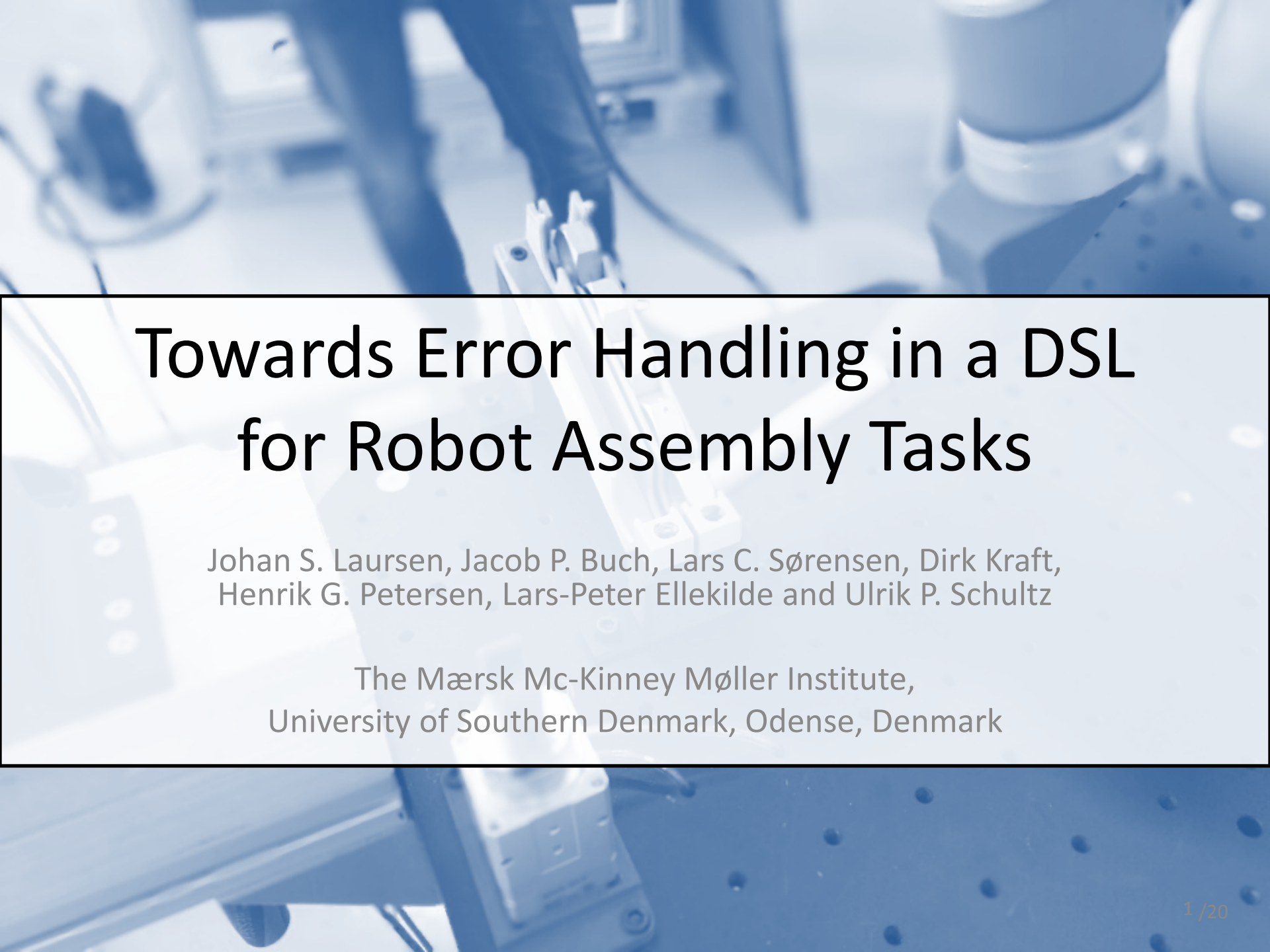


Towards Error Handling in a DSL for Robot Assembly Tasks

Johan S. Laursen, Jacob P. Buch, Lars C. Sørensen, Dirk Kraft,
Henrik G. Petersen, Lars-Peter Ellekilde and Ulrik P. Schultz

The Mærsk Mc-Kinney Møller Institute,
University of Southern Denmark, Odense, Denmark

