtime [ms]	Solved [%]	Path length [2-norm]
RRTConnect	165	95	10.9
RRTConnect - SMAC	146	96	10.2
BiTRRT	183	96	10.6
BiTRRT - SMAC	172	96	10.3
BKPIECE	751	45	12.6
BKPIECE - SMAC	450	94	10.6
KPIECE	443	79	11.8
KPIECE - SMAC	337	90	10.6
BIT	286	80	9.7
BIT - SMAC	266	81	9.2

14.04.3. To model the environment ROS Indigo was used in combination with Moveit!. The planners that have been tested are part of the OMPL planning library.

A. *UR5 simple pick-and-place problem*

The first UR5 problem represents a pick-and-place scenario where the robot is to rearrange objects on a table. For these experiments, Ψ_t consisted of 20 problems that were each iterated 5 times for a total of 100 queries. The planning time was selected to be 1 second and SMAC was allowed 30 minutes to find the best configuration. The validation results were obtained by testing on a distinct validation set Ψ_v. Figure 1 shows a photo of the environment of the UR5. With the start close to the surface on the lower corner of the table, and the goal states on the far corner, this can be considered a relatively easy problem with the movement of the UR5 mostly unobstructed. This allowed for SMAC to make several hundred calls to the planning algorithm, especially for faster planners.

Table I shows the averaged total results for the complete validation set. The path length is represented as the average 2-norm in the configuration space of the manipulator and is included to serve as an indicator for the planning algorithms ability to find short paths.

As expected, RRTConnect and BiTRRT have not improved much. For KPIECE, BKPIECE and BIT, the results are more striking. With computation times significantly improved and finding shorter paths, it is safe to conclude that the tuned configuration shows better performance than the default.

For BIT, the percentage of solved problems was increased from 92% to 99%. Closer inspection of the results shows that BIT was struggling with one of the problems in particular, only managing to solve about 40% within the allocated time. After tuning, BIT managed to solve this problem in over 95% of the cases.

B. *UR5 difficult pick-and-place problem*

The second scenario for the UR5 is a more difficult one. Aside from being in a constrained environment, the UR5 is fitted with a 20 cm long vacuum tool. The problem scenario is depicted in Figure 2. With significantly longer planning times, SMAC could do fewer iterations in the allocated time of 30 minutes. Instead of 5 iterations of 20 problems, this scenario was tuned on 4 iterations of 25 problems. The start and goal states were selected as realistic picking and placing states, with the tip of the vacuum tool well inside the stow bin that can be seen on the right.

Consider Table II for the results of tuning on the second scenario. As with the first scenario, the improvement of RRTConnect and BiTRRT is rather small. An impressive tuning result can be seen when considering BKPIECE, whose solving rate was increased from just 45% to 95%.

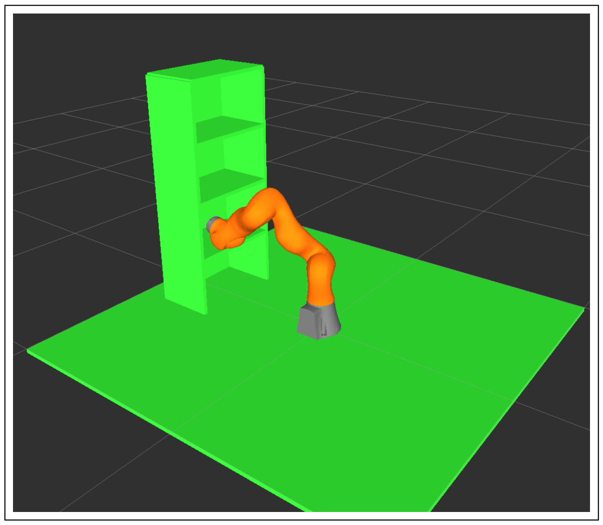
Fig. 3. Model of the environment for the KUKA problem.

TABLE III
SMAC RESULTS FOR THE KUKA PROBLEM

Planner	Runtime [ms]	Solved [%]	Path length [2-norm]
RRTConnect	725	100	6.5
RRTConnect - SMAC	687	100	6.6
BiTRRT	851	99	6.7
BiTRRT - SMAC	807	99	6.9
BKPIECE	1868	17	18.3
BKPIECE - SMAC	1930	18	16.2
KPIECE	1399	68	7.2
KPIECE - SMAC	613	99	6.0
BIT	1239	54	8.5
BIT - SMAC	871	84	6.8

C. KUKA LBR iiwa 7 problem

A problem was designed for the KUKA LBR iiwa 7 manipulator that requires maneuvering to and from different poses inside a cabinet. The problem scenario is shown in Figure 3. With an extra degree of freedom and several start and goal states that are difficult to reach, this presents a more challenging problem than the previous two.

