A systems engineering analysis of robot motion for Team Delft's APC winner 2016

Carlos Hernandez and Mukunda Bharatheesha

Abstract In this chapter we take a systems engineering stand to perform a post-mortem analysis of the design of the robot motion subsystem in Team Delft's robot winner of the Amazon Picking Challenge 2016, understanding the benefits and limitations of the motion planning approach taken. We use an analysis framework based on Model-Based Systems Engineering with the ISE&PPOOA methodology, and a novel model of levels of robot automation. The functional approach of ISE&PPOOA helps us understand how the design decisions in the architecture of the solution impact the performance and quality attributes in capabilities required for the competition. The levels of robot automation help analyze the fundamental properties of the control schemas applied at different levels of the control architecture when handling uncertainty.

1 Introduction

The Amazon Picking Challenge (APC) 2016 involved two manipulation tasks, to *pick* and *stow* products from an Amazon shelf, which addressed some of the challenges for reliable picking in a real warehouse, such as diversity of products, cluttered spaces, uncertain environment conditions or full autonomy. Team Delft robot won both competitions with a solution based on 3D cameras for deep-learning based item detection, a planning-based solution for the motions of an industrial manipulator, a custom gripper, and the integration of off-the-self software and application-specific components with the Robot Operating System (ROS.)

Carlos Hernandez
Delft University of Technology, Mekelweg 2, 2628 CD Delft, The Netherlands, e-mail: c.h.corbato@tudelft.nl

Mukunda Bharatheesha
Heggeranklaan 1, 5643 BP, Eindhoven, The Netherlands. e-mail: mukunda.bharatheesha@nobleo.nl

System-level integration is acknowledged as one of the key issues in the development of autonomous robotic systems [8, 1, 4, 5], as in other modern complex systems. Therefore, in this chapter we take a systems engineering stand to perform a post-mortem analysis of Team Delft's solution for robot motion, understanding the benefits and limitations of the planning approach taken. We focus on the picking task for simplicity, because it includes all the challenges and proved to be more demanding[1], although it is important to note that the designed solution solved both tasks. For a detailed analysis of the overall solution, we refer the reader to [4]. Our analysis framework is presented in Section 2. Section 3 analyzes the design of the motion subsystem, and its resulting properties and performance. Finally, Section 4 discusses general issues on the design of complex autonomous robots, looking into future challenges.

2 A framework to analyze an autonomous robot design

The development of modern robotic systems, such as those in the APC, require the *integration* of diverse technologies, e.g. 3D cameras, deep neural networks, planning, robot arms, and involve different disciplines, such as mechanical design, machine learning, control and software engineering. Model-based Systems Engineering provides methods and tools to consistently integrate the information, assumptions and decisions through the development of the system, from high-level capabilities operational needs, to detailed design decisions. the appropriate framework for the design and implementation of autonomous robots.

Besides the integration challenge, to address the capabilities needed, the selection of the control schemas at different levels of the solution architecture is based on assumptions about the structure of the task and the environment, and the associated uncertainty. For this reason, our framework also includes model of levels of robot automation[4]. This model categorizes the different control schemas, based on the degree to which they address uncertainty with prior assumptions or runtime data, and the resulting quality attributes (QAs) in the robot capabilities they address.

2.1 Functional analysis under the ISE&PPOOA method

Autonomous robots are complex systems that require powerful systems engineering methods to allow teams[2] to build solutions that meet the demands of industry and

[1] The structure of the problem resulting for the relative perpendicularity of gravity and the shelf's bins opening, together with the bins form factor with large depth/opening ratio proved still more of a challenge for robotic manipulation than the picking from the tote required for the stow task.

[2] It is important to note that we are not directly referring here to the robots participating in the ARC, for example. These are research prototypes whose development is only concerned with functional requirements and some non-functional requirements, such as speed or error recovery, but only to

society, especially QAs such as reliability and safety, and associated non-functional requirements (NFRs).