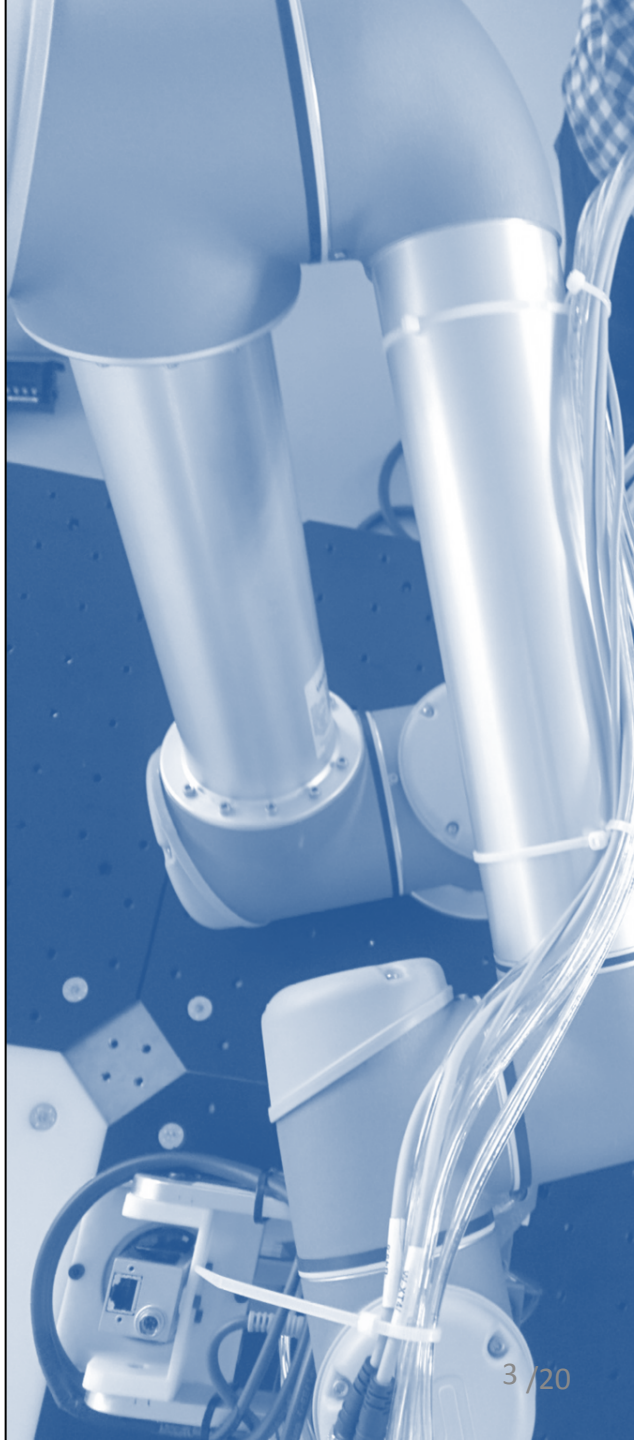
Platform and context

- Bringing robotics to small-sized production is highly important [1]
- The platform and project
  - Robot arms
  - Assembly tasks
  - Small-sized batch production
- Essentials
  - Easy programming
  - Fast setup of assemblies
  - Allows quick changeovers
  - Easy adjustments and reconfigurations.

[1] M. Haegele, T. Skordas, S. Sagert, R. Bischoff, T. Brogårdh, and M. Dresselhaus, "White paper-industrial robot automation", 2005.

The traditional approach within assembly

- Based on repeatability and the high precision of robots
- Relies on an accurate real world model
- Deterministic behaviour is ensured by
  - Customised components
  - Fixtures
  - Sensors