In order for SMAC to be able to test a configuration within reasonable time, the number of iterations were decreased and the planning timeout increased to 2 s. See Table III for the benchmark results after tuning.

Comparing the results of this scene with the UR5 scenes, it is obvious that all planners require more time to find a feasible plan but some (BIT and BKPIECE) seem to suffer more from the extra degree of freedom. The fact that BKPIECE even got *worse* (in terms of runtime) than before tuning, is related to the planning timeout and discussed in Section VI. As with the previous scenarios, a significant improvement can be observed for KPIECE and BIT.

VI. DISCUSSION

A running theme in the previous section that is substantiated with the tabulated results is that automated tuning and configuration is beneficial for all the presented algorithms. In this section, we highlight some of the important aspects from the obtained results. A comparison of planner parameter values before and after SMAC tuning for three planners which showed significant improvement in performance on all three experimental setups is shown in Table IV. Consider for example, the KPIECE planner. We can clearly see that along with the `goal_bias` parameter (which is a common candidate for ad-hoc tuning), the use of SMAC tuning also ensures the rest of the parameters are tuned. As a consequence, these parameters collectively lead to the improved performance seen in the experimental results. However, an interesting analysis would be to also study the impact of each parameter individually and use the corresponding information to formulate better performance measures.

TABLE IV
COMPARISON OF PLANNER PARAMETERS BEFORE AND AFTER TUNING

	Before	After
RRTConnect		
`range`	default‡	3.33
BKPIECE		
`range`	default‡	0.3
`border_fraction`	0.9	0.47
`failed_expansion_score_factor`	0.5	0.24
`min_valid_path_fraction`	0.5	0.31
KPIECE		
`range`	default‡	4.12
`goal_bias`	0.05	0.2
`border_fraction`	0.9	0.96
`failed_expansion_score_factor`	0.5	0.71
`min_valid_path_fraction`	0.5	0.7

‡ *default* indicates the value computed by MoveIt! for a given problem instance.

It is important to note that the performance improvement is not necessarily uniform over all the problems considered. Complex problems typically take longer to be solved and as there is no scaling between problems in the test set, configurations that improve on these more difficult problems are prioritized. With the BIT* algorithm, we have observed a number of problem queries that could not be solved even after tuning. While tuning improved the performance on the other problems of the set, these specific problems remained either very difficult or outright unsolvable to BIT. A SMAC performance measure that puts a bigger penalty on unsolved queries can potentially be used to seek for configurations that do manage to solve all queries, but this is a topic for further research.

In the SMAC manual, it is advised to allow SMAC at least 300 to 400 attempts at finding a good configuration. In addition, the best results are achieved when SMAC is run several times with different starting configurations. However, in the 30 minutes of allowed planning time that was used in the experiments, SMAC often only had time for about 100 configuration tests. As SMAC is often used for algorithms with many more parameters than these planning algorithms, perhaps these requirements can be relaxed somewhat, but further research into the configuration of SMAC and the problem instance is encouraged.

It is pertinent to note the importance of correctly specifying the performance measure and planning timeout. For the KUKA scenario, BKPIECE only managed to solve 17% of the queries when the timeout was set for 2 s. Running SMAC with a 2 s timeout means that for each query that goes over the average even very slightly, a planning time of 2 s is reported. This means that SMAC gathers very little information on the actual performance of a configuration, and is not able to construct a good model. Consider Table V for different tuning results.

TABLE V
BKPIECE TUNING RESULTS WITH A VARIABLE PLANNING TIME
BENCHMARK WAS RUN WITH 2S TIMEOUT

Planner	Runtime [ms]	Solved [%]	Path length [2-norm]
untuned	1868	17	18.3
2 s tuning	1930	18	16.2
4 s tuning	1699	64	8.9

Lastly, the combination of MoveIt! and OMPL has been chosen specifically because of the seamless integration between the two software packages and the relative ease with which the combination allows one to use different planners for a given planning problem. However, we would like to reiterate that our approach itself is not limited to this specific software combination.

VII. CONCLUSIONS AND FUTURE WORK

Our goal was to come up with a tool to enable easy tuning of planning algorithms for a given problem. To this end, we present the SMAC-based tuning method in our work and substantiate the achieved performance improvement in three realistic scenarios using industrial manipulators. Our primary goal with this work is to minimize the amount of background knowledge required to use planning algorithms, particularly in industrial robotic applications. Our long term goal is to further develop this approach in an extensive manner to include any dynamic parameters as well. Eventually, we would like to enable robotics end-users to transition from teaching task specific motions to robots to using planning algorithms to perform task specific motions.