The ISE&PPOOA[3] process is an integrated systems and software engineering methodology that adopts the functional paradigm for model-based systems engineering. The systems engineering subprocess in ISE&PPOOA proceeds as follows:

1. Identify the operational scenarios, obtaining the needs, that are the main input for the

2.a. specification of the capabilities and functional requirements of the system, and in parallel the

2.b. specification of the relevant QAs and associated system NFRs. From this, the solution is created by iterative refining of

3. the functional architecture, which is obtained by transforming the previous level functional requirements into a functional hierarchy, and the
4. physical architecture, obtained by the allocation of functions to the building blocks of the solution, refined by design heuristics and patterns to address specific QAs.

In this chapter, we use the basic concepts in ISE&PPOOA to analyze the design of the motion subsystem in Team Delft's robot. The functional approach of ISE allows to abstracts the architectural properties of the solution, independently of particular implementation details. Additionally, ISE&PPOOA explicitly incorporates QAs through the application of heuristics and design patterns to define the physical architecture of the solution. This helps understand how the assumptions and the design decisions impacted the architecture of the solution and the performance obtained in the capabilities required for the robotic challenge.

2.2 Levels of robot automation

The control architecture of an autonomous robot is designed to automate its runtime actions despite a certain level of uncertainty, by exploiting knowledge and assumptions about the structure of the task and the environment. The decisions made to select specific control schemas or patterns embed assumptions that critically determine the resulting quality attributes for the robot's capabilities (e.g. reliability of the grasp, speed of pick&place, etc). We have elaborated a characterization of control schemas in *levels of robot automation* [4]. This characterization is aimed at supporting design decisions on *how* to automate each function in the system, i.e. which

the extent that they directly relate to the scoring in the competition. Actually, to the knowledge of the authors, none of the teams in the ARC competition followed an MBSE approach. In the case of Team Delft, the leading author partially followed the ISE&PPOOA methodology during the earlier stages of requirement analysis and conceptual design. However, to apply any MBSE methodology for the complete development of the system, the team needs to be familiar with it, and this was not the case.

[3] http://www.ppooa.com.es

control pattern to use, considering the uncertainty level and the requirements on the QAs. Previous models, such as the on in [6], focused on *which* function to automate, and which ones to allocate to a human operator.

The base criteria to differentiate our levels of robot automation is time: when the information that decides the action is produced (at design-time or at run-time), and how it is used to generate the action (periodically at a certain frequency in a loop with runtime data, or in an open-loop manner based on strategies pre-defined at design-time).

Level 0 corresponds to classic open-loop automation. A complete and static model of the task and the environment is assumed and no runtime data is used. This allows to specify a fixed sequence of motions and actions during development, which are blindly executed by the robot to perform the task.

Level 1 includes sensing information at runtime to perform a binary verification of the assumptions about the state of the system or its environment. This is the case of a sensor to detect a correct grasp of an object, or if it is dropped. Note that, no matter whether the sensory data comes a posteriori or prior to the action, the later is specified at design time, and only a discrete selection is performed at runtime. Level 1 mechanism overcomes some of the limitations of level 0, e.g. addressing uncertainty through basic error handling mechanisms. However, a complete discrete, predefined model of the environment and task is still assumed.

Level 2 corresponds to the so called sense-plan-act paradigm. A prior model of the environment and the task is updated with sensory data at runtime to compute the control action. This level accounts for more complex environments, through perception capabilities that estimate the operational state an account for limited, static uncertainties, and planning to produce actions sequences adapted to the perceived situation. Still a perfect motor model is assumed to execute the planned actions, which results in well-known limitations.

Level 3 is the classic feedback control, in which the robot actions continuously adapt to the perceived state through closed-loop control. This level can cope with higher levels of uncertainty in the variables considered at development-time for the design of the control system, and can also address bounded unknown perturbances.

Level 4 considers more advanced control schemas including *prediction* to compute optimal control actions, considering not only current runtime sensory data and state estimation, but future states too. Even these control schemas still assume a prior model of the robotic system and its environment, though in this case it can be a complex model including dynamics.

Level 5 [4] Control architectures in this level are no longer bounded by the models at design-time, but incorporate mechanisms to learn and improve upon those models using runtime data, for example exploiting machine learning techniques, and adapt their control policies or learn new ones for new tasks, for example with reinforcement learning.

[4] This level of robot automation has been included, compared to our first model in [4], to account for robotic systems that can adapt and learn.

