Journal of Software Engineering for Robotics 1(1), September 2013, 1-10
ISSN: 2035-3928

Reducing the Barrier to Entry of Complex Robotic Software

David COLEMAN[1] Nikolaus CORRELL[1]
[1] Department of Computer Science, University of Colorado at Boulder, 430 UCB, Boulder, Colorado 80309-0430

Abstract—Developing robot-agnostic software frameworks involves synthesizing many disparate fields of software engineering and robotic theory while at the same time accounting for a large variability in hardware designs and control paradigms. As the capabilities and power of robotic software frameworks increases, the complexity and learning curve to new users also increases. If the learning curve for applying and using the software on robots is too high, even the most powerful of frameworks is useless. A growing need exists in robotic software engineering to aid users in getting started with and customizing the provided components as necessary for particular robotic applications. In this paper a case study is presented for some of the best practices found for lowering the barrier of entry of the MoveIt motion planning framework that allows users to 1) quickly get basic motion planning functionality with very little initial setup, 2) automate the configuration and optimization of the framework where possible, and 3) easily customize components of the framework. A graphical interface that assists the user in configuring MoveIt is the cornerstone of our approach, together with a standardized robot model for input, automatically generated configuration files and launch files, as well as a plugin-based architecture for extensibility. These best practices are summarized as a set of barrier to entry design principles. The approaches for lowering the entry barrier are evaluated by a user survey and comparison against our design objects for their effectiveness to the users.

Index Terms—Robotic Software Frameworks, Motion Planning, Barrier to Entry, Setup, Usability, MoveIt

1 INTRODUCTION

MANAGING the increasing complexity of modern robotic software is a difficult engineering challenge faced by roboticists today. The size of the code bases of common open source robotic software frameworks such as ROS [1], MoveIt [2] and OROCOS [3] are swelling [4], and the required breadth of knowledge for understanding the deep stack of software from control drivers to high level planners is become more formidable. As it is beyond the capabilities for any one researcher to have the necessary domain knowledge for every aspect of a robot's tool chain, it is necessary to assist users in the configuration, customization, and optimization of the various software components of a robotic framework.

1.1 Barriers to Entry

The term *barriers to entry* is used here in the context of robotic software engineering to refer to the time, effort, and knowledge that a new user must invest in the integration of a software component to an arbitrary robot. This can include for example creating a virtual model of the robot's geometry and dynamics, customizing configuration files, choosing the fastest algorithmic approach for the application, and finding the best parameters for various algorithms.

Powerful robotics software generally requires many varying degrees of customization and optimization for any particular robot to operate properly. Choosing the right parameters for each utilized algorithm and software component typically involves expert human input using domain-specific knowledge. Many new users to a software package, particularly as robotics becomes more mainstream, will not have the breadth of knowledge to customize every aspect of the tool chain. When the knowledge of a new user is insufficient for the requirements of the software, the barriers to entry become insurmountable and the software unusable. One of the emerging requirements of robot agnostic frameworks is implementing mechanisms that will automatically setup and tune the pipeline for arbitrary robots.

Another motivation for lowering the barrier to entry of complex robotics software is the *paradox of the active user*. This paradox explains a common observation in many user studies that *users never read manuals* but start attempting to use the software immediately [5]. A user's desire to quickly

Regular paper – *Manuscript received April 19, 2009; revised July 11, 2009.*

- *This work was supported by xxxxxxxx (No.xxxxxxxx) (sponsor and financial support acknowledgment goes here).*

accomplish a task results in them skipping reading any provided documentation or gaining deeper understanding of the system and instead diving right into completing their task. The *paradox* is that the user would actually save time in the long run if they learned more about the system before attempting to use it, but from these studies it was that shown that in reality people do not tend to invest time upfront into learning a new system.

TODO: Paragraph on the need for quick start guides? Cite a paper on "hello world" type applications?

Even experts in the area of the associated robotics software will become frustrated if all initial attempts to setup and configure the framework fails and no progress is made. Researchers and engineers today do not have the time or ability to completely understand the entirety of robotics software before they start using it. It is very important for the first experience of the user with a piece of software be positive for the continued use of the software by the user.