ACKNOWLEDGEMENT

The work leading to these results has received funding from Technolution B.V and from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 609206.

REFERENCES

[1] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System, in ICRA Workshop on Open Source Software, 2009

[2] I. A. Şucan and S. Chitta. MoveIt! [Online]. Available: http://moveit.ros.org

[3] R. Diankov. "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, 2010.

[4] L. Jaillet, J. Corts, and T. Simon, "Sampling-based path planning on configuration-space costmaps," IEEE Transactions on Robotics 26(4), 635-646, 2010

[5] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in 2014 IEEE International Conference on Intelligent Robots and Systems (IROS), 2014, pp. 2997-3004.

[6] J. D. Gammel, S. S. Srinivasa, T. D. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 3067-3074.

[7] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration", Algorithmic Foundation of Robotics VIII. Springer Berlin Heidelberg, 449-464, 2009.

[8] I. A. Şucan, M. Moll and L. E. Kavraki, "The Open Motion Planning Library", IEEE Robotics & Automation Magazine, 19(4):72-82, December, 2012, http://ompl.kavrakilab.org

[9] B. Cohen and M. Likhachev, "The Search Based Planning Library (SBPL)", 2009, [Online] Available: http://www.ros.org/wiki/sbpl

[10] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in IEEE International Conference on Robotics and Automation (ICRA), 2009, pp. 489-494.

[11] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning.", in IEEE International Conference on Robotics and Automation (ICRA), pp. 4569-4574, 2011.

[12] J. Kuffner and S. LaValle, "RRTConnect: An efficient approach to single-query path planning," IEEE International Conference on Robotics and Automation (ICRA), pp. 995-1001 vol.2, 2000.

[13] F. Hutter, H. Hoos, K. Leyton-Brown, and T. Sttzle, "ParamILS: an automatic algorithm configuration framework." Journal of Artificial Intelligence Research, 36(1), 267-306, 2009.

[14] M. Birattari, Z. Yuan, P. Balaprakash, and T. Sttzle, "F-Race and iterated F-Race: An overview.", Experimental methods for the analysis of optimization algorithms, (pp. 311-336), Springer Berlin Heidelberg, 2010.

[15] F. Hutter. "Automated configuration of algorithms for solving hard computational problems." Diss. University of British Columbia, 2009.

[16] F. Hutter, H. Hoos, and K. Leyton-Brown. "Sequential model-based optimization for general algorithm configuration." Learning and Intelligent Optimization, 507-523, Springer Berlin Heidelberg, 2011.

[17] H Robbins. "Some aspects of the sequential design of experiments." Bulletin of the American Mathematical Society 58, 527-535, 1952.

[18] J Sacks, W. Welch, T. Mitchell and H. Wynn. "Design and Analysis of Computer Experiments." Statistical Science 4, 409-423, 1989.

[19] F. Hutter, H. Hoos, and K. Leyton-Brown. "An evaluation of sequential model-based optimization for expensive blackbox functions." Proceedings of the 15th annual conference companion on Genetic and evolutionary computation. ACM, 2013.

[20] J. Seipp, S. Sievers, M. Helmert and F. Hutter, "Automatic Configuration of Sequential Planning Portfolios.", AAAI, pp. 3364-3370, January, 2015.

[21] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni and M. Birattari, "AutoMoDe: A novel approach to the automatic design of control software for robot swarms.", Swarm Intelligence, 8(2), 89-112, 2014.

[22] F. Berkenkamp, A. Krause, and A. Schoellig. "Bayesian Optimization with Safety Constraints: Safe and Automatic Parameter Tuning in Robotics." arXiv preprint arXiv:1602.04450, 2016.

[23] N. Killingsworth, J. Nick, and M. Krstić. "PID tuning using extremum seeking: online, model-free performance optimization." Control Systems, IEEE 26.1, 70-79, 2006.

[24] M. Moll, I. Şucan and L. Kavraki, "Benchmarking Motion Planning Algorithms: An Extensible Infrastructure for Analysis and Visualization.", IEEE Robotics & Automation Magazine, 22(3):96102, September, 2015.

[25] F. Hutter, H. Hoos, K. Leyton-Brown, and K. Murphy, "An experimental investigation of model-based parameter optimisation: SPO and beyond.", Proceedings of the 11th Annual conference on Genetic and evolutionary computation, pp. 271-278, ACM, July 2009.