It is important to note that the control architecture of a complex autonomous robot typically leverages control solutions for functions at different levels in the functional hierarchy. For each element in the architecture a solution is chosen from the appropriate robot automation level, based on a trade-off analysis between the QAs (e.g. reliability, impacted by the level of uncertainty), function performance, e.g. speed, and cost, e.g. in terms of resources and development effort, heavily impacted by the use of off-the- shelf. Following we examine the design decisions in Team Delft's motion subsystem, using the framework provided by ISE&PPOOA and the levels of robot automation to analyze architected solution in terms of the control patterns selected and the resulting measurement for its QAs.

3 Motion Subsystem Design

Pick and place systems require a generic set of robotic capabilities, regardless of the specific application, namely: locate and identify objects, attach or grasp individual objects, and manipulate or move them [5]. In this section, we analyze Team Delft's solution for the motion capabilities following the functional approach presented in Sect. 2.1, which allows to abstract general (functional) design considerations from the specific solution adopted.

3.1 Motion Requirements

The main operational scenarios for the motion subsystem in Team Delft solution were: 1) move the camera attached to the robot's end-effector to obtain images of the bin for the next target item, and 2) plan and execute the motions to retrieve the target item from the bin and place it in the tote. The operational needs of these two scenarios were transformed into the detailed functional requirements, NFRs and design constraints in Table 1 and assumptions in Table 2, some of which come from the design of the overall system, the competition rules and desired quality attributes in the system.

3.2 Robot manipulator

A typical decision in manipulation applications is the selection of the robot. It is a core building block in the physical architecture of such systems that is allocated the motion functionality.

Table 1 Requirements for the motion subsystem in Team Delft's robot. These are more refined requirements based on the overall system analysis in [4]

ID	Requirement
FReq.1.1[ab]	The system shall move the camera in the robot end-effector to an appropriate pose in front of the target bin avoiding collisions.
Related:	From ON.1 and QA harmless.
FReq.2.1	The system shall generate planned motions to reach with the gripper all locations inside the shelf's bins that are relevant for grasping items.
PReq.5.2	The robot shall execute the trajectory to reach a target location as fast as possible.
Related:	Derived from speed req., related to QA reliability, as this maximizes the opportunities to pick items in the allotted time.
FReq.5.1	The system shall find a collision-free path to grasp the product, and a retreat path to retrieve the item from the shelf.
FReq.3	The system shall achieve and hold a firm grasp on all different products.
Cons. 1	The robot shall move following a velocity and acceleration profile that guarantees a reliable grasp when holding an item.
Related:	From QAs reliability and harmless, to prevent dropping a grasped item when moving.
Cons. 2	The robotic system shall fit inside an area of $2x2$ m^2 separated 20cm from the front of the shelf and the start of the task.
Related:	From the competition rules.

[a] FR: functional requirement. NFR: non-functional requirement. Cons.: Constraint.
[b] The numbering is consistent to the one reported in [4].

Table 2 Assumptions for the motion subsystem, some are requirements for other subsystems in Team Delft's robot.

Assumption 1 (from Req.4)	a suitable grasp surface is always directly accessible from the bin opening that allows to grasp and retreat holding the product, and no complex manipulation inside the bin or the tote is needed. This way the level 2 assumption of environment invariance holds.
Assumption 2	Not all the locations inside the bins are relevant for grasping. Due to gravity and based on the dimensions of the objects, no grasp candidates are expected in the higher part of the bins.
Assumption 3	Environment out of the shelf is known and static.
Assumption 4	Environment inside the bin unknown and static during grasp operation. This required that the collision scene of the bin needs to be updated every time a grasp attempt is executed.

A solution based on an off-the-shelf industrial manipulator and the MoveIt! library[5] was deemed the most feasible and desirable, given the resources available and also the skills in the team, very familiar with motion planning in industrial manipulators[6].

A configuration with an off-the-shelf robot manipulator and an additional axis for enhanced reachability would render the required precision and speed. Multiple configurations were considered (figures 1a and 1b), constrained by the robot cell dimensions Cons. 2, and subsequently evaluated according to the requirement for reachability (FReq.3) using the MoveIt! workspace analysis tools[7].

