Evaluation of rapid development of embedded control software for cyber-physical systems with feature-based development cycles

R.P.J. (Ron) Koomen

MSC ASSIGNMENT

Committee:
dr. ir. J.F. Broenink
T.G. Broenink, MSc
dr. ir. M.E.T. Gerards

June, 2020

018RaM2020
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

The number of cyber-physical systems is constantly increasing, and the systems can do more complex tasks. Due to the increment in the complexity of those systems, the development time rises as well. To decrease the development time a good framework for rapid development is needed. In (Broenink and Broenink, 2019) such a framework is proposed. The Structured Design Method proposed should be applied to all kinds of systems to evaluate the applicability. In this thesis it will be applied to the youBot.

The feature-based design method does not advise a structure for the division of the cyber-physical system in feature-based parts. A new structure is proposed to specify the different features. The structure is based on two methods that describe the layout of an embedded system. By applying the SDM to the example in (Broenink and Broenink, 2019) results in the same set of features. Therefore this structure is also used in the case study. Specifying the features in the case study resulted in two sets of features that are merged at the end. The implementation of the complete set of features results in a few iterations for the features. When there are not a lot of iterations needed to implement the features, the order, and division of the features was done correctly according to (Broenink and Broenink, 2019).

For the evaluation of the SDM a structure is proposed for the comparison of different design methods. Usually there is a high level of human influence in the comparison. By using different metrics the human influences will be reduced. This is needed for an objective comparison. According to the comparison there is a long time spent on the design phase. Due to a long time spent in the design phase the testing and code phase are relatively short. Usually the debug time in the development process takes around 50 % of the total time. In this case it is less so the framework of the design method can be useful for the rapid development of cyber-physical systems.

The design method (Broenink and Broenink, 2019) is useful for the implementation of cyber-physical systems. The additions made with the specification of the features can be used to give the design phase a better structure. When the preparation is done well, the programming phase takes less effort. This design method can lead to a reduced development time for a cyber-physical system.

For feature projects a structure for detail levels of the model can be useful. By adding this to the SDM mistakes in the preparation phase can be reduced.

Contents

1 Introduction **1**
1.1 Problem statement . . . 1
1.2 Project Goals and approach . . . 2
1.3 Outline . . . 3

2 Background information **4**
2.1 Structured Design Method . . . 4
2.2 YouBot . . . 5

3 Analysis of the SDM **6**
3.1 Analysis of the specification for different features . . . 6
3.2 Analysis of the reflection according to the SDM . . . 9

4 Analysis of evaluation for design methods **11**
4.1 Evaluation of programming methods . . . 11
4.2 Evaluation of codebases . . . 12
4.3 Evaluation metrics . . . 12

5 Apply the design method **14**
5.1 Specification of the features of the youBot . . . 14
5.2 The model . . . 15
5.3 Implementation of the features . . . 17
5.4 Evaluation of the implementation . . . 21

6 Reflection and evaluation of the SDM **23**
6.1 Reflection of the SDM . . . 23
6.2 Evaluation of the SDM . . . 24

7 Conclusion and recommendations **27**
7.1 Conclusion . . . 27
7.2 Recommendations . . . 27

A Overview of the models **28**

Bibliography **31**

1 Introduction

Cyber-physical systems are everywhere around you. Common areas where you can find cyber-physical systems are smart household equipment, control systems for cars, medical monitoring systems, and robotic systems. An example of such a system is the robot vacuum cleaner. They are used in households a lot more the last couple of years. Most cyber-physical systems do more and more to make life simple and they become an important part of life.

The increment in the complexity of such systems also leads to longer development times. According to (Calantone and Di Benedetto, 2000) the time to market for a system is important. This makes the rapid development of systems important for companies. This is one of the reasons why rapid prototyping/development is necessary.

For the rapid development of control software for cyber-physical systems a good framework can be a very significant factor to reduce the time to market. A framework can reduce the development time by giving a structured approach to developing such systems (Lee, 2008). By reducing the time to market, the profits can increase, and the development costs can be reduced.

In the area of computer architecture, it is useful to make the common case fast. This will save the most time in the execution of a program with the least effort. According to (Greg Law, 2014) the time spent on fixing bugs in the code takes 50 % of the development time. This will be the common case in the development process. If it is possible to save time spend on the development of cyber-physical systems, then the first area to investigate is the debugging. So, the framework must also contain a way to optimize the debugging process.

1.1 Problem statement

There is a need for a structured framework to design control software for cyber-physical systems that optimize the debugging process. By decreasing the debugging time, the development time can be reduced. Every system has its characteristics and needs its approach. This makes it difficult to develop one framework that will help to develop the control software for all different kinds of devices.

The goal for developers is to get a way of working that can be used to design a large variety of cyber-physical systems. The use of a structured framework for every system will help to decrease the development time. Developers will get familiar with the method and can apply it quickly to different systems. There is a need to get a design method, that can be applied to all different systems.

One such approach is the Structured Design Method (SDM) proposed in (Broenink and Broenink, 2019) that can be applied to different kinds of cyber-physical systems. The method is using small feature-based development cycles. By using small cycles, the time spent on debugging can be reduced. The method itself has only been tested at one setup before publishing. They claim that their structured approach allows the development of embedded control software. To confirm their claim, it should be applied to multiple other cyber-physical systems. Those other cyber-physical systems should be different in structure and number of parts to confirm the working of the SDM on a large variety of systems.