1.2 Benefits of Larger User Base

The need to lower the barrier of entry is beneficial to the software itself in that it enables more users to utilize the framework. If the software framework is being sold for profit, the benefits of larger user base is obvious. If instead the software is a free open-source project, as most successful robotic frameworks currently are [4], lowering the barrier to entry is also beneficial in that it creates a larger community of users. As the number of users increases, the speed in which bugs are identified and fixed increases [6]. It is also typically hoped that development contributions to the code base increases, though this correlation is not as strong [6]. Additionally, one of the key strengths of a larger community for an open source project is increased participation of users assisting with quality assurance, documentation, and support [7].

Another benefit of lowering the barrier of entry is it allows the robotics software to additionally become an educational tool for robotics. Not only is the software accessible for academic research and industrial applications, but undergraduate and even primary school students can use it learn some of the higher level concepts of robotic applications as has been demonstrated in [8], [9], [**?**]. This should not be the main focus of most robotics software, but a nice side effect.

1.3 Related Work

There has been much written and developed to addressed the software engineering challenges of complex robotic frameworks, but typically the identified design goals have emphasized the need for properties such as platform independence, scalability, real-time performance, software reuse, and distributed layouts [10], [11], [12]. In [12]'s survey of nine open source robotic development environments, a collection of metrics were used which included documentation and graphical user interfaces (GUI), but no mention was made of setup time, barrier to entry, or automated configuration. CITE MORE RELATED WORK

1.4 Objectives

In this paper we will be using the new MoveIt Motion Planning Framework as our case study for barriers of entry to the usage of robotics software. In section 2 we will discuss the many software components that make motion planning frameworks a good example of the difficulties of complex robotics software. In section 3 we will introduce the MoveIt motion planning framework, in section 5 the methods under taken in MoveIt to reduce the entry barriers will be discussed, and in section 6 we will present the results of these methods on the size of the user base and ease of adoption of the software framework. Will conclude with our experiences in the development process.

2 Motion Planning Frameworks

Robotic motion planning is a maturing and central field in robotics [9] that translates a task into a series of discrete motions such that a robot can move within its environment. The typical use case that will be used as an example in this paper is the problem of controlling a robotic arm from one configuration to another while taking into account various constraints.

The software development of a motion planning framework is challenging and involves combining many disparate fields of robotics and software engineering [13]. We refer to the software as a *framework* in this context because it abstracts the various components of motion planning into generic interfaces as discussed later.

One of the most important features of a framework for motion planning is providing the structures and classes to share common data between the different components. These basic data structures include a model of a robot with its collision bodies, a method for maintaining the state of the robot during planning and execution, and a method for maintaining the environment as perceived by the robot's sensors ("planning scene").

In addition to the common data structures, a motion planning framework requires many different algorithmic components. The motion planning component itself includes one or more algorithms suited for the solving the expected planning problems a robot will encounter. The field of motion planning algorithms is large and no one-size fits all solution exists yet, so a framework that is robot agnostic should likely include an assortment of algorithms.

Other primary components include a collision checking module that detects the intersection of geometric primitives and meshes in the planning scene and robot model. A forward kinematics solver is required to propagate the robot's geometry based on its joint positions, and an inverse kinematics solver is required when planning in the Cartesian space of the end effector. Other potential constraints, such as joint/velocity/torque limits, and stability requirements, require additional components.

Secondary components must also be integrated into a powerful motion planning framework. Depending on what configuration space a problem was solved in, a generated motion planning solution of position way points must be parameterized into a time-variant trajectory to be executed. A controller manager must decide the proper low level controllers for the necessary joints for each trajectory. Finally a perception interface must update the planning scene with recognized objects from a perception pipeline as well as optional raw sensor data.

Higher level applications are then built on top of these motion planning components to coordinate more complex tasks, such as pick and place routines. Other optional components of a motion planning framework can include benchmarking tools, introspection and debug tools, as well as the user-facing GUI.

There already exists a number of motion planning frameworks available today, both open and closed source. A quick survey of these software projects... TODO SURVEY

- ROS Arm Navigation
- OpenRave - Open Robotics Automation Virtual Environment
- OOPSMP - An Object-Oriented Programming System for Motion Planning
- MPK - Motion Planning Kit
- Motion Strategy Library
- Webots
- Custom software

3 MoveIt Motion Planning Framework

MoveIt is the primary motion planning framework in ROS and has been successfully integrated with many robots including the PR2 [14], Robotnaut [15], Baxter, and Atlas. It is written entirely in C++ but includes Python bindings for higher level scripting. It follows the principle of software reuse as advocated for robotics in [4] of not tying itself exclusively to one robotic framework - in our case ROS - by creating a formal separation between core functionality and the framework-dependent aspects.