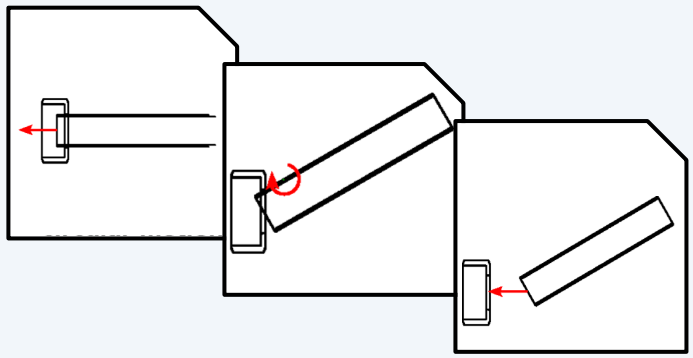
The problem and proposed solution

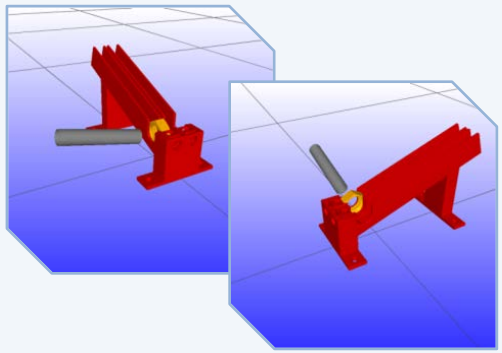
- **Problem:**
  Traditional assembly is too time consuming for small-sized batch productions

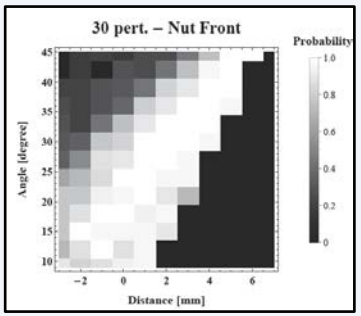
- **Solution:**
  Loosen deterministic requirements through software.
  - A probabilistic approach
  - Active handling of uncertainties

The underlying approach

- A probabilistic approach for active handling of uncertainties
- An action library where actions are parameterized
- Simulation facilitates the learning of uncertainty-tolerant actions through an optimal choice of parameters

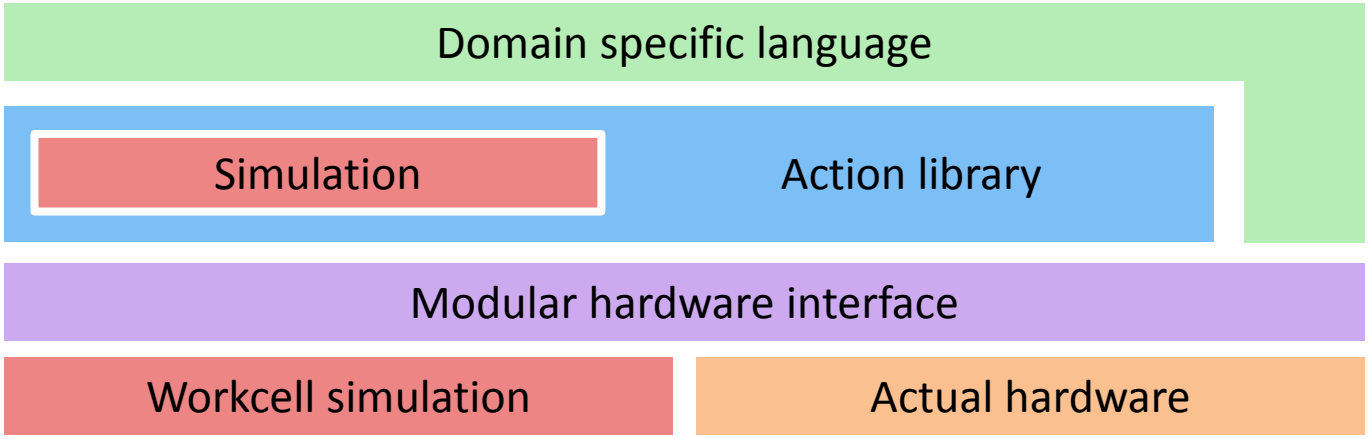






The software architecture

- A four-layer system architecture is used as the foundation of the software
- Takes inspiration from the SoftRobot project[2]



[]2 A. Angerer, A. Hoffmann, A. Schierl, M. Vistein, and W. Reif,
"Robotics API: Object-Oriented Software Development for Industrial Robots,"
Journal of Software Engineering of Robotics, vol. 4, pp. 1–22, May 2013.

Programming the framework

- Allows the use of both high level actions and hardware-near calls
- Styled as a traditional robot scripting language
- The framework and original DSL is presented in [3]

```
IOoperations().
  manipulation("gripper_open").
    setLow().bit(0).sleep(0.5).
  manipulation("gripper_close").
    setHigh().bit(0).sleep(0.5);

JointConfiguration
  startPosition = {3.425, -1.0...},
  handlePosition = {3.379, -1.2...};

sequence("peg_in_hole_recovery").
  move().to(startPosition, handlePosition).
  io("gripper_open").
  ...
  move().to(startPosition);
```

[3] J. P. Buch, J. S. Laursen, L. C. Sørensen, L.-P. Ellekilde, D. Kraft, U. P. Schultz, and H. G. Petersen, "Applying simulation and a domainspecific language for an adaptive action library," in Proceedings of International Conference of SIMPAR 2014, October 2014.