The SDM consists of three parts, preparation, implementation, and reflection. In the preparation phase of the design process the different features, and detail levels of the models are specified. If the preparation is done well, it pays off for implementation. For the implementation feature-based development cycles are used as a structure for designing and testing. After the implementation there is a reflection phase. The reflection is used to learn from the mistakes that are made during the preparation or implementation. The mistakes can be prevented in the following projects. This increases the quality of the following projects and can reduce the development time.

The SDM has all requirements for a good framework for implementing cyber-physical systems. The method needs to be evaluated on multiple systems to control if the framework can be applied to all kinds of systems.

1.2 Project Goals and approach

The research goal of this project is to:

- Evaluate the feature-based development method of a cyber-physical system described in (Broenink and Broenink, 2019), by applying it on the youBot.

To reach the research goal and give a good evaluation of the SDM, the SDM must be used to implement another cyber-physical system. The first part of the research will be about the implementation. In the second part the SDM will be evaluated against other design methods. For those two parts two sets of sub-questions are formed.

1.2.1 Sub-question for applying the SDM

For the reflection of the method, the SDM must be applied to a cyber-physical system. In the preparation phase of the SDM, the cyber-physical system must be divided into separate parts, so-called features. The SDM is tested on a small system with only three features, a structured approach for the specification of the features was not needed. For larger systems a structure for the specification of the futures can be useful, but the SDM does not include in a structure to specify the features of a system. In the case study the youBot will be used as a cyber-physical system. This is a more complex and larger system, where it could be useful to use a structure for the specification of the features. The first sub-question is to specify a structure for the division of the features.

- What kind of structures can be used to split the system into feature-based parts?

An embedded system can be described by multiple structures for example (Broenink and Hilderink, 2001) (RobMoSys, 2019). Those structures will be investigated and might be combined to a more detailed structure. The combination will be evaluated against the mini-segway (Broenink and Broenink, 2019). The resulting structure will be used in the case study, to specify the different features.

The second sub-question is about the reflection part of the SDM. The reflection in the SDM can be used for the following projects, but can not be used for the current project in the way described in the SDM. To use the reflection in the current project a reflection cycle must be done more often. To reflect more often is suggested in (Morsinkhof, 2019).

- When to reflect and how is this usable for designing the other parts of the system?

Multiple reflection cycles during the development can result in a better design structure for the resulting part of the project. In the case study of this report, there will be more reflection cycles during the implementation phase.

1.2.2 Sub-questions for the evaluation of the SDM

The second part of the research goal is to evaluate the SDM. An evaluation could be used to compare the SDM with other design methods. A comparison is useful to select the best design method for a project. With the development of an embedded system and the evaluation of a method there is always a human factor involved. An evaluation method must eliminate the largest part of the human influence, to get an objective judgment. The last sub-question is about an evaluation structure for design methods.

- Which evaluation method can be used for the evaluation of a design method?

For the evaluation of the design method it is needed to compare it with other methods. To compare design methods feedback of the method is needed. But the feedback always has a level of human influence. To make a comparison possible, a structure must be designed that will eliminate human influence. Some comparisons are available for design methods (Chandra, 2015), and for codebases also comparison structures are available (Sonjara inc, 2019). Both have human influences. The methods will be compared to get to an evaluation method that minimizes human influence.

1.3 Outline

The thesis is structured as follows. In Chapter 2 the design method (Broenink and Broenink, 2019) is explained. The platform, the youBot, that will be used for the case study is described. Chapter 3 specifies the features by using multiple structures for describing embedded systems. In this Chapter the findings of (Morsinkhof, 2019) are also mentioned. In Chapter 5 the design method is used to create the control software of the youBot. In Chapter 4 the structure for the comparison of multiple design methods is explained. The structure of Chapter 3.1.3 is used to specify the different features. After the implementation of the method the evaluation according to Chapter 4 is used to evaluate the working of the design method in Chapter 6. In the end the discussion and conclusion will be noted in Chapter 7.

2 Background information

For the thesis the method described in (Broenink and Broenink, 2019) is evaluated. The method is explained in Section 2.1. For the evaluation a case study is done. The case study is executed on the KUKA youBot. The youBot is described in Section 2.2.

2.1 Structured Design Method

The design method that is evaluated is described in (Broenink and Broenink, 2019). This method is designed to give a structured framework for the rapid development of cyber-physical systems. The approach consists of two parts. A cycle for the rapid development of a set of features, and a variable-detail approach using model-driven development. The variable-detail approach is used to develop and test single features. The development method consists of two cycles, an inner and outer cycle. The outer cycle adds a feature to the system. In the inner cycle the feature is developed and tested with the variable-detail approach.

Combining the two cycles result in the following series of steps:

- Order and split the features and levels of detail as preparation
    - Design new features and tests.
    - Implement and test the new feature
        - Design a feature based on an ideal model
        - Combine the feature with a more detailed version of the rest of the system, adding more detail when needed.
        - Determine if the test passes if so, add more detail.
    - Continue to the next feature until all features are implemented.
- Evaluate and reflect on the cycle process.

By using this structure and increasing the detail level of the model in small steps, mistakes should be found quickly. Finding mistakes in an early phase makes it easier to solve them. This will reduce the total development time, and especially the time spend on debugging.

The method also consists of a reflection for the design process. After the design for the cyber-physical system is implemented, the number of iterations will be counted of a feature. If there are a lot of iterations needed to implement one feature, there is probably a mistake made in the way how the detail of the model increases. The reflection is used to increase the quality of the next projects.

The design method is already been tested by (Morsinkhof, 2019). After the implementation phase, some recommendations were made to improve the method. The recommendations will be used during the case study.

