Examining Robotic Systems with Shape-Adjustable Manipulators under Dynamic Environments: from Simulation to Verification

Chih-Hong Cheng* and Alois Knoll* and Christian Buckl** and Javier Esparza* and Yang Chen*

***Abstract*— In this paper, we present our preliminary report in applying formal verification to the design process of robotic systems under dynamic environments; the goal is to complement existing testing or simulation techniques by experimenting an adaptable framework, where verification models with tamable complexity are generated from the simulation model. Our targets are robotic systems with shape-adjustable manipulators (e.g., robot arms), which in essence bring different challenges compared to existing research. By investigating the problem structure, we propose ingredients for successful verification of such systems, conduct experiments, and outline future studies.**

I. Introduction

As current trends in robotics advocate interactions between robots and humans, robotic design with stronger safety claims is of high interests among industries. To complement existing approaches using testing or simulation, we investigate possibilities to perform formal verification on robotic systems.

The introduction of automatic formal verification is to release designers' burden concerning quality assurance. It has been applied in both hardware and software; research and industry advances have made techniques applicable even on source code level, for example, [1], [7], [12]. Designers can specify desired properties of the system, and the verification engine either reports a concrete satisfaction proof or a counter-example witness.

Verification techniques has also been applied in real-time or cyber physical systems, where components are composed and coordinated to accomplish certain tasks. As we treat robotic systems as component-based, analogies can be applied similarly. Although our ultimate goal is to construct a model-based workbench tailored for robotics, which automates the design-verification-codegen flow similar to [4], the first problem we encounter immediately is to perform verification effectively under the particular problem structure of robotics, which will be the main focus of this work. In this paper, the context of verification is on system level, i.e., the goal is to check mathematically whether a given system configuration of sensors, actuators, and controllers can satisfy certain safety criteria

We proceed our presentation as follows: We start with a simple (yet representative) scenario to facilitate further studies (sec. II), followed by a brief introduction of verification techniques we used (sec. III). We then specify a meet-in-the-middle design methodology (sec. IV), which offers a mediated view of designing robotic controllers while facilitating verification. Nevertheless, we observe that there exist special challenges when verifying shape-adjustable robots in comparison to shape-nonadjustable robots. Concerning these challenges, we propose our theoretical solutions for successful verification (sec. V). For evaluation, we have two experiments with reports and analysis (sec. VI), and offer analogies facilitating the process from simulation to verification. We review existing work and conclude in section VII and VIII.

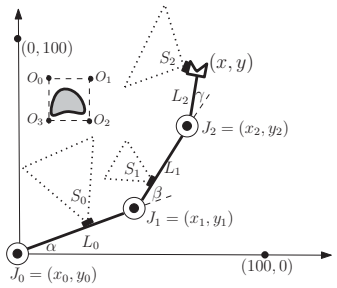

Fig. 1. A robot arm and an obstacle are moving in an 100×100 squared area; S_1, S_2 and S_3 are attached distance sensors (the dotted triangle represents the effective sensing range).

As our research area is interdisciplinary, our presentation must contain self-explanatory materials in both verification and robotics. By doing so, we hope that our process of introducing verification in robotics can be replicated for robotic researchers, and our proposed challenges can be understood clearly and refined by researchers in the verification community.

II. Scenario: Simple 2D Robot Arm

We first clarify our definition concerning robots with shape-adjustable manipulators. Intuitively, given two interior points of a shape-adjustable manipulator, the distance of these two points can vary over time. As robot arms are amongst commonly used manipulators in industries, in following sections we only use robot arms for discussion.

A robot arm consists of joints and links, and an end-effector is attached on it for object retrieval. Throughout this paper, our formulation will be based on 2D-robot arms with 3 revolute joints (as shown in fig. 1); in this case, the position-orientation pair (x, y, θ) of the end-effector has at most one corresponding configuration (α, β, γ), representing angular displacement of each joint.

In this report, we set up a relatively simple goal for safety criterion, that is to guarantee that given the behavioral model

*Department of Informatics, TU München, D-85748 Garching, Germany. {chengch,knoll,esparza,yang.chen}@in.tum.de

**fortiss GmbH, Guerickestr. 25, D-80805 Munich, Germany. buckl@fortiss.org

of an obstacle, the robot arm under control should never collide with the moving obstacle. However, the criterion can be more general; it is applicable for physical properties concerning spacial movements among objects and robots.