By default the MoveIt Motion Planning Framework using the Open Motion Planning Library (OMPL) [16] for planning, the Fast Collision Library (FCL) [17] for collision checking, and the OROCOS Kinematics and Dynamics Library (KDL) [18] for forward and inverse kinematics.

MoveIt is fast and can generate complete pick and place plans for a typical robotic arm in BENCHMARK TIME in preliminary experiments on an 64 bit Ubuntu 12.04 PC with a 3.6Ghz Intal Core i7 processor and 16 GB ram.

THINK OF MORE THINGS TO SAY

Choosing the best combination of planning components and parameters for MoveIt is challenging because of the number of choices that need to be made. For example, picking which class and variant of planning algorithm to use for any particular robot and problem is a daunting task even for experts [19].

4 Entry Barrier Design Principles

In designing the configuration process required to allow MoveIt to work with many different types of robots using almost any combination of planning components, several contending design principles for lowering the barrier of entry emerged. These requirements were drawn partially from standard HCI principles, from work on previous robotic software, and from an iterative design process where feedback was solicited from the target users. We believe these *entry barrier design principles* transcend motion planning and can be applied to most robotic software:

Immediate: The amount of time required to accomplish the most primitive task expected from the robotic software component should be minimized. This is similar to the time-honored ”Hello World” demo frequently used by programming languages and typical Quick Start guides in documentation. Immediacy is essential for the *paradox of the active user* as it provides cursory feedback to the user that the software works

Transparent: The configuration steps being performed automatically for the user, and the underlying mechanisms utilized in the software components, should be as visible as possible. Transparency is important so that the user can later understand what is specific to their robot and know how to customize the aspects they desire. Hide too much of the setup process and the user will be hindered in the future.

Intuitive: The need to read accompanied documentation, and the amount of required documentation, should be minimized. A well designed user interface, be it graphical or command line, should be as intuitive as possible by following standard design patterns and providing interface context clues. An ideal GUI for configuration would not require any documentation.

Reconfigurable: The automatically generated parameters and default values for the initial setup of a robot should be easy to modify at a later time by the user. Typically these parameters and values are chosen to work for the largest number of robots possible, but are not optimal for any particular robot. Providing easy methods to reconfigure the initial setup is very important.

Extensible: The ability of the user to customize as many components and behaviors as possible within the reasonable scope of the software. Providing the means to extend the software with custom solutions for a particular application will make the software far more powerful and re-usable. A typical solution for this is providing a plugin interface.

Documented: The written text explaining how to use the software should be maximized for as many aspects and user levels as possible. Even the most intuitive software requires documentation for various aspects of the operation or modification of the software itself. Different types of documentation are needed for different users - for example developers and end users - though in robotics this is frequently the same. Documentation is arguably the most important factor in reducing the barrier to entry of new software.

Many of these principles have opposing objectives that require a trade off or balance be found between them. For example, the desire for transparency in the underlying mechanisms often leads to slower setup times (immediacy) and more complicated configuration steps (unintuitive). The need for extensibility of various components in the software often results in far more complicated software design as more abstraction is required, resulting in a less intuitive code base and difficult documentation. Nevertheless, compromises can be made between these principles that result in a superior user experience as will be demonstrated in the next section.

5 Methods to Lower The Entry Barrier

One of the unique features of MoveIt is the ratio of its power and features to the required setup time. A beginner to motion planning can with very little effort use a model of a robot and execute motion plans in a virtual environment. With a few extra steps of setting up the correct hardware interfaces, one can then execute the motion plans on actual robotic hardware.

The *entry barrier design principles* discussed above were applied to the MoveIt motion planning framework to address the challenges faced for new users to this complex software framework. Developing these solutions required difficult software challenges be overcome as discussed in the following.

5.1 Basic Motion Planning Out of the Box

To address the entry barrier design principle of *immediacy*, a streamlined ”Quick Start” for MoveIt was created that consists of a series of fairly trivial steps relative to our target users. The most challenging of these steps - creating a *robot model* - is not directly related to the configuration of MoveIt, but rather is a prerequisite of using the software framework. Nevertheless, we will discuss this important prerequisite before preceding to the more directly-related configuration steps.

Robot Model Format: The robot model is the data structures and accompanying file format used to describe the three dimensional geometric representation of the robot, its kinematics, as well as other properties relevant to robotics. These other properties can include the geometric visualization meshes, courser-grained collision geometry of the robot used in fast collision checking, joint limits, sensors, and dynamic properties such as mass, moments of inertia, and velocity limits. In our application, as well as most state of the art motion planning frameworks, we will restrict our definition of modeled robots to arbitrarily articulated rigid bodies.