The first recommendation was to simulate multiple details at once. During the project it became clear that some detail levels do not have a significant influence. It should be more efficient by skipping the simulation of those detail levels and simulate multiple detail levels at once.

The second recommendation is to reflect on the design project more often. If there are multiple reflection cycles the mistakes in the design structure can already be solved in an early phase. The time spent on implementing other features can be reduced.

The last recommendation is that some features are not influenced by other features. So they can be tested independent of the other features. The SDM describes that the features is tested always on top of the previous one.

2.2 YouBot

The youBot is a mobile robot designed by KUKA as an open-source educational platform. One of the goals of the youBot is to bridge the gap between industrial applications and mobile manipulation research (Bischoff et al., 2011).

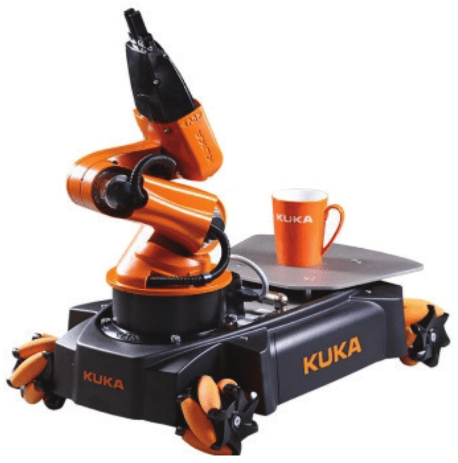

Figure 2.1: The layout of the KUKA youBot (Bischoff et al., 2011)

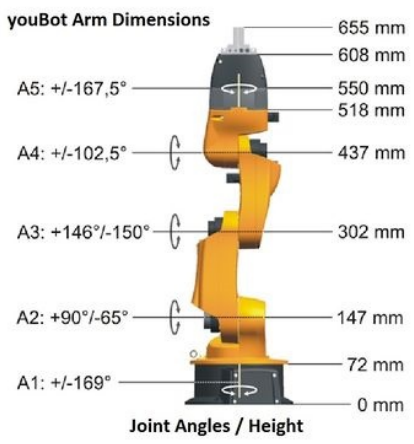

Figure 2.2: The layout and limits of the youBot arm (KUKA, 2015)

In Figure 2.1 the KUKA youBot is shown, which has the following characteristics,

- Omnidirectional mobile platform
- 5-DOF Manipulator
- Two-finger gripper
- Real-time EtherCAT communication

The omnidirectional platform has 4 wheels that can be controlled separately, this makes it possible to move the youBot in every direction. The commands that can be send to the wheels will include an angular velocity.

The joints of the 5-DOF manipulator are controlled by the TMCM-1632 controllers of Trinamic. The arm including the movement limits is shown in Figure 2.2 These BLDC motor controllers can be activated through Ethercat commands. The commands that can be given can have a setpoint where to move, a velocity, or a torque. The list of commands can be found in (TRINAMIC, 2011).

3 Analysis of the SDM

For the analysis of the SDM two different parts are used. The first part is in the preparation phase of the SDM. In the preparation phase the specification of the features is important. The SDM does not support in a structure to split the system in features based parts. For larger systems with more possibilities it can be useful to have such a structure. In the first part a structure will be proposed for the specification of the features. The second part is reflection of the SDM. In (Morsinkhof, 2019) recommendations were made that can be useful to improve the SDM.

3.1 Analysis of the specification for different features

For the implementation of the method, the cyber-physical system under development must be split into features. Features are parts of the system that can be designed and tested separately or on top of other features in this case.

The method that is evaluated, does not give a structure to divide the features. Therefore, it is needed to find a way to split a system into feature-based parts. This is the way the sub-question of this research is: What kind of structure can be used to split the system into feature-based parts?

For the design of the cyber-physical system it is necessary to have a good division of the features. Spending enough time on this part before implementation will result in a better structure of the final system.

The SDM explains the working of the framework with an example, the in-house produced device, called a mini-Segway. This is a self-balancing device with two wheels, like a normal Segway. This is a small system with only a few features. For the specification of the features there was not a given structure. This mini-segway example will be used to confirm the working of the proposed structure for the specification of the features.

In this thesis the structure for the specification of the different features is based on two different classification systems for cyber-physical systems. The following two systems will be explained:

- (RobMoSys, 2019) describes the different abstraction layers of an embedded system.
- In (Broenink and Ni, 2012) is a system explained with more other details is the software part.

A combination of those two structures will result in a structure for the specification of the different features of a system. The combination will be explained after the two ways for the classification of cyber-physical systems.

3.1.1 Abstraction layers

According to (RobMoSys, 2019) a system has different abstraction layers. Each layer from bottom to top ads more functionality. Starting at the hardware layer, this is the layer where the drivers of the actuators are implemented and the readout of the sensors is implemented. Ending at the mission layer, where a system can perform a series of tasks.

In Figure 3.1 the different abstraction layers are shown with an example. In some systems the top layer is not implemented. Those systems stop at the task layer, because some systems are specific to one task. All other layers are always implemented but sometimes they cannot be pointed separately.