A. Physical Space and Joint Space

One important feature in robotics is that during the operation of a robot, it continuously switches its view between its *physical space* and its *joint space*. The physical space represents the set of states perceived by the environment, and the joint space represents the set of states directly related to the configuration of actuators. In our example, the orientation (x, y, θ) of the end-effector belongs to the representation in the physical space, while angular displacements (α, β, γ) are representations in the joint space.

III. Model Checking

In computer science, formal verification is a research discipline which applies mathematical methods to examine the correctness of the system under certain properties. As verification itself is a very broad field with numerous approaches, here we apply one particular approach called model checking [5], and overlook other methods like theorem proving, abstract interpretation, and so on. From our point of view, it is relatively easy to introduce model checking into engineering projects, compared to other methods.

Model checking has the following features:

- It is *automatic*; the process of checking does not require any intervention of developers.
- It is *exhaustive*; if the result indicates that the model is correct, it is definitely correct as it had calculated all possibilities before the result is reported. If the property fails, a counter example will be generated automatically.
- To perform model checking, a *model* and the *specification* (unambiguous definition of correctness; can be partial to certain aspects) must be given beforehand. Regarding the process of model construction, we give concretized illustrations and analogies in our experiment section (sec. VI).

IV. Mediating the Gap between Robotics and Verification

When performing verification in the field of robotics, we have to overcome inherent differences between two disciplines: In robotics, operations are commonly operating over real numbers (i.e., infinite states), while in verification currently mature and efficient techniques only allow finite-domain integer manipulations. Unfortunately, verification in infinite state systems can only be achieved with limited extensions while maintaining decidability results; we have to be cautious concerning the boundary of decidability.

Therefore, to make verification possible, we advocate common approaches to start with an abstract design, followed by refined implementation. However, our abstract design will be based on manipulation of integers (or fixed-point numbers) and thus verifiable; the implementation is a behavioral refinement of the abstract design, and what has been proven in the abstract design remains applicable. The drawback of this approach can be sacrificing of optimality, while we regard it a tradeoff between efficiency and safety.

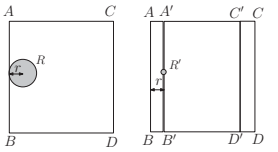

Fig. 2. The real robot with obstacles, where $\overline{AB}$ and $\overline{CD}$ are cliffs (a), and the abstract point robot with the expanded radius of cliffs with C-Space approach (b).

A. Special Features for Shape-Adjustable Robots

The above approach has been applied in some field of robotics, for example, path planning or collision avoidance[1]. To our best knowledge, existing work mainly focus on shape-nonadjustable robots (e.g., robot platforms). Nevertheless, our investigated problem (robots with shape-adjustable manipulators) is more complex because of two reasons:

- First, for planar robots with non-changeable shapes, when C-space approach is applied, the collision problem can be simply converted to a point-reachability problem; the position-time trajectory of the robot is merely a curve in three dimensions (X-axis, Y-axis, Time axis). We illustrate this concept using fig. 2. In fig. 2(a), the radius of the circular robot R is r, while line segments $\overline{AB}$ and $\overline{CD}$ represent cliffs along road sides - if the robot has reached the cliff, it is regarded unsafe for the reason of potential falling. When the C-space approach is applied, the robot can be simplified to point R' shown in fig. 2(b), while two cliffs $\overline{AB}$ and $\overline{CD}$ are pushed further to $\overline{A'B'}$ and $\overline{C'D'}$, respectively. Comparing fig. 2(a) and fig. 2(b), we can conclude that *R is safe iff point R' never reaches $\overline{A'B'}$ and $\overline{C'D'}$*. However, for robot arms, this approach is not applicable, since the shape is continuously adjusted (as shown in fig. 1), and no trivial abstraction exists.
- As joint space configurations influence directly the shape, this information must be part of the model description. Concerning shape-nonadjustable robots, it is not required, and a simpler formulation which overlooks the configuration of actuators is possible.