Extensible robotics software requires using a standardized format that can express the intricacies of varying hardware

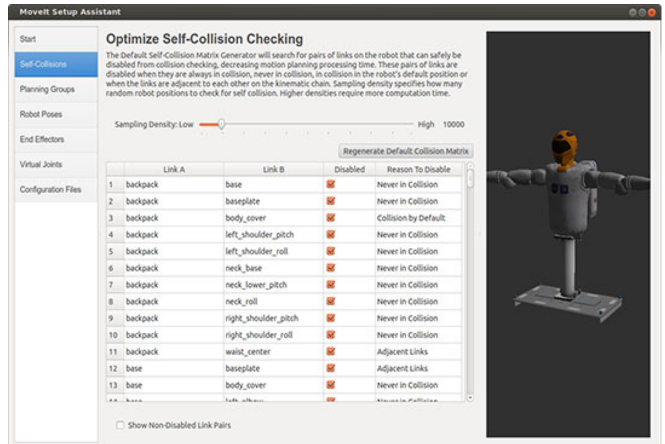

Fig. 1. Screenshot of MoveIt Setup Assistant with the Robonaut loaded TODO: better picture

configurations. An additional design requirement for this standardized format is that it is *intuitive* for users to setup. This was accomplished for MoveIt by utilizing the human-readable XML schema provided by ROS for robot models - the Universal Robotic Description Format (URDF) [20]. This data structure format can accept URDF-formatted files, as well as the industry standard Collada [21] format.

To create the URDF or Collada model for a robot, computer aided design software is used to create the 3D geometries of the robot hardware. The generated individual geometric models, often referred to as *links* in robotics, must then be combined into a connected model using *joints*. Creating an accurate coherent model of a robot is one of the most difficult steps of using MoveIt and is an area in need of future improvement.

MoveIt Setup Assistant: The main facility that provides out of the box support for beginners is the MoveIt Setup Assistant (SA). The SA is a GUI that steps new users though the initial configuration requirements of using a custom robot with the motion planning framework (Figure 1). It accomplishes the task for the user of automatically generating all the many configuration files necessary for the initial operation of MoveIt. These configurations include a self-collision matrix, planning group definitions, robot poses, end effector semantics, virtual joints list, and passive joints list. To increase the immediacy of results and transparency of the configuration, a three dimensional model of the robot as its being configured is displayed on the right side of the Setup Assistant GUI and various links on the robot are highlighted during configuration to visually confirm the actions of the user, as shown in Figure 1.

Using a properly formatted robot model file with the SA, MoveIt can automatically accomplish many of the required tasks in a motion planning framework including forward and

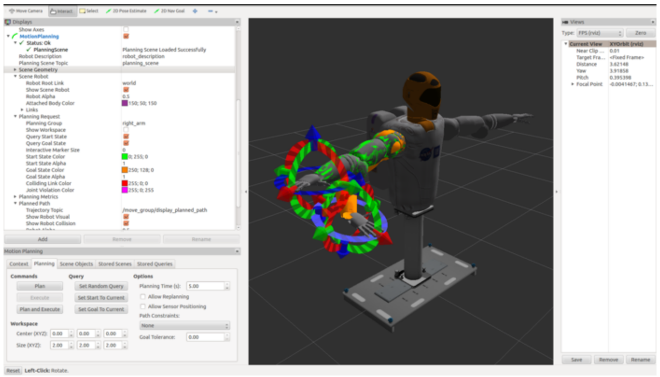

Fig. 2. Screenshot of MoveIt Rviz Motion Planning Plugin with the Robonaut planning between two configurations TODO: better picture

inverse kinematics, collision checking, and joint limit enforcement. If one desired, the steps within the SA could almost entirely be automated themselves, but they have been kept manual so 1) increase transparency and 2) provide extensibility for edge cases and unusual customizations.

MoveIt Rviz Motion Planning Plugin: The details of the automated configuration is left for the next section, but after the steps in the SA are completed the output is a ROS package containing a collection of configuration files and launch scripts customized for the particular robot. The launch files include a demo script that will automatically startup a visualization tool (rviz) with the new robot loaded and ready to run state of the art motion planning algorithms in a non-physics based simulation. The typical demo task would be using the computer mouse to visually drag interactive arrows situated on the robot's end effector from a start position to a goal position around some virtual obstacle. The demo can then quickly plan the arm in a collision free path around the obstacle and visualize the results within rviz.