The configuration shown in figure 1a was finally chosen, as the reachability analysis results ensured that the gripper could reach all bins with adequate maneuverability. An important aspect that can be observed in the figure is that some of the deep corners of the bins have no markers, indicating that those regions are not accessible with this system configuration. This limitation was compensated by slightly increasing the length of the suction tool (represented by the solid gray block in the robot's end-effector in figure 1a).

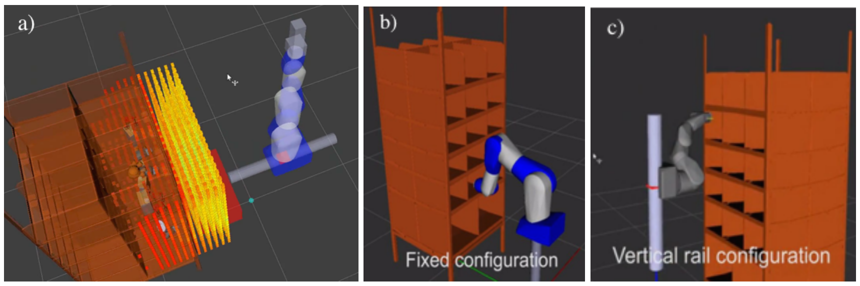

Fig. 1a Reachability Analysis results of the selected robotic system configuration. The color of the marker in the figure indicates the number of IK solutions available to reach the corresponding cartesian location in the robot cell. Red markers are for locations that have very few IK solutions.

Fig. 1b and **1c** Other configurations analyzed.

[5] The documentation for the various APIs referred in this text can be accessed via: http://moveit.ros.org/code-api/

[6] At an early stage a cartesian robot was discarded because it required extensive and specific mechanical and software development for its control. Its potential benefits in speed and simplicity of the kinematics were not as immediate in the shelf configuration of APC as in 2017 edition, when such a cartesian manipulator system won[5] by using that edition's possibility to have a custom horizontal storage system, which made the task decomposable into a series of straight motions by exploiting the cartesian structure in the environment

3.3 Motion Software module design

To fulfill the motion requirements of both the picking and stowing tasks, the motion module[7] was built on two fundamental motion primitives, namely, coarse motions and fine motions, that relied on the assumptions in Table 2. Coarse motions were offline generated trajectories between pre-defined start and goal positions in the robotic cell. Fine motions involved online (cartesian) path planning for performing object manipulation in the bins or the tote. In the following sections, these two primitives will be explained in further detail.

3.4 Offline Coarse Motions

Assumption 3 of a static environment allowed us to select an efficient ‘level 0’ solution for the robot motions outside of the shelf and the tote, using predefined joint trajectories, or *coarse motions*, between configurations associated to key positions in the workspace.

These configurations, named Master Pose Descriptors, are robot joint states set at appropriate values in front of each bin of the shelf for the functions FReq.1.2 to take a bin image and FReq.5.1.1 to approach target object. Our choice of having the camera mounted on the manipulation tool entailed that we have two master pose descriptors per bin, namely, the *camera* and the *bin*[8] master poses[9]. Collision checking using 3D model (URDF) of the elements in the robot cell (enlarged to account for known uncertainties, such as 3 cm deviations in the positioning of the shelf) ensured safe motions (FReq.5.1).

Subsequently, a *trajectory database* with around 250 trajectories between different master poses was generated using the RRT-Connect randomized path planner via MoveIt!. Other planner options were analyzed, such as RRT* and the OMPL[10] implementation KPIECE, but RRT-Connect outperformed both of them for our planning problem (characterized by the robot kinematics chosen and the geometric structure of the shelf and tote). The main reason for RRT-Connect outperforming was that it maximally exploited the benefit of the linear joint (rail), because it explores solutions starting from both the start and end nodes, using Euclidean distance in configuration space.

[7] The open source software repository hosting the code for the motion module and the rest of the ARC 2016 software is available at:
https://github.com/warehouse-picking-automation-challenges/team_delft

[8] starting configuration to approach target object

[9] Similar master pose descriptors were also defined for the tote drop-off locations.

[10] MoveIt!, the framework chosen to implement our motion module, uses OMPL for planning.