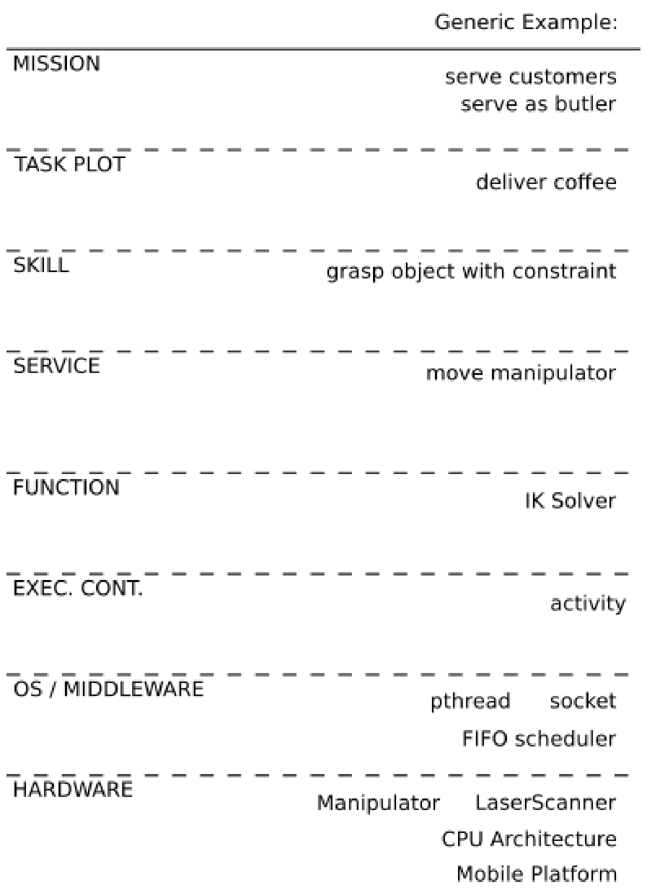

Figure 3.1: overview of abstraction levels (RobMoSys, 2019)

3.1.2 Division of an embedded system

According to (Broenink and Ni, 2012) every embedded system can be divided into three parts. The process, I/O hardware, and software. The division is shown in Figure 3.2. The process part of an embedded system consists of the motors, sensors, and the physical system parts. At the hardware the translation from analog signals to digital and visa versa is done. There is a small overlap between the hardware and software parts, in the overlap the drivers of the hardware components are implemented. At the software there are different layers.

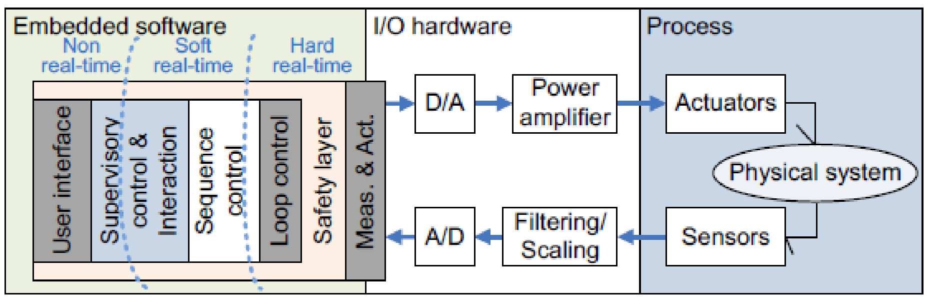

Figure 3.2: structure of an embedded system(Broenink and Ni, 2012)

The parts in the software, are mostly based on safety and sequence control. To make sure that the program will run and does not crash the embedded system.

3.1.3 Complete structure for the specification of a cyber-physical system

The structure of (RobMoSys, 2019) has more detail on the software part than the structure noted in (Broenink and Ni, 2012). The combination of those two different structures will lead to a more detailed overview of a cyber-physical system. The combination will lead to more detail in the supervisory control and interaction part of the structure of an embedded system. This is useful to make a specification of the different features. The hardware part of the structure was already more extensive, here a combination will not lead to more detail. The combination of both models is shown in Figure 3.3.

3.1.4 How to implement the structure

For the specification of different features the structure will be used from left to right. First, define the mission of the system under development. When the task is clear, it can be divided into multiple skills that are needed to execute the task. The same applies to define services and functions. From the specification of the functions it is possible to decide the hardware that is needed for the execution of the functions.

In the structure there are layers for the safety and execution control. Those layers are important to run the software. For finalizing the cyber-physical system the layers must be implemented as well. Those layers are not specified as separate features. The safety layer and loop control will be implemented in the detail level of the features. For example in a motor controller, the base is to run it in two directions. The next level of detail is to add speed control. The last level is to implement end positions of the system as well for safety.