Extending the concept of
uncertainty handling to the way we program assembly tasks.

THIS CONTRIBUTION

Complexity in assembly

- As the complexity of assembly increases so does the likelihood of errors

- Our approach:
  - It is assumed that errors are inevitable and will appear at some point during the assembly
  - Instead of avoiding errors, the errors are to be managed and rectified.

Solution

- Integrating error management into the language

- Defining an error-aware move instruction suitable for operation sequences where an error is likely to occur

User-defined error management

- Information embedded in errors
  - Action to take upon recovery
  - Severity of error
  - How to proceed after recovery
- Resembles the try/catch-construct from ordinary programming languages and the approach taken in [4]

```
Error( "peg_not_inserted" )
  .recovery_using_sequence( "peg_in_hole_recovery" )
  .recovery_starts_after( currentAction )
  .postrecovery_behaviour( returnToSequence );
```

[4] R. Simmons and D. Apfelbaum,
"A task description language for robot control,"
in in Proceedings of the Conference on Intelligent Robots and Systems (IROS), 1998.

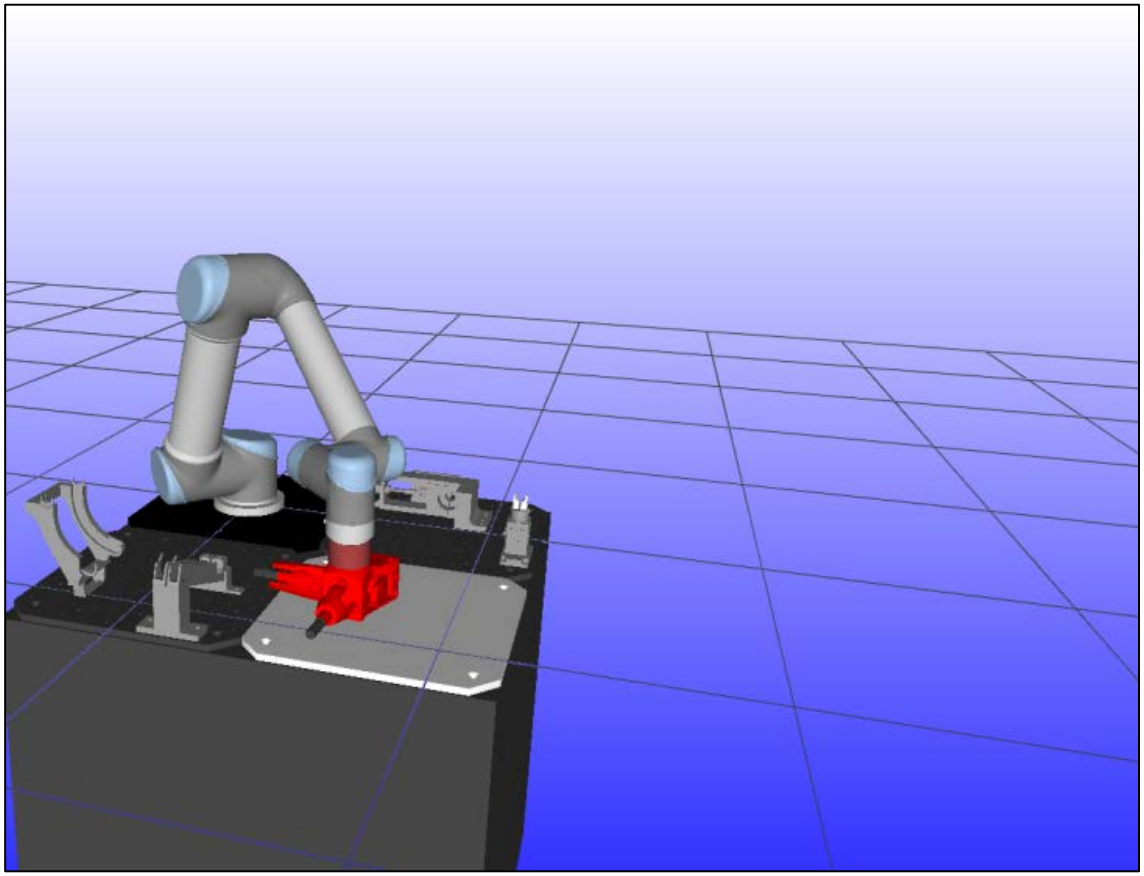