3.5 Grasping and manipulation: fine motions

From a motion perspective, the grasp strategy for all objects consisted of a combination of linear segments. We call these segments as *Approach*, *Contact*, *Lift* and *Retreat*. The segment names are indicative of the motions that they are meant for. This solution maximally exploited the geometry of the bins, and Assumption 2.

The idea of fine motions is an adaptation of the approach in the standard pick and place pipeline of MoveIt!, where cartesian path planning is used during the pre-grasp approach and the post-grasp retreat stage of the pipeline. We removed the standard filter stage in the MoveIt! pick and place pipeline consisting on evaluation of the reachable and valid poses. To optimize the end-to-end time performance of the planning step in the overall application flow, this filtering was divided in multiple steps.

The first one is performed during the function grasp synthesis, where impossible grasps are eliminated heuristically, exploiting knowledge of the products and their graspability given their relative pose inside the bin. Then second filtering is performed by the motion subsystem, by checking the feasibility of the manipulation plan associated with the grasping candidate. The feasibility is determined by avoiding collisions, and obtaining a feasible joint trajectory (Inverse Kinematics solution).

3.5.1 Collision avoidance

During grasping and posterior manipulation to retrieve the object from the bin, collisions need to be avoided to prevent damaging items or the shelf and to lose a grasped item. For collision checking the following information is used: runtime data, consisting of the occupancy octomap from the scanned point cloud, from which the points corresponding to the target object are removed to allow for contact in the case of suction grasping, and additional prior information, consisting of the 3D model of the shelf in URDF, and a collision model for the specific bin (mesh), which position is nevertheless computed online in the perception pipeline.

3.5.2 Generation of the complete grasp plan

Departing from the pose of the target item, the grasp synthesizer module generates heuristically a set of key waypoints for the start and end of each linear segment. These waypoints are input to the motion module, which obtains the manipulation plan proceeding as follows:

1. The key waypoints are sequentially checked for collision and feasibility [11]. If any one of them is in collision, the corresponding grasp pose is discarded and an alternative one is requested.

[11] using the `getPositionIK` service call

2. Once all the key waypoints are collision free, each linear motion segment is computed using cartesian planning[12] with collision checking enabled. Similar action is taken if any of these segments are in collision.
3. If all the linear motion segments are collision free, a final planning in joint space is done, using RRT-Connect because of its fast solution times[13]. This is required because, the final joint configuration at the end of the Retreat segment resulting from the cartesian planning need not necessarily match the starting joint configuration in front of the bin or tote from where the coarse motions start. This is natural because of the redundancy in the system's degrees of freedom. The final planning ensures that such a mismatch does not exist which would otherwise lead to a motion safety violation [14]. To obtain last joint configuration in the cartesian trajectory, the inverse kinematics solver is configured to minimize the steady state error[15], which yields the closes joint solution to the configuration goal, minimizing configuration changes.

These steps ensure that all the desired motion segments for manipulating the object of interest are generated as required and are collision free. These segments were then stitched together before being executed on the robot, as explained in the following section.

3.6 *Trajectory execution*

After obtaining a feasible grasp plan (complete joint trajectory to retrieve the target item), it had to be executed. To address requirements PReq.3.2 and Cons.1, two solutions were designed that prepared the trajectory for execution [16]: trajectory stitching and I/O synchronization.

3.6.1 Trajectory Stitching

The motion stitching module accepts all the motion segments that are generated as explained in the previous section, plus the coarse motion trajectories from the corresponding bin to the tote drop-off location. The stitching process combines the joint state configurations from each segment into one single motion plan and time parameterizes[17] it so that it results in a executable trajectory for the robot.

[12] `computeCartesianPath` in the MoveGroup API.

[13] `plan` the MoveGroup API.

[14] A motion safety violation is triggered whenever the starting configuration of the robot does not match the starting configuration in the trajectory that is about to be executed.

[15] *distance* setting in the Trac-IK solver used.

[16] The time parameterized trajectory was finally executed in the robot controller through the MoveGroup API, `execute`.

[17] `computeTimeStamps` from the TrajectoryProcessing API was used for this purpose.