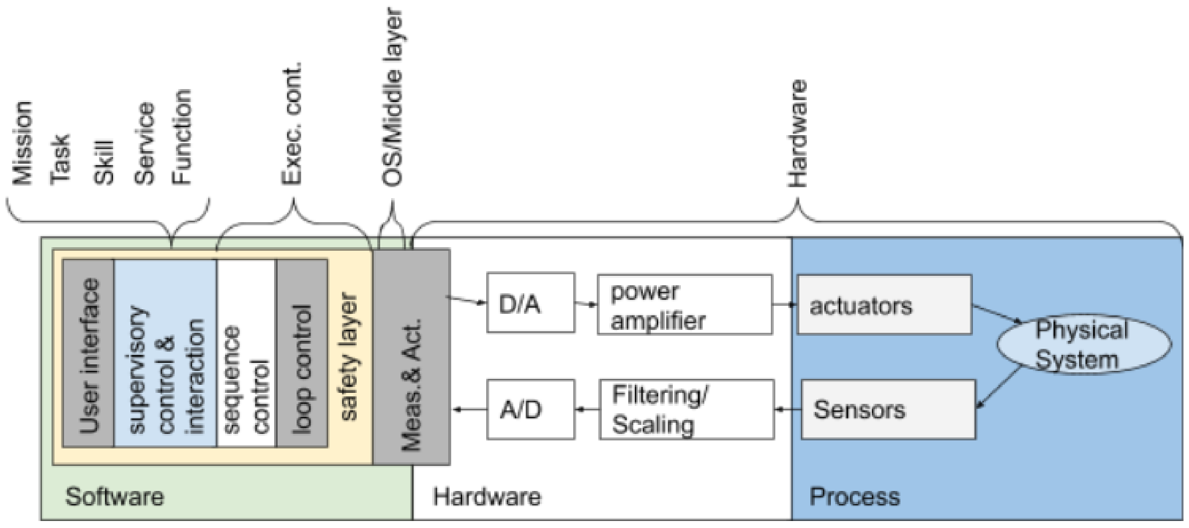

Figure 3.3: combination of both structures of an embedded system

3.1.5 Example of the specification of features of a cyber-physical system

In the SDM an example of the mini-segway is used to make their method clear. This example will also be used to explain how the structure can be used.

The mini-segway is a self-balancing device with two wheels. The scenario for this cyber-physical system is to move balanced around the room. The scenario can be translated into the service of the system. The service of the system is to move around on two wheels without touching the ground with another part. To reach this service there are multiple functions needed. The functions of the mini-segway will be, move forward/backward, balancing, and steering. For the three functions three hardware parts are needed, a sensor for measuring the

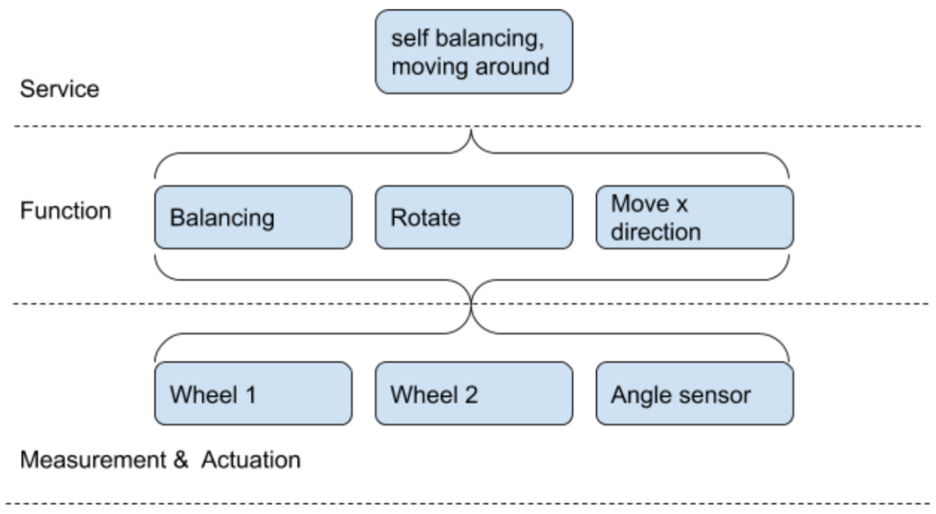

Figure 3.4: The division of features of the mini-segway

angle of the mini segway, and two wheels, with the possibility to control their speed. The specification of the different features is shown in Figure 3.4.

The specification of the features as in 3.1.5 is also corresponding with the features according the SDM. The structure for the division of features will also be used for the specification of features in the case study.

3.2 Analysis of the reflection according to the SDM

The SDM also describes a way to keep track of the number of revisions. This will also be used to evaluate the method. For the design method models with different detail levels of the physical system are used. It is assumed that the development process is started with an ideal model or a more abstract model. Every cycle a richer model is used until a point is reached where the model is competent to describe the behaviour of the prototype. The order of adding these aspects should be determined before the cycles are started, as they influence the structures of the models that are required.

The first step is to divide the system into features that should be added. When the features are specified they can be implemented. After the implementation of the features it is possible to evaluate. The evaluation of this method will be the number of iterations that are needed to implement one feature. In the ideal world there is only one iteration for each feature. If there are a lot of redesigns needed for a feature it would suggest that the order in which the details were implemented was sub-optimal. Because the evaluation is after the development process, the results can be used as an experience in the next project and will result in a better-structured method.

In (Morsinkhof, 2019) the method is already evaluated on another setup. In this report recommendations are made according to the evaluation of the design process. In the process that has been gone through, no more mistakes were found by testing each feature separate, than simulation on a couple of features together. One feature can work, but the combination of features is where the mistakes are made. For simulation purposes it is better to combine multiple features and simulate them at once.

When a simulation is used to test the software, the model needs to be detailed according to (Morsinkhof, 2019). When the model is not detailed enough, it can be the case that the software will be fine according to the simulation, and will not work well at the physical system. When this is the case debugging can still take the same amount of time compared to other design methods. This makes it necessary to have a well-detailed model otherwise, the simulation will not work to find all mistakes.

According to the method that will be evaluated, the number of iterations for each feature is important. The method describes that you first add the one feature and add on top of that feature the next one and try to simulate every feature. According to (Morsinkhof, 2019) it is better to skip the simulation some times, and test multiple features at once. The evaluation will still be done on every feature. The recommendations of (Morsinkhof, 2019) will be kept in mind during the implementation of the method.

4 Analysis of evaluation for design methods

If a new design method is suggested, most times the design method is not compared to other design methods. The result of this is that the choice of a design method for a project is based on user experience.