Error-aware move commands

- Allows specifications of
  - Move specifications
  - Success criteria
  - Behaviour on success
  - Behaviour on failure
- Utilize simple measurements
  - Position and distance measurements
  - Force measurements
- Build-in support structures:
  - Query max force
  - Intuitive frame mapping
  - Repeat with perturbations
  - … and more

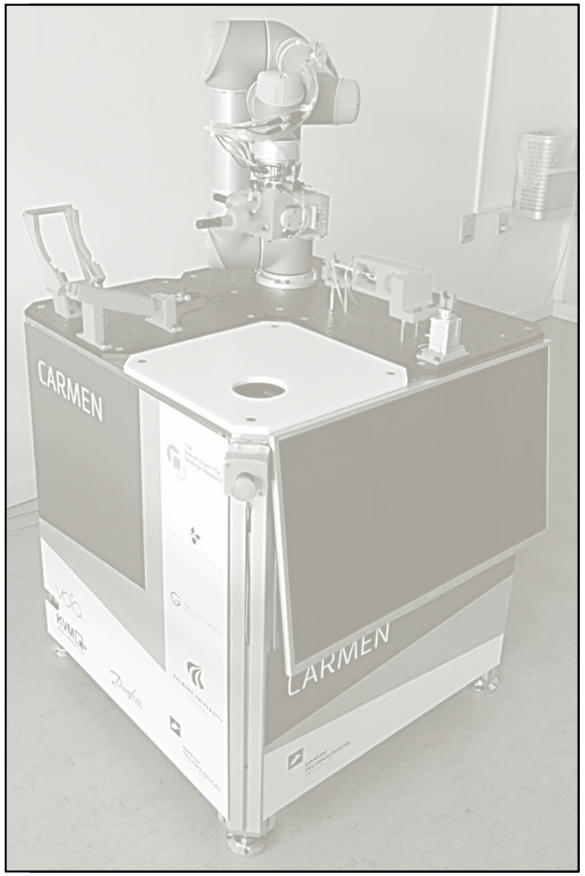
```
advanced_move( "insert_peg" ).
specifications()
     .distance(0.30, direction::forward, frame::tcp)
     .stop_ifForcesExceed(5)
     .setting(speed::slow).
evaluation()
     .distanceCovered(crocodile::moreThen, 0.20).
behaviour_on_success()
     .returnToInitialPosition().
behaviour_on_failure()
     .returnToInitialPosition()
     .repeatMoveWithPerturbations(3)
     .throwError( "peg_not_inserted" );
```

Implementing and testing

Simulation



Real world



Design considerations

– keeping the DSL internal in C++

- The DSL is developed alongside the underlying software model
- Keeping it internal ensure it remains tightly up-to-date with the framework
- Developers are already comfortable working in C++ and C++-environments
- The ability to experiment with the DSL and features more easily.