V. Towards the Verification of Shape-Adjustable Robots

Based on previous observations, we mention features of our proposed solutions for verification of shape-adjustable robots.

A. Discrete Time Update

The first assumption of our approach is to treat time in the verification model discrete rather than continuous. In verification, continuous time verification is in general computationally more expensive than discrete time verification;

[1]Here we focus on dynamic changing environments.

in our problem, the use of continuous time might easily lead to explosion of memory without generating any meaningful results. By selecting an appropriate fine-grained accuracy in combination with suitable over-approximation, we can provide the same guarantee regarding system safety.

B. Manipulation between Physical Space and Joint Space

For controlling robot arms, the change between joint space and physical space requires inevitably the call of complex functions not implemented in a verification engine. We illustrate the process again using fig. 1. For this robot arm, the parameter (x, y, θ) of the end-effector and the joint angles have the following relations:

$$x = x_0 + L_1 \cos\alpha + L_2 \cos(\alpha+\beta) + L_3 \cos(\alpha+\beta+\gamma)$$

$$y = y_0 + L_1 \sin\alpha + L_2 \sin(\alpha+\beta) + L_3 \sin(\alpha+\beta+\gamma)$$

$$\theta = \alpha + \beta + \gamma$$

Therefore, each time when the robot updates its moves, three functions should be calculated. In robotics, these operations are called *forward kinematics*, which is required for occupied space update in verification. On the other hand, *inverse kinematics* is used to decide given a certain position (x, y, θ) in the physical space, the set of required angle positions for each joint in the joint space. This is commonly used in all controller algorithms. Besides mathematical functions used in forward kinematics, for 2D robot arms, calculating inverse kinematics requires additional functions, e.g., square root functions, inverse cosine/tangine functions.

In addition, as most verification engines can only manipulate with integers, the above operations must be carefully redesigned to fit in the existing context of verification engines. For example, to calculate $L_1\cos(\alpha)$ in the verification engine, one possible method is to first calculate the multiplication of L_1 and the numerator of $\cos(\alpha)$, followed by dividing the denominator of $\cos(\alpha)$. Because of this, traditional range of integers is not enough; manipulation over large integers should also be encoded.

We summarize our observations as follows:

Ingredients for Robotic Verification 1: For verification of robotic systems, at least the following mathematical functions should be encoded:

1) Trigonometric functions and their inverse.
2) Square root functions.
3) Arithmetic operations over large integers.

C. Taming Complexities for Shape Information

Besides encoding mathematical functions, another important concern is to control the set of variables used in the verification engine. Without explicit control, the state space might easily turn too large for any existing verification engines to manipulate.

It is worthwhile to perform a rough estimation on the robot (without attached sensors) in fig. 1 concerning the number of variables (or bits) used in verification; the number of states is exponential to the number of bits used in variable declarations in the model:

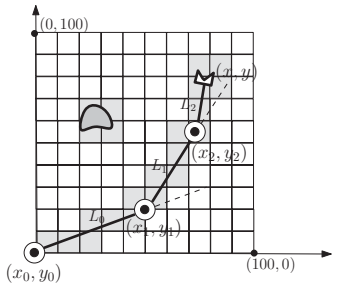

Fig. 3. Coarser grids for the use of verification space.

1) For the moving obstacle, its position, speed, and acceleration should be defined as variables in verification engines.
2) For each joint, its angular displacement, angular velocity, and angular acceleration should also be encoded.
3) Lastly, we need to record the occupied space of the 2D robot arm.

Since the use of the first two variable categories is unavoidable, an efficient manipulation of occupied space recording plays a crucial role whether verification can generate results within reasonable time. One intuitive but parsimonious approach is to use a 100×100 boolean matrix to keep the record of the object, implying the use of 10000 bits. However, existing techniques in verification cannot tackle systems with such complexities. In the following, we propose two conceptual methods, and discuss their theoretical benefits and drawbacks.