With respect to PReq3.2 on the end-to-end speed of the robot motions, by stitching multiple motion segments we get rid of overheads such as goal tolerance checks at the end of each segment execution. This indeed provided us quite a significant time gain while executing the motions. In relation to Cons.1 to prevent dropping items, the stitching performed object specific velocity scaling for the trajectory to adapt the motion to the product, for instance, moving at low speeds when carrying heavy objects such as the dumbbell, socks and kitchen paper roll.

3.6.2 Input/Output Synchronization

The final and a critical component of the motion module is the I/O handling, designed to ensure the end effector (suction or pinch) is actuated at the right times along the trajectory, to guarantee successful grasps in relation to Cons.1. The timely activation of the suction gripper was critical. For instance, the vacuum pump needed a couple of seconds before full suction power was realized, which meant it had to be switched on before the suction cup reached the contact point. However, turning the suction on too early could also pose problems. For example, the items with a loose plastic covering could get suctioned before the contact point was reaching, leading to either an unstable or a failed grasp (Req.3). In order to address this, a custom trajectory tracking module[18] was developed that provided continuous joint state information.

The trajectory tracking module used an event based-approach. The events corresponds to reaching the key waypoints along the trajectory, and the tracking module used a gradient descent based approach based on the distance to those points (level 3 based on runtime data from the encoders). These events were used not only to trigger the suction, but also to trigger the pressure sensor feedback, for example to evaluate the success or failure of a grasp at the end of the retreat segment. This runtime data allows to cancel the open loop execution of the trajectory if an error is detected at certain points, and take appropriate actions (level 1 mechanism), for example re-try the grasp if the pressure sensor detects no vacuum seal after retreat (i.e. the item is not attached)[19].

4 Discussion and concluding remarks

Team Delft's robot was based on an overall sense-plan-act (level 2) solution, that relied on detailed solutions for specific motion functions that ranged from level 0 to level 3, as discussed along Sec. 3 and summarized in Table 3:

It is important to note that intuitively, neglecting uncertainty, performance and simplicity (a desired property when building a system) are better for lower lev-

[18] based on the `/joint_states` topic.

[19] More details on this finite state machine approach to error handling can be found in [4]

Table 3 levels of robot automation of the different elements in the design of Team Delft's motion module.

level	design element
Level 0	Coarse motions (predefined) to move the camera to take images and to move the picked object from the shelf to the tote.
Level 1	velocity scaling of the trajectories according to the product grasped, during the stitching process.
Level 1	detect suction grasp failure using a pressure sensor in the nozzle.
Level 2	Fine motions (online cartesian planning) to grasp and retrieve the items.
Level 3	Trajectory tracking to synchronize actions along the trajectory.

els of robot automation. Although they are brittle to uncertainty, in a partially structured and known environment such as the one in APC16, simple solutions error-detection and handling at level 1 seem sufficient. However, our planning and collision-avoidance solution is based the assumptions that an unobstructed grasp is directly accessible for the target item and the environment is static during the operation, which simplifies the manipulation problem. This does not scale in general for more complex manipulation scenarios, for example inside a densely packed bin. Reactive grasping that allows and even exploits contact (level 3) is more general and robust approach. To account for handling novel objects, level 5 methods are needed, such as deep learning, to recognize them and adapt the control actions, as reported by successful entries in 2017 edition.

The functional-level analysis provided by the ISE&PPOOA methodology allows to identify useful patterns that we intuitively applied to design our solution. For example, we use a service-based blocking architecture for the motion module, which renders a deterministic behavior, e.g. preventing race conditions. This is a well known pattern for safety and reliability [3]. However, this impacted negatively on the complexity of the implementation tracking module, which required a dedicated thread of execution. An asynchronous *ROS action* based architecture is more scalable for systems that require level 3 closed-loop solutions (multiple control threads).

In relation to the synchronization of actions along the trajectories, we must note that MoveIt! allows for I/O handling to be synchronized with trajectory execution by incorporating the external devices as joints in the planning. However, this poses some problems for binary joints like our suction pump. A decision was made to create our trajectory tracking module, which creates a more reusable solution, independent of MoveIt!, that also allows to synchronize sensors in addition to actuators (modularity and integration vs. reuse an existing solution).