This user interaction is accomplished with the MoveIt Rviz Motion Planning Plugin (RMPP), an additional GUI that allows beginning users to learn and experiment with a large subset of the functionality provided by MoveIt (Figure 2). While the long term goals of robotics is to provide more autonomous solutions to motion planning and human-robot interactions [22], the RMPP fulfills the immediate needs of direct operation for testing and debugging the framework's capabilities easily. This interface is a vital component of MoveIt's strategy to provide immediate results for motion planning with a robot that does not require any custom coding. Once the user is comfortable with the basic feature set and functionality of MoveIt, extensibility is provided via varying levels of code APIs for more direct, non-GUI, access to the robot's abilities.

The RMPP provides a large number of features and visual tools for motion planning. Using Rviz and RMPP, visualizations are provided of:

- Start and goal configurations of the robot for planning
- Current robot hardware configuration
- Animated planned path before executing
- Detected collisions
- Sensor data and recognized objects
- Pick and place data such as grasp positions
- Attached bodies such as manipulated objects
- Planning Metrics

Additionally, the RMPP contains many other non-visualization tools such as:

- Connecting to a database of planning scenes
- Adjusting IK settings
- Changing the utilized planning algorithm
- Adjusting the workspace size
- Adjusting goal tolerance and planning time
- Tweaking manipulation plans
- Loading and moving collision objects
- Exporting/Importing scenes and states
- Viewing the status of MoveIt

Hardware Configuration and Execution: Once the user is comfortable with the basic tools and features provided by MoveIt, the next step is to configure their robot's actual hardware actuators and control interfaces to accept trajectory commands from MoveIt. This step is not as easy as it requires some custom coding to account for the specifics of the robot hardware - the communication bus, real-time requirements, and control theory implementations. At the abstract level, all MoveIt requires is that the robot hardware accepts a standard ROS trajectory message containing a discretized set of time-variant waypoints including desired positions, velocities, and accelerations.

5.2 Automate configuration and optimization

The size and complexity of a feature-rich motion planning framework like MoveIt requires many parameters and configurations of the software be automatically setup and tuned. MoveIt accomplishes this in the 1) setup phase of a new robot, using the Setup Assistant, 2) during the runtime of the application, and 3) using benchmarking and parameter sweeping.

Self Collision Matrix: The first step of the SA is the generation of a self-collision matrix for the robot that is used in all future planning to speed up collision checking. This collision matrix encodes pairs of links on a robot that never need to be checked for self-collision due to the kinematic infeasibility of there actually being a collision. Reasons for disabled collision checking between two links includes:

- Links that can never intersect due to kinematics
- Adjacent links that are connected and so are by design in collision
- Links that are always in collision for any other reason including inaccuracies in the robot model and precision errors

This self-collision matrix is generated by running the robot through tens of thousands of random joint configurations and recording statistics of each link pair's collision frequency. The algorithm then creates a list of link pairs that have been determined to never need to be collision checked. This reduces future motion planning runtimes because it reduces the amount of required collision checks.

Configuration Files: The other six steps of the SA all provide graphical front ends for the data required to populate the Semantic Robotic Description Format (SRDF) and other configuration files used by MoveIt. The SRDF provides semantic meta data of the robot model useful to motion planning but not relevant to the URDF because it does not describe physical properties of the robot. The SRDF describes meta data such as which set of joints constitutes an arm and which set of links is considered part of the end effector. It is one of the main components that allows MoveIt to be robot agnostic and to avoid dependencies on specific robots [2]. Requiring the user to configure all the semantic information by hand in a text editor would be far more tedious and difficult than using an interface that populates the available options for each required field and guides the user through its completion.

The last step of the SA is to generate all launch scripts and configuration files. This step includes outputting to file the collected configurations during the step-by-step user interface, as well as generates a series of default configuration and launch files that are automatically customized for the particular robot using the URDF and SRDF information. These defaults include velocity and acceleration limits for each joint, kinematic solvers for each planning group, available planning algorithms, and projection evaluators for planning. Default planning adapters are setup for pre- and post-processing of motion plans. Default benchmarking setups, controller and sensor manager scripts, and empty object databases are all generated.