1) Differentiation of Physical Space and Verification Space: The first conceptual method tries to differentiate the verification space of the robot from the physical space using abstractions. For fig. 1, instead of using a 100×100 matrix M for verification space, we can apply an abstraction coefficient c_{abs}, such that we have a smaller $\frac{100}{c_{abs}} \times \frac{100}{c_{abs}}$ matrix M'. If $c_{abs} = 10$, $M'_{i,j} = true \Leftrightarrow \exists i', j'$, where $\frac{100}{10} i \leq i' < \frac{100}{10}(i+1)$, $\frac{100}{10} j \leq j' < \frac{100}{10}\ (j+1)$, such that $M_{i',j'} = true$. For other variables, no further abstraction applies. An illustration can be found in fig. 3, where an coarser abstraction with $c_{abs} = 10$ is applied. In this way, in 2D robotics the number of boolean variables used in the verification space can be reduced by a factor of c^2_{abs}.

Considering the verification condition, it will simply be a statement mentioning if the object is in position (x, y), the matrix element with corresponding indexes should always be false.

As for evaluation, this method can generate false positives due to the imprecision of the verification space; thus the size of the abstracted grid cannot be too large. Also, this method is unlikely to be applied in 3D robotics for the excessive usage of variables; concerning our example, it will use 10^3 boolean variables for space recording in 3D.

2) Vertex Recording with Complex Specifications: The second conceptual method is achieved by recording vertices of exterior points on edges of the robot and the obstacle, and in 2D robots the specification (collision condition) is described using the occurrence of line-segment intersection.

GENERATECOLLISIONCONDITION(J_0, J_1, O_0, O_1){
/* $J_0 = (x_0, y_0)$, $J_1 = (x_1, y_1)$, $O_0 = (x_{O_0}, y_{O_0})$, $O_1 = (X_{O_1}, y_{O_1})$ */
/* Define the following predicate:
$On(x_a, y_a, x_b, y_b, x_c, y_c) = (\min(x_a, x_b) \leq x_c) \wedge (x_c \leq \max(x_a, x_b)) \wedge (\min(y_a, y_b) \leq y_c) \wedge (y_c \leq \max(y_a, y_b))$.
Define syntactic sugar $D_1 := (J_0 - O_0) \times (O_1 - O_0)$.
Define syntactic sugar $D_2 := (J_1 - O_0) \times (O_1 - O_0)$.
Define syntactic sugar $D_3 := (O_0 - J_0) \times (J_1 - J_0)$.
Define syntactic sugar $D_4 := (O_1 - J_0) \times (J_1 - J_0)$. */
Let $\psi = false$;
$\psi = \psi \vee (((D_1 > 0) \wedge (D_2 < 0)) \vee ((D_1 < 0) \wedge (D_2 > 0)) \wedge ((D_3 > 0) \wedge (D_4 < 0)) \vee ((D_3 < 0) \wedge (D_4 > 0)))$;
$\psi = \psi \vee ((D_1 == 0) \wedge On(x_{O_0}, y_{O_0}, x_{O_1}, y_{O_1}, x_0, y_0))$;
$\psi = \psi \vee ((D_2 == 0) \wedge On(x_{O_0}, y_{O_0}, x_{O_1}, y_{O_1}, x_1, y_1))$;
$\psi = \psi \vee ((D_3 == 0) \wedge On(x_0, y_0, x_1, y_1, x_{O_0}, y_{O_0}))$;
$\psi = \psi \vee ((D_4 == 0) \wedge On(x_0, y_0, x_1, y_1, x_{O_1}, y_{O_1}))$;
return ψ;
}

Fig. 4. Algorithm generating logical formula for collision between two line segments $\overline{J_0J_1}$ and $\overline{O_0O_1}$ adapted from [6].

To achieve this goal, we adapt existing algorithms in computational geometries [6] to generate specifications.

As the number of variables used in the verification is reduced, we can expect the speed increase of the verification process. Furthermore, by adapting this method, it is possible to encapsulate the spacial information when we move to a 3D robot arm:

1) An industrial robot in general consists of 3 links.
2) Each link can be represented as a parallelepiped, which can be described using 8 points.
3) For each point, 3 variables are used for each axis.

Therefore, with each axis of range $[0, 100]$ (can be represented by 7 bits), the least number of boolean variables required for verification space can be $3 \times 8 \times 3 \times 7 = 504$; a system with such complexity is considered solvable by verification engines.