Automated recovery of probabilistic errors through

REVERSE EXECUTION

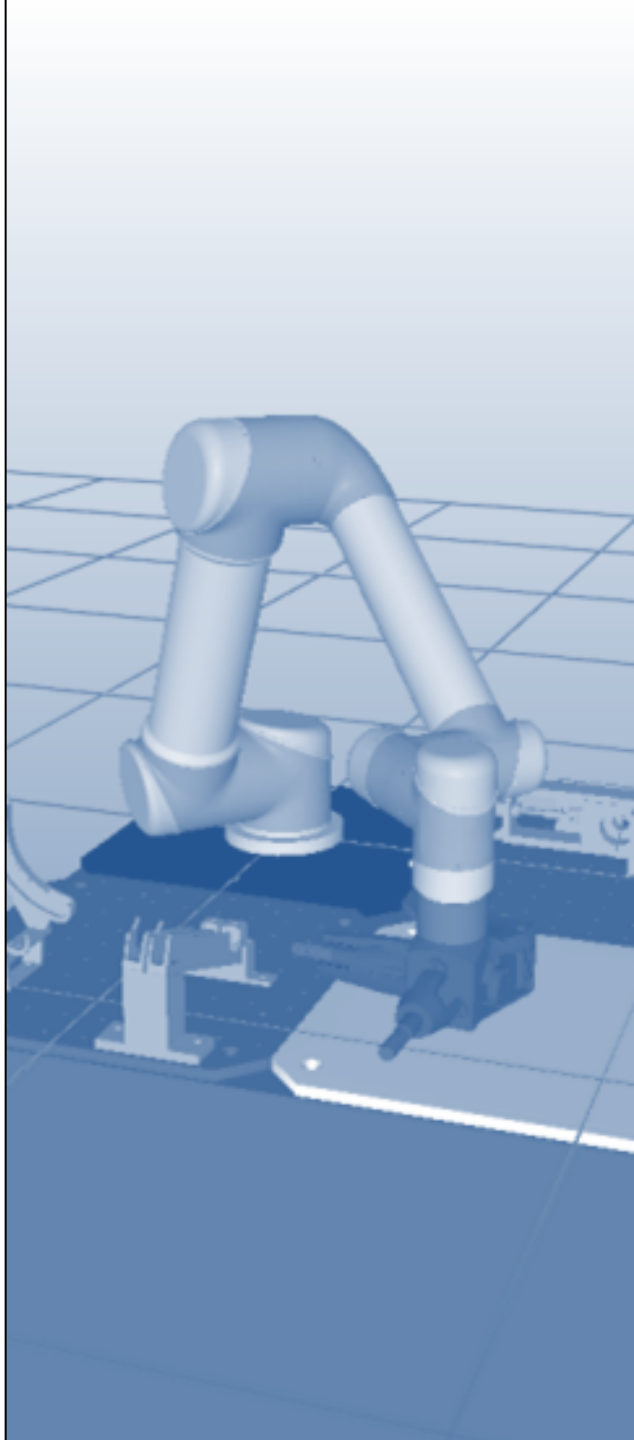
Motivation and context

- **Errors can be rectified by repeating the assembly process**

- Errors have a probabilistic nature
  - Precision of movements
  - Pose uncertainties
  - Sensor readings

- An active approach
  - Adding perturbations
  - Change in parameters to make execution more conservative

- Our approach:
  Automatic repetition through reverse execution of programs.