The evaluation of design methods is difficult, due to human influence. To eliminate human influence a lot of people need to implement all different design methods. If different humans use the same system every time, probably the last method will be the best. The person that is implementing the method, will get familiar with the system under development and will do it better every time he tries to design the system. Another approach could be to use another system every time, but that approach is also hard to compare because of the number of different systems and methods. Therefore for a good evaluation of multiple methods requires multiple persons with the same skill level.

A possible way to compare different methods is with evaluation metrics. In areas that have an overlap with the design of cyber-physical systems several comparisons exist. For example programming methods and codebases have a benchmark for evaluation. In codebases most of the metrics are used to evaluate the structure of the resulting code. In programming methods most metrics say something about the process. A combination of those two evaluation benchmarks can be used for the evaluation of cyber-physical systems.

4.1 Evaluation of programming methods

A comparison between different programming methods is made in (Chandra, 2015). He claims that the following stages of software development projects are mandatory:

- Feasibility analysis,
- Requirement analysis,
- Design,
- Coding,
- Testing,
- Implementation/Deployment,
- Maintenance.

The stages can also be used to develop cyber-physical systems. In software the middle part of the phases is based on code. In the development process of a cyber-physical system the middle phases includes also the hardware development. In (Chandra, 2015) there are metrics used to compare different methods. Some of these metrics can also be used for the evaluation of a design method.

The following metrics are used for the comparison of different programming methods and can be applied to the evaluation method for design methods.

- Cost, How much needs to be expended to use the method?
- Risk factor, Is the final product exactly functioning with the expected behavior?
- Simplicity, Is the method easy to implement?
- Development time, How much time is needed?

- Expertise required, Is there a high skill level required to use the method?
- Phase containment of error, Is an error made in a phase directly visible, or can it occur that it becomes visible at a later phase?
- Resource required, How much resources are needed to use the method?

These metrics are based on the process side of development. The measured metrics are on a scale from 1 to 5, where 1 is not or low, and 5 is always or high. Most of the metrics can not be measured exactly, so there will always be a human influence in the comparison. Cost is a combination of development time and the required resources. Because the metric is double it will not be included in the evaluation.

4.2 Evaluation of codebases

For the evaluation of codebases the following article is used (Sonjara inc, 2019). This article describes metrics that can be used to compare different codebases. For cyber-physical systems they can not be used directly. The explanation of those metrics is suited for codebases, but the metrics itself are usable. So, the explanation of the metrics is changed compared to (Sonjara inc, 2019).

Some of the metrics are not involved because they can not be applied to design methods. For the evaluation of a design method, the following metrics can be used:

- Readability, Are the code & models clear to read, is there a nice overview of the structure?
- Adaptability, Is it possible to make a quick minor change?
- Maintainability, If there is a change of a module, is it possible to replace it rapidly?
- Performance, Does the final code meet the time requirements? Or does the method cause much overhead?
- Interoperability, Are the features/ modules easy to implement in another setup?
- Software engineering process, In this case, it would be a comparison of the time spent preparation, coding, and testing/debugging.

4.3 Evaluation metrics

According to (Chandra, 2015) there are six phases of development. The feasibility study and requirement analysis will be skipped for the evaluation of design methods. The two phases in front of the design process are necessary to check if the goal of the system is reachable. But this is not necessary for implementing a design. So only the phases: design, code, testing, and implementation will be used during the design process. Maintenance is important for every system to make sure that it will work when in use, but it is not included in the design process.

Some of the metrics of (Chandra, 2015) and (Sonjara inc, 2019) can be used to describe a phase of the design process. Other metrics can be used in multiple phases or describe something about the complete design process. Metrics like the development time can be used for each phase. The time used to complete a phase can be useful to compare different methods. The same for simplicity. One phase can be harder to complete than another. Normally if the design phase is well executed the implementation takes less effort and should be easier. But this can differ between different methods.

The combination of the two structures will lead to an evaluation structure that consists of three parts, the design phases, the design process, and the resulting models and code. For a good comparison the metrics need to have some levels, like in chandra2015comparison the metrics scale from 1 to 5. Where 1 is less and 5 is many. If the metrics are used to say something specific about one design method, and there is not a comparison performed, then the metrics will scale from 1 to 3. This is because the extra levels are only necessary to indicate the difference between different methods.

In the resulting structure the first metrics will be used for every phase, the others will be used for the complete process.

- Metrics for each phase, design, code, test and implementation

  Simplicity, Is the phase easy to implement?

  Development time, What is the time spent in the phase?

- Metrics for the design process,

  Risk factor, Is there a risk that the system under development will fail?

  Expertise required, Can this method be used by everyone without experience?

  Phase containment of an error, Is a error made in a phase directly visible, or can it occur that it became visible at a later phase?

  Resources required, Are there a lot of resources needed to implement the method?

  Flexibility, If a phase is implemented and there is a small change needed can that be done?

- Metrics for the resulting code and models,

  Readability, Is the resulting structure of the models and code readable?

  Maintainability, Is it possible to extend the lifetime of a system with maintenance?

  Performance, Is there a lot of overhead in the system that will reduce the performance?

  Adaptability, Is it possible to adjust the system to a new goal/environment?

With the use of those metrics the human influence will be less, but still there will always be some influence of the person that is evaluating the design method.

5 Apply the design method

To evaluate the design method, it will be applied to the KUKA youBot. See Section 2.2 for information on the youBot. By applying the method to a cyber-physical system, three things can be concluded,

1. The functionality of the design method
2. The structure for evaluation of different design methods
3. The structure for specifications of different features

The evaluation of the design method will take place according to the metrics explained in Chapter 4. With the evaluation of the design method, it can be compared to other methods.