To adapt algorithms in computational geometry to specifications, there are several criteria under considerations. First, not all algorithms are applicable. For example, these algorithms should not include division; verification engines manipulating on integer division lose precision, and this can generate false negatives of the result. Furthermore, as in essence the algorithm will be flattend to logic formula, the algorithm should not introduce new variables.

Nevertheless, the drawback of this approach is the difficulty of describing the specification; conditions which lead to collision must be enumerated explicitly. Consider fig. 1, which is a simplified case where each link is merely a straight line. With the rewriting of the line intersection algorithm utilizing cross products in [6], the condition for link $\overline{J_0J_1}$ to collide with $\overline{O_0O_1}$, $O_1 = (x_{O_1}, y_{O_1})$ and $O_0 = (x_{O_0}, y_{O_0})$, can be constructed by the $\mathcal{O}(1)$ algorithm in fig. 4. In our experience, the specification is hard to grasp.

VI. Experiments and Evaluation

A. 2D Robot Arm in Lego Mindstorm NXT Platform

In our experiment, we first construct a 2D robot arm based on the Lego Mindstorm NXT, where the control software is written using RobotC (a C-like language). Given the target position-orientation pair of the end-effector, the controller software can calculate the inverse kinematics for required angular displacements, and trigger motors for rotation of joints based on calculated results. However, since the rotation motor is not able to control the acceleration precisely, this demonstrator is used merely for concept formulations.

1) Mathematical Functions: In the control software, to fulfill the requirement of verification (ingredients for robotic verification mentioned earlier), we implement all mathematical functions operating over integers. This is achieved by a binary search over predefined look-up tables.

B. Simulation and Verification of a Hypothetical Robot Arm

In our second experiment, the goal is to determine whether collisions between the robot and the obstacle occur given the robot and the behavior model of the obstacle. Our tasks can be divided into two:

- We first design a simple simulation environment; a number of perfect hypothetical 2D robot arms can be attached in the environment. For each arm, each joint can be controlled using constant accelerations specified by the controller program. Also, sensors can be attached arbitrarily on the robot. Lastly, the user can manually control the obstacle movement for avoidance checking. Fig. 5-(a) shows the conceptual framework of the simulation environment.
- Based on one particular system setting, we manually construct the corresponding verification model using UPPAAL [11][2]. Since UPPAAL supports C-like functions as part of the model description, previous mathematical functions implemented in the Lego Mindstorm NXT can be completely reused.

(Remark) We can establish an analogy by comparing the simulation environment in fig. 5-(a) and the verification environment in fig. 5-(b). In these two settings ingredients are roughly the same:

[2]The example can be downloaded from `http://www6.in.tum.de/~chengch`.

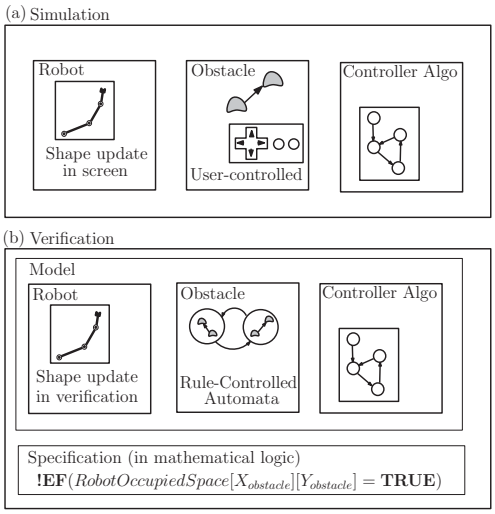

Fig. 5. Conceptual framework for robot simulation (a) and verification (b).

- Both in simulation and verification the model of the robot is required. However, in simulation the shape update will be shown in the screen, while in verification the shape update must be stored as a model itself or be part of the inner-states in the engine.
- Both in simulation and verification the capture of object movement is required. However, in simulation the object movement is specified as test cases or is controlled by human, while in verification the object movement is specified as rules (automata).
- The controller algorithms are identical in simulation and verification, though they differ in textural formats.

In the following, we describe ingredients in our constructed verification model (obstacle, controller algorithm, robot), as shown in fig. 5-(b).