Optimization: The ability to configure and switch out different planning components is a powerful feature of MoveIt, but its usefulness is limited without the ability to quantify the results of different approaches. Optimization criteria such as path length, planning time, smoothness, distance to nearest obstacle, and energy minimization need benchmarking tools to enable users and developers to find the best set of parameters and planning components for any given robotic application.

MoveIt lowers the barrier to entry to benchmarking by providing a command line-based infrastructure and benchmarking configuration files that allows each benchmark to easily be setup for comparison against other algorithms and parameters [19]. An additional GUI is currently in development that will make benchmarking even easier and reduce the learning curve.

A common method to optimize an algorithms performance is to perform single and multivariable parameter sweeps during benchmarking. MoveIt provides an interface for this in its benchmarking infrastructure by allowing a upper, lower, and increment search values to be provided by the user. Results can be output into generic formats for use in different plotting tools for analysis of which combination of parameters performed best.

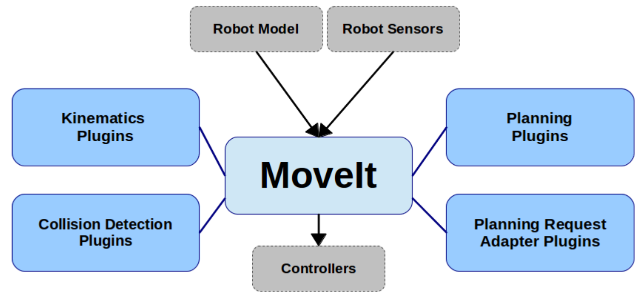

Fig. 3. Available planning component plugins for easily extending the functionality of MoveIt.

5.3 Easily customize components of the framework

Out of the box MoveIt lowers the barrier to entry by not requiring users to provide their own implementation of any of the components in the motion planning framework. The default planning components are the aforementioned OMPL, FCL, and KDL. However, these default components are limiting to more advanced users who have their own application or research-specific needs to fulfill.

MoveIt is *extensible* by allowing its various planning components to be customized through a lightweight plugin interface. This is accomplished using C++ shared objects that are loaded at run time, reducing dependency complexities. This plugin-centric framework, as seen in Figure 3, provides interfaces for forward and inverse kinematics, collision detection, planning, and planning request adapters. Planning request adapters provide necessary pre- and post-processing of motion plans such as fixing slightly invalid start states and smoothing generated trajectories.

A particular strong point of MoveIt's feature set is its kinematics plugins. The default KDL plugin uses numerical techniques to convert from a Cartesian space to joint configuration space. A must faster solution can be achieved by utilizing OpenRave's IKFast [23] plugin that analytically solves the inverse kinematics problem. A combination of MoveIt scripts and the IKFast Robot Kinematics Compiler can automatically generate the C++ code and plugin needed to increase the speed of motion planning solutions by up to 3 orders of magnitude [23].

Essentially, MoveIt provides a set of data sharing and synchronization tools, sharing between all planning components the robot's model and state. The extensibility of MoveIt's framework is greatly enhanced by not forcing users to use any particular algorithmic approach.

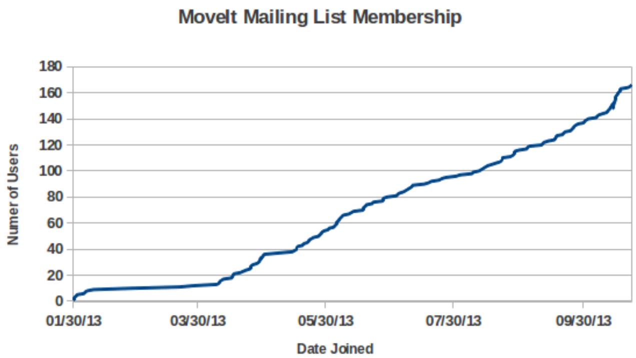

Fig. 4. Membership of MoveIt mailing list over time.

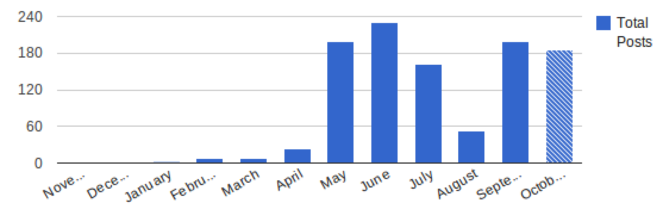

Fig. 5. Total posts on the MoveIt mailing list. TODO: add axis labels

6 Results

The success of MoveIt's efforts to lower its barrier of entry to new users through the application of the barrier to entry principles is quantified in the following. Its adoption rate, community activity, contributors, and results from a user survey are used as indicators of its progress.