Other design strategies were intuitively applied to the functional architecture of our solution. The function to generate grasp candidate(s) relied on heuristics based on the product type and the current pose in the planning scene, as well as a looped interaction with the function to generate the associated waypoints, which relies on the same information. Therefore, both functions were allocated to the same module, the grasp synthesizer, in an example of the systems engineering strategy that advocates for jointly allocate functions that are closely related. This decision primed the benefits of integration, in terms of simplicity and efficiency, over the re-usability of a more modular design.

The functional architecting of our solution also presented some oversights in our functional design resulting in some limitations. For example, in our functional design we did not considered explicitly the function "move item", and the associated requirement was implicitly allocated to the gripper and the suction system. This neglect resulted in problems with the grasp when moving heavy items, only partially mitigated by the velocity scaling. If this would have been addressed explicitly earlier by the functional architecture, the the motions could have been better designed to address this, e.g. maintain alignment of the nozzle with gravity to minimize detaching torque forces.

4.1 Concluding remarks

Most teams participating in the Amazon challenge acknowledge that a development methodology focused on system-level integration and testing for early validation of design solutions that maximize their performance are key to develop such autonomous robots [5]. In this chapter, we have used the ISE&PPOOA for a system-level analysis of Team Delft's robot. This method allows to track the allocation of the different requirements in the system to the design decisions in the the architecture of the solution. Besides, we propose a series of levels of robot automation to understand the fundamental properties of the different control patterns as design solutions to address uncertainty in the robot's operational scenarios. Furthermore, the use of MBSE approaches in the development of autonomous robots, when extended with formalisms to allow capturing these design decisions and its underlying rationale, result in formal models that capture the functional relation between the architecture of the system and its mission requirements, and which can be exploited by the systems at runtime, providing for new levels of self-awareness [2].

Acknowledgements This work has received funding from the European Union's Seventh Framework Programme under grant agreement No. 609206, and Horizon 2020 research and innovation programme under grant agreement no. 732287. The authors are very grateful to all their team members, and supporting sponsors and colleagues. Special thanks to G. van der Hoorn for his help during the development of the robotic system, and to Prof. J.L. Fernandez-Sanchez for the discussions on the design of robotic applications from a systems engineering perspective.

References

1. Eppner, C., Höfer, S., Jonschkowski, R., Martín-Martín, R., Sieverling, A., Wall, V., Brock, O.: Lessons from the amazon picking challenge: Four aspects of building robotic systems. In: Robotics: Science and Systems XII (2016)

2. Hernández, C., Bermejo-Alonso, J., Sanz, R.: A self-adaptation framework based on functional knowledge for augmented autonomy in robots. Integrated Computer-Aided Engineering **25**(2), 157–172 (2018)
3. Hernandez, C., Fernandez-Sanchez, J.L.: Model-based systems engineering to design collaborative robotics applications. In: 2017 IEEE International Systems Engineering Symposium (ISSE), pp. 1–6 (2017). DOI 10.1109/SysEng.2017.8088258
4. Hernandez Corbato, C., Bharatheesha, M., van Egmond, J., Ju, J., Wisse, M.: Integrating different levels of automation: Lessons from winning the amazon robotics challenge 2016. IEEE Transactions on Industrial Informatics **PP**(99), 1–1 (2018). DOI 10.1109/TII.2018.2800744
5. Morrison, D., Tow, A.W., McTaggart, M., Smith, R., Kelly-Boxall, N., Wade-McCue, S., Erskine, J., Grinover, R., Gurman, A., Hunn, T., Lee, D., Milan, A., Pham, T., Rallos, G., Razjigaev, A., Rowntree, T., Vijay, K., Zhuang, Z., Lehnert, C., Reid, I., Corke, P., Leitner, J.: Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 7757–7764 (2018). DOI 10.1109/ICRA.2018.8463191
6. Parasuraman, R., Sheridan, T.B., Wickens, C.D.: A model for types and levels of human interaction with automation. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans **30**(3), 286–297 (2000). DOI 10.1109/3468.844354
7. Prats, M., Şucan, I., Chitta, S., Ciocarlie, M., Pooley, A.: Moveit! workspace analysis tools. URL http://moveit.ros.org/assets/pdfs/2013/icra2013tutorial/ICRATutorial-Workspace.pdf
8. Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., Mahoney, P.: Winning the darpa grand challenge. Journal of Field Robotics (2006). Accepted for publication