1) Obstacle Movement: First we must clarify the behavior model of the moving obstacle. For the use of safety verification, an overapproximation of the behavior is desirable. Consider the abstraction scheme in figure 6. To simplify our discussion, a single point object is used[3], and the time for discrete state change is 1 second. If an abstraction is applied such that for all positions in the rectangle $(a_{i,j}, a_{i+1,j}, a_{i+1,j+1}, a_{i,j+1})$, we use the point b_{ij} to represent its presence, then the behavior model specifying that b_{ij} can move to $b_{i'j'}$, where $i' \in \{i, i+1, i-1\}$ and $j' \in \{j, j+1, j-1\}$, is sufficient to describe all possible behavior of the object which has the maximum speed equal to 1 $unit/sec$.

Overall, obstacle movements in our verification model have the following features:

- **(Discretized Directions)** We define 9 directions, namely $North$, $NorthEast$, $East$, $SouthEast$, $South$, $SouthWest$, $West$, $NorthWest$, and $Still$.

[3]In our verification scheme, it corresponds to the geometric center of the object; we then construct the overapproximated boundary of the obstacle, and establish the collision condition. Another method is to use multiple exterior points of the obstacle; verification conditions can still be generated analogously.

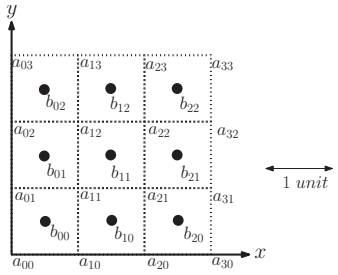

Fig. 6. An illustration of movement abstraction.

With the given direction and speed of the obstacle, the location can be updated as discrete time advances.

- **(Reasonable Trajectories)** Based on predefined directions, we have explicit constraints over allowable trajectories. For example, between consecutive two timer readings, the obstacle is not allowed to change to its opposite direction.
- **(Constrained Arena)** We define boundaries concerning obstacle movements.

2) Controller Algorithm: For the controller under verification, we put a self-designed algorithm called *simple end-effector avoidance algorithm* into the verification model. The sketch of the algorithm is as follows: First calculate the distance between the end-effector[4] and the obstacle, followed by comparing the distance with the user specified "risk" range.

- If the distance is greater than the "risk" range, then do nothing.
- Otherwise perform the following actions.
  1) Use `decideNewLocation()` to generate the future position of the end-effector.
  2) Use `calInvKinematics()` to calculate required angular displacements of each joint compared to the current angular position.
  3) Perform joint movement, which consists of two phases.
     - Use `accelerate()` to configure the angular acceleration of the robot, such that it accelerates its speed for a specified time with constant rate.
     - Use `decelerate()` to configure the angular acceleration of the robot, such that it decelerates its speed for a specified time with constant rate.

3) Robot Arm with Adjustable Constant Acceleration: For robot arm, it is designed such that for every timer update, it performs the following actions.

- Each joint updates its angular displacement and velocity, based on the its previous configuration.
- Based on the updated joint angles, use forward kinematics to calculate the position of each joint.
- Update its abstract shape in the boolean two-dimension matrix. This step is omitted if the method in section V-C.2 is applied.

[4]In our settings, the omnidirectional distance sensor is attached on the end-effector.

Method	Differentiation (sec. 5.1)		Complex Spec (sec. 5.2)
Matrix size	$20 * 20$	$10 * 10$	NONE
Execution Time	No result	93.39 sec	77.70 sec
Memory Consumption	> 2 GB	419.51 MB	112.90 MB

Fig. 7. Result of preliminary evaluation.

- Update the angular acceleration of each joint if the controller algorithm changes it explicitly.

C. Result

Based on above settings, we experiment with two model construction methods, and the result is shown in fig. 7; data is collected from the system using Intel Duo-Core 2.33 Ghz CPU and 3GB RAM. By applying the first method, a verification model using 20×20 boolean matrix has already been too complicated to manipulate within predefined memory limits. Concerning the second approach, as fewer variables are used in the system, the result of verification can be generated relatively fast; it is also memory efficient compared to the first construction method. When memory consumption is crucial as we extend our work to $3D$ robotics, applying method 2 is considered better.