To apply the design method to the youBot, two conditions must be met. The first is to get the goal of the cyber-physical system clear. The second is to specify the different features that are needed to reach the goal. For the specification of the features the structure of embedded systems proposed in Chapter 3 is used.

The hardware drivers are already developed for the youBot. The drivers are developed by KUKA and cannot be changed. Because a part of the system is already implemented, the implementation will not cover the complete structure. The part of the youBot that still needs to be developed is in the area of supervisory control & interaction.

5.1 Specification of the features of the youBot

For the specification of the features of the youBot, first a scenario is needed that is reachable with the possibilities of the youBot. The goal of the youBot will be translated to the following scenario.

5.1.1 Scenario

The youBot has a base that can move and an arm that can make a countermove to keep the end-effector on the same position. Until it is not possible anymore to reach the position due to movement limits. This will be the scenario that will be implemented in the system. The scenario could be used to stabilize a camera position for instance or to pick up an object when driving by. For the implementation the last segment of the arm will be kept in constantly the same angle. This is needed for the orientation of the object that can be picked up or attached to the end-effector.

5.1.2 Features of the youBot

For the implementation of the scenario first, the specification of the features is needed. The structure proposed in Chapter 3 will be used to specify the different features of the youBot. The features will be specified from top, the skill layer, to bottom, the hardware.

The stabilization of the end-effector can be handled as a skill in the proposed structure. So, in this case the skill is the same as the scenario of the system. To implement the skill two services are needed.

The first is the movement of the base and the other is the movement of the arm. If the two services are implemented they can be combined and only the location of the arm needs to be translated with inverse kinematics to the angles of the joints.

According to the division in the services, this will work down to the lower levels of the structure. For the arm there are different features than for the base, and there is no overlap between those two services. The function of the arm is the inverse kinematics to control the location of the end effector. The function for the base is converting movement speed into RPM of the joints. By keeping the rotation, x-direction, and y-direction in mind. The specification of the x and y direction is shown in Figure 5.1.

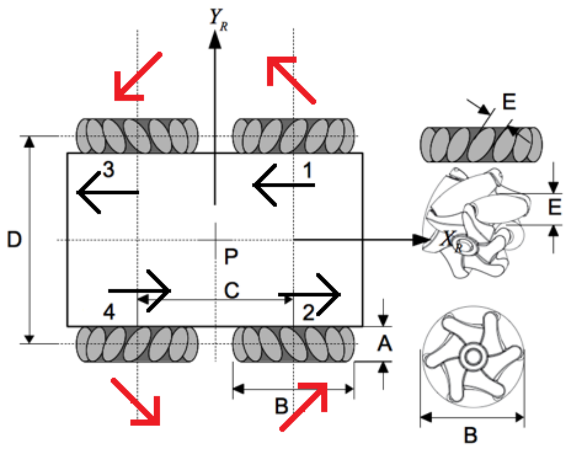

Figure 5.1: Movement directions of the base. (KUKA, 2015). Red arrows are the force of the wheels while moving in the direction of the 4 black arrows.

The execution is already implemented in the software structure of the youBot. The commands that will be sent to the hardware will be sent through the EtherCAT interface. Al lower layers of the structure are also already implemented on the system. So, for the design process they are not needed. For the simulation part, the lower layers are still needed. A minimalistic version will be implemented of the features that are needed to prove the working of the higher layer features.

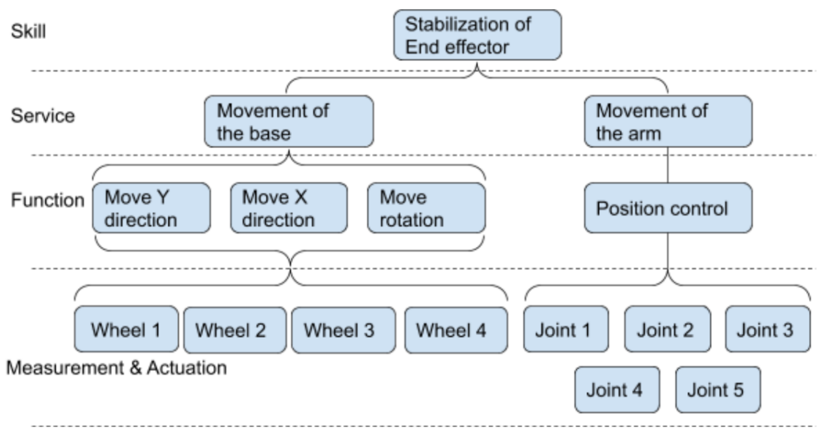

Figure 5.2: Specification of the features

5.2 The model

In the implementation phase there are two different parts, the model for simulation purposes, and the control software. The model is used to test the control software and verify the correct working. The control software should be designed in a way, that it is directly useable on the physical system. But in this case the hardware drivers are already implemented and the communication to the hardware is not implemented in the simulation. The result is that for the translation of the control software to the physical system a conversation is needed.

During the development process the model needs to evolve as well. In the beginning, an ideal, or more abstract model is enough to verify the software. The ideal model will not be competent to describe the real behavior of the physical system. The model that is used will become richer during the development process, until a point where the model is competent to describe the behavior of the cyber-physical system. The model will be created in 20-Sim. 20 sim makes it possible to simulate the software also.

According to the design method, all features will be implemented after each other. In this case the features can be divided into two sets of features. Figure 5.2 shows that there is no overlap in the features sets of the arm and base in the lower levels of the structure. The only overlap is in the skill level, there the influence of the base on the arm is the position in real-world coordinates. So, for the implementation, the youBot will be split into two development cycles, and in the end they will be combined.