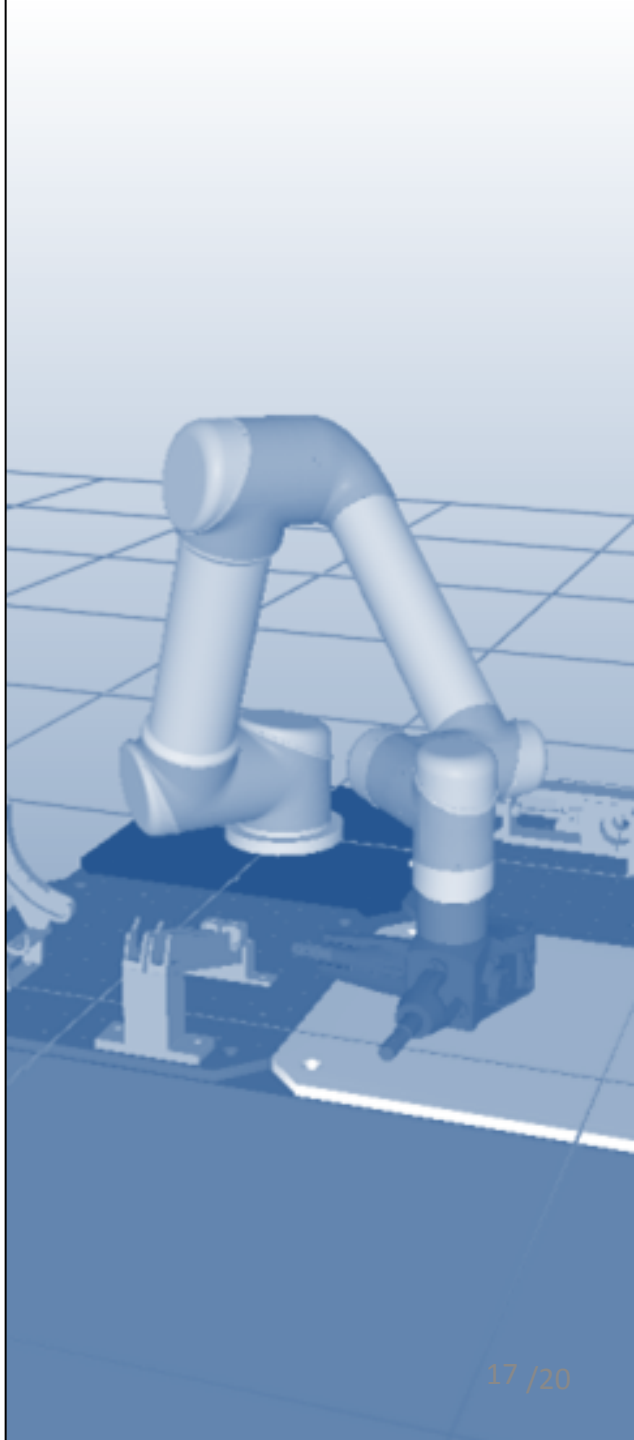
The problem of reverse execution

- Reverse execution in commercial controllers
  - Debugging and programming feature
  - Executes commands in reverse order
  - No reverse counterpart of commands
  - Can’t handle advance sequencing structures

Problems with reverse execution:

- Not all parts of assembly tasks are reversible.
  - Two objects have been clicked together
  - Not be possible to return an grasped object

- Reverse counterparts to primitives change
  - On ↔ Off
  - Enable**(** do something **)**
    ↔ Enable**(** undo something **)**



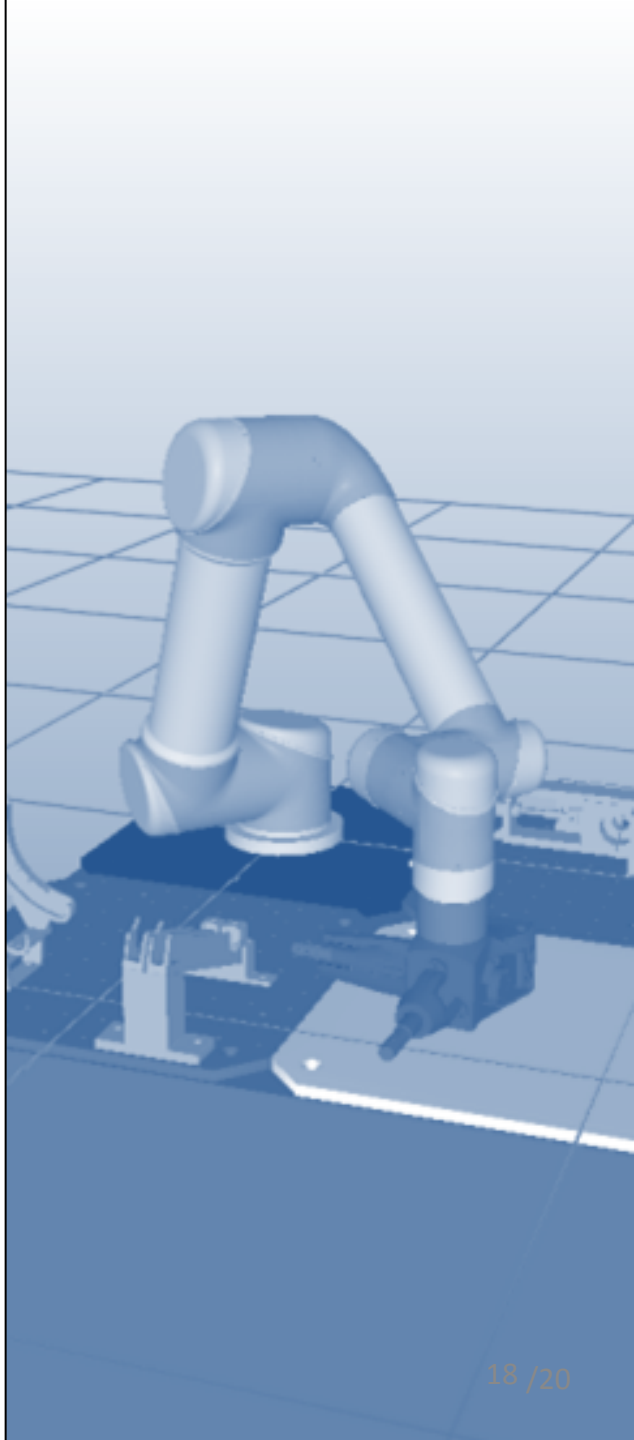
Reversibility and sequencing constructs

Reversibility

- Categories
  - Always reversible
  - Reversible after forward execution
  - Never reversible
- Default reverse counter parts
  - Ambiguity makes it difficult
  - Kinematic reversible

Sequencing and ordering

- Common structures in robot assembly:
  - Sequences
  - Hierarchical
- Advanced structures
  - Branching and forking
  - Inspiration from reversible computing [4]
  - Approaches not viable for computer programs.