VII. Related Work

Early this year in ICRA'09, a workshop was organized with invited talks concerning recent advances in applying formal methods in robotics[5]. We categorize these works into two groups. The first group focuses on using logic or stochastic based approach for verification or synthesis, for example [3], [9], [10]. The second group focuses on the construction methodology of the robotic systems such that the system can be analyzable or verifiable, for example [8], [9], but no signs concerning how verification is achieved and how complexities are tamed.

Our work differs from the above researches mainly with the following fact that we focus on the verification of shape-adjustable robots, while the above works are in general investigating shape-nonadjustable robots. We discuss main challenges for verification (lack of mathematical functions, efficient state space recording for collision detection) due to this difference, and propose our solutions for verification engines.

In automatic control, formal methods have been applied on PLC programs, where they are translated into verification models representable by various model checkers. A survey paper [2] summarizes existing work. As the verification focus is on the PLC program itself, it has only been applied on very abstract levels within the context of control sequencing, as the environment is not part of the verification model

VIII. Conclusion

This report investigates the problem of modeling and verification of shape adjustable robots with controllable complexities. We summarize our contributions as follows:

[5]http://web.mae.cornell.edu/hadaskg/ICRA09/index.html

We indicate precisely discrepancies between robotics and verification, where a mediated approach stepping from both sides is required to perform verification in robotics.

In the context of shape-adjustable robots, we mention ingredients for integer-based verification engines to include, and propose theoretical criteria with experiments on model construction techniques; the resulting model is verifiable within the current limit of verification engine.

We illustrate the process to perform verification on a simulation robot system; we hope that the process can be easily adapted by others for the introduction of verification in their projects. Our analogy between the simulation environment and the verification environment offers such guidelines.

We propose a few directions for near-term investigations. First, we need user-friendly tools to convert specifications in natural languages to automata concerning object movements. Also, an automatic generation of collision specifications is required. Second, it is worth investigating how existing algorithms can be tailored into our proposed framework. Third, we expect to develop a transformation scheme to convert simulation models to verification models automatically.

Acknowledgements

We thank Máté Kovács, Markus Rickert, and Michael Kasseker for their opinions concerning the presentation, and Markus Weissmann for information concerning PLC verification. The problem was preliminarily formed during the discussion of safety problems of Starmac projects with Gabriel Hoffmann, Stanford University.

References

[1] T. Ball and S.K. Rajamani. The SLAM project: debugging system software via static analysis. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 1–3. ACM New York, NY, USA, 2002.

[2] M. Bani Younis and G. Frey. Formalization of Existing PLC Programs: A Survey. In *IEEE/IMACS Multiconference on Computational Engineering in Systems Applications (CESA)*, number S2-R-00-0239, 2003.

[3] J.M.B. Braman, R.M. Murray, and D.A. Wagner. Safety verification of a fault tolerant reconfigurable autonomous goal-based robotic control system. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007. IROS 2007*, pages 853–858, 2007.

[4] C. Cheng, T. Fristoe, and E.A. Lee. Applied verification: The ptolemy approach. Technical report, UCB/EECS-2008-41, UC Berkeley.

[5] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model checking*. MIT Press Cambridge, MA, USA, 1999.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms*. Cambridge, MA: MIT Press, 2001.

[7] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Mine, D. Monniaux, and X. Rival. The ASTREE analyzer. *Lecture Notes in Computer Science*, 3444:21–30, 2005.

[8] M. Kloetzer and C. Belta. Hierarchical abstractions for robotic swarms. In *IEEE International Conference on Robotics and Automation (ICRA'06)*, 2006.

[9] M. Kloetzer and C. Belta. Temporal Logic Planning and Control of Robotic Swarms by Hierarchical Abstractions. *IEEE Transactions on Robotics*, 23(2):320–330, 2007.

[10] H. Kress-Gazit and G.J. Pappas. Automatically synthesizing a planning and control subsystem for the DARPA urban challenge. In *IEEE International Conference on Automation Science and Engineering, 2008. CASE 2008*, pages 766–771, 2008.

[11] K.G. Larsen, P. Pettersson, and W. Yi. Uppaal in a Nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.

[12] K. McMillan. Cadence SMV. *Cadence Berkeley Labs, CA*.