6.1 Statistics

MoveIt was officially alpha released on May 6th, 2013. As of this writing, 170 days since its release, there have been XX debian downloads from its supported Ubuntu operating system.

There are currently 165 users on the MoveIt mailing list as shown over time in Figure 4. The posting activity of the mailing list over time in shown in Figure 5.

There have been a total of 48 contributors to the code base since its initial development began in 2011. The number of contributors over time is shown in Figure 6. According to statistics gathered by Ohloh, MoveIt is one the largest open-source teams in the world and is in the top 2% of all project teams on Ohloh [24].

6.2 Survey

A survey on MoveIt user's experience with the software were administered within

details
details

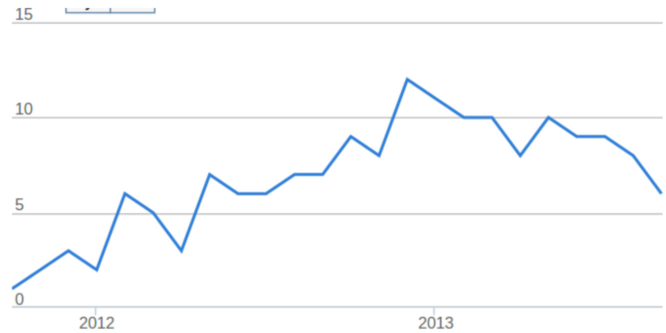

Fig. 6. Total code contributors to MoveIt across all repositories. TODO cleanup diagram, add titles and axis labels

details
details
details
details
details
plot
plot
plot
plot
plot
plot
plot
plot
plot
plot
plot
plot

7 Discussion

We believe the setup process for MoveIt is easier than most open source robotics software available today, and as a result it has in a short amount of time become popular in the robotics community as a powerful motion planning framework that is extensible to most user's needs. The adoption rate of MoveIt since its official release less than half a year ago has been very positive and the number of contributors has been better than expected. Ohloh's ranking of MoveIt as one of the largest open source teams in the world confirms our belief that by making complex software more accessible, more developers will be able to report and fix issues.

It is important to emphasize the effect of a quick *Getting Started* demo on a new user unaccustomed to MoveIt or motion planning in general. The positive reinforcement of a quick initial success encourages the novice to continue to use the software and enables them to begin going deeper into the functionality and code base. If the entry barrier is too high, that is to say it is too complex with , a new user will likely give up and turn to other frameworks or custom solutions rather than continue to blindly fix software that they have no experience in. TODO: move this paragraph?

The setup and configuration process

Rigid body dynamics

"Will conclude with our experiences in the development process."

There are still barriers to entry

Setting up controllers is difficult

Large code base is intimidating

Built by one programmer so not the easiest layout

Creating the SA or any GUI is a time consuming process that many robotics developers avoid in favor of manual or command-line based configuration. As discussed in the introduction, we believe that the trade off in the time invested was worthwhile for the higher adoption rates and

8 Conclusion

Beyond the usual considerations in building successful robotics software, an open source project that desires to maintain an active user base needs to take into account the barrier of entry to new users. As the algorithms become more complicated and the number of components and code base increases, configuring an arbitrary robot to utilize robotic software becomes a daunting task requiring domain-specific expertise in a very large breadth of theory and implementation. To account for this, quick and easy initial configuration, with partially automated optimization, and easily extensible components for future customization are becoming a greater necessity in motion planning and robotic software engineering in general.

Acknowledgments

The authors would like to thank...

References

[1] M. Quigley, K. Conley, B. 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*, vol. 3, no. 3.2, 2009. 1

[2] S. Chitta, I. Sucan, and S. Cousins, "Moveit!" *IEEE Robotics Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012. 1, 5.2

[3] H. Bruyninckx, "Open robot control software: the orocos project," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 3. IEEE, 2001, pp. 2523–2528. 1

[4] A. Makarenko, A. Brooks, and T. Kaupp, "On the benefits of making robotic software frameworks thin." 1, 1.2, 3

[5] J. M. Carroll, *Interfacing thought: Cognitive aspects of human-computer interaction.* The MIT Press, 1987. 1.1

[6] D. C. Schmidt, "Why software reuse has failed and how to make it work for you," *C++ Report*, vol. 11, no. 1, p. 1999, 1999. 1.2