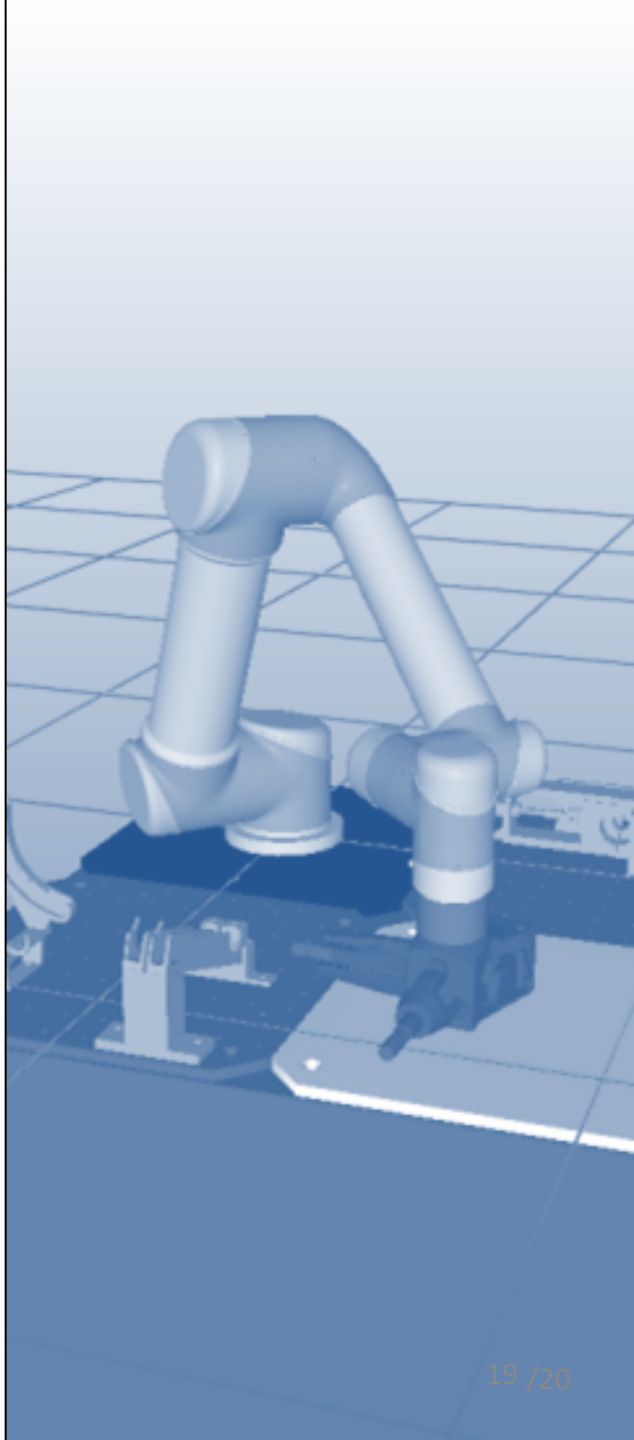
[4] T. Yokoyama, H. B. Axelsen, and R. Gl¨uck,
"Principles of a reversible programming language,"
in Proceedings of the 5th conference on Computing frontiers. ACM, 2008, pp. 43–54.

Language development plans and goals

- Programs capable of both forward and backwards execution can be made by combining the constructs.

- The Language instruction set
  - Flag sequences as non-reversible
  - Indicate sequence parts to be skipped
  - Changes to parameters, default options or modification of execution order

- Language design goals
  - Non-intrusive
  - Focus on forward execution
  - Simple and good default options



In conclusion

- Deterministic requirements in assembly can be loosened through software with a probabilistic approach and active handling of uncertainties
- Uncertainty handling can be integrated through error management and error-aware move functions
- A concept for automated recovery of probabilistic errors through reverse execution was presented along with some of the associated challenges