[7] D. C. Schmidt and A. Porter, "Leveraging open-source communities to improve the quality & performance of open-source software," in *Proceedings of the 1st Workshop on Open Source Software Engineering*, 2001. 1.2

[8] N. Correll, R. Wing, and D. Coleman, "A one-year introductory robotics curriculum for computer science upperclassmen." 1.2

[9] M. Moll, I. A. Sucan, J. Bordeaux, and L. E. Kavraki, "Teaching motion planning concepts to undergraduate students," in *Advanced Robotics and its Social Impacts (ARSO), 2011 IEEE Workshop on*. IEEE, 2011, pp. 27–30. 1.2, 2

[10] Y. hsin Kuo and B. MacDonald, "A distributed real-time software framework for robotic applications," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, 2005, pp. 1964–1969. 1.3

[11] T. H. Collett, B. A. MacDonald, and B. P. Gerkey, "Player 2.0: Toward a practical robot programming framework." 1.3

[12] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, no. 2, pp. 101–132, 2007. 1.3

[13] A. Perez and J. Rosell, "A roadmap to robot motion planning software development," *Computer Applications in Engineering Education*, vol. 18, no. 4, pp. 651–660, 2010. 2

[14] K. A. Wyrobek, E. H. Berger, H. M. Van der Loos, and J. K. Salisbury, "Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 2165–2170. 3

[15] R. O. Ambrose, H. Aldridge, R. S. Askew, R. R. Burridge, W. Bluethmann, M. Diftler, C. Lovchik, D. Magruder, and F. Rehnmark, "Robonaut: Nasa's space humanoid," *Intelligent Systems and their Applications, IEEE*, vol. 15, no. 4, pp. 57–63, 2000. 3

[16] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org. 3

[17] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012, pp. 3859–3866. 3

[18] R. Smits. (2013, Oct.) Kdl: Kinematics and dynamics library @ONLINE. [Online]. Available: http://www.orocos.org/kdl 3

[19] B. Cohen, I. A. Sucan, and S. Chitta, "A generic infrastructure for benchmarking motion planners," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 589–595. 3, 5.2

[20] W. Garage. (2013, Oct.) Urdf: Universal robotic description format @ONLINE. [Online]. Available: http://wiki.ros.org/urdf 5.1

[21] (2013, Oct.) Iso/pas 17506:2012 industrial automation systems and integration – collada digital asset schema specification for 3d visualization of industrial data @ONLINE. [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=59902 5.1

[22] M. W. Kadous, R. K.-M. Sheh, and C. Sammut, "Effective user interface design for rescue robotics," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, ser. HRI '06. New York, NY, USA: ACM, 2006, pp. 250–257. [Online]. Available: http://doi.acm.org/10.1145/1121241.1121285 5.1

[23] R. Diankov. (2013, Oct.) Ikfast: The robot kinematics compiler @ONLINE. [Online]. Available: http://openrave.org/docs/latest_stable/openravepy/ikfast/#ikfast-the-robot-kinematics-compiler 5.3

[24] Ohloh. (2013, Oct.) Ohloh: Moveit! project summary factoids @ONLINE. [Online]. Available: https://www.ohloh.net/p/moveit_/factoids#FactoidTeamSizeVeryLarge 6.1

First Author received his B. Sc. and M. Sc. degrees in mechanical engineering from the *** University, in 1977 and 1984, respectively, and the Ph. D. degree in computing from *** University, in 1992. In 1994, he was a faculty member at *** University and in 1996 at *** University. Currently, he is a professor in the Department of Information System Engineering at *** University. He has published about 100 refereed journal and conference papers. His research interest covers robotics, software engineering, and distributed systems. Prof. Author received research award from Science Foundation, and the Best Paper Award of the XX International Conference in 2000 and 2006, respectively. He is a member of ACM and IEEE.

Second Author received his B. Sc. and M. Sc. degrees in mechanical engineering from the *** University, in 1977 and 1984, respectively, and the Ph. D. degree in computing from *** University, in 1992. In 1994, he was a faculty member at *** University and in 1996 at *** University. Currently, he is a professor in the Department of Information System Engineering at *** University. He has published about 100 refereed journal and conference papers. His research interest covers robotics, software engineering, and distributed systems. Prof. Author received research award from Science Foundation, and the Best Paper Award of the XX International Conference in 2000 and 2006, respectively. He is a member of ACM and IEEE.