The models of the arm and base will be separated until the last phase. This makes it possible to do the design process on both parts separately. Both parts will have an overlap in the design process. The model of both systems will increase in detail the same way. For the last step where both models are combined the arm must be capable to get an offset of the base.

5.2.1 Detail level

Before the method can be used a decision should be made about the order of detail which is needed. The joints of the arm have rotation limits and can not rotate freely. This should be implemented in the detail levels of the model. The movement limits does not count for the wheels. This is where the behavior of the joints of the arm is different from the wheels. This results in an extra limit of the arm joints compared to the base wheels. The next detail is to include the power of the motors. The motors cannot run immediately at the desired speed. The delay in reaching the desired speed must be included in the model. Otherwise the position of the end-effector will not be accurate, at the beginning of a movement. The last detail that should be added is the delay sensor readout due to communication. This will result in a delayed feedback loop and therefore there is a disturbance in the location calculation of the youBot. So, before going to the physical system, there should be three levels of detail added to the ideal model. The ideal model is in this case the kinematic model. This results in the following detail levels.

1. Kinematic model,
2. Motor movement limitations,
3. Motor power limitations,
4. Communication time limits [1],
5. The physical system.

The first level is the kinematic model, in this model the behavior of the motors is described. The place of the motors on the base of the youBot and the lengths of the bodies of the arm. The position and orientation of the omni-wheels influence the movement of the physical system. This is the basic behavior of the youBot, and will not be competent to describe the working of the cyber-physical system.

By adding motor movement limits to the arm, the behavior became closer to the physical system. The limits are implemented in the arm model. The bodies of the arm cannot rotate 360°

[1] This level of detail will not be implemented.

freely. Due to the limits not every position of the end-effector is possible. The inverse kinematics change by adding more detail to the system. There are at most two options possible to reach a position of the end-effector. The position with the least movements of the bodies should be used to reach the location. The base does not change when there are limits in the movement. The wheels can rotate freely and the behavior stays the same. The movement limits of the motors are necessary for the arm, by skipping this detail level, the model will evolve to fast too find errors quickly.

To increase the detail level of the model to the next step, the maximum power of the motors is added to the behavior of the model. The model became closer to describe the real-world behavior of the system. Due to the maximum power limits the motors will not rotate instantly on the desired speed. The result is that the youBot moves slower than desired at the beginning of a movement. This needs to be compensated to reach the goal at the given time. To stabilize the end-effector, this behaviour should be taken into account. The implementation of the motor behavior in the models, is already made by (Spil, 2016).

The last level of detail that should be added is the delay due to communication limits. Due to communication limits the feedback of the wheels is not instantaneous, but has a small delay. The communication can handle 10.000 messages every second. So, divided by the 10 joints of the system the communication can handle 1.000 messages for each joint, this is both sending and receiving. Due to the communication the update frequency of an encoder value is at least 500Hz, with a movement-speed of a maximum of 0.5 m/s. The error due communication is only 0,001m. This error is small but will result in a different behavior of the model when implemented. The change is not required to prove the design method. This level of detail will be skipped in the design process, because the disturbance in the signal of a mm can be neglected. This would result in four levels of detail.

The last level is the physical system itself. The model of the fourth level of detail should be competent to describe the behavior of the physical system. The control software should not show any bugs when this level of detail is used. The communication protocol is not implemented in the model. This should be added to the software before testing it on the physical model. The environmental influences on the youBot are also not added to the model.

The features will be implemented and tested at the detail level of the model. If the test holds, the level of detail of the model will increase, the control software should also increase. This is the inner cycle of the model. Some functions of the software will work for a basic model, but not for a detailed model and the other way around. The outer cycles will add a new function. According to the specification of the features in Section 5.1.2 four features should be designed in the function layer. Two features in the service layer and one in the skill layer. There are in total seven features that should be implemented, thus the inner circle must be completed seven times.

5.3 Implementation of the features

The implementation of a feature takes at least four cycles. The cycles are specified by the levels of the model. Some of the features need another revision to meet the test requirements. The steps that are taken for each feature are listed. Step 1,2,3, are the outer cycles of the design-method, and A, B, C are the inner-cycles. The second level of detail is not added to the base features. The wheels do not have a physical end-stop. To show the working of the design method, the first feature is described in detail. For the other features only the important issues are noted.

5.3.1 Move X-direction

The first feature is that is implemented is the movement in the X direction, forward and backward. The first feature is also noted with the cycles the method describes. The other features are implemented with the use of the design method, but this is noted.

Step 1

The function needs to move the four wheels of the youBot at the same speed, and the left wheels in the opposite direction compared to the right wheels. The assumption is made that there is no object blocking the path of the youBot. To confirm the working of the feature, the feedback from the absolute value of the encoders should be the same for all wheels, and also the count of the encoder pulses should be correct with the distance that should be covered. The encoder pulses of wheel 1 and 3 should be the inverse of the value of wheel 2 and 4. For the test a movement time and an end-position will be given. The base should reach the end-position in the given time

Step 2

A1: The control of the wheels is based on the position and orientation of the wheels on the youBot.

B1,1: The code is tested against the kinematic model.

C1,1: With the designed control software, the model of the youBot reaches the end-position in the given time.

B1,2: This level of detail is skipped for the base. The detail level includes the motor end limits. The wheels of the base do not have end limits.

B1,3: The code is tested against the non-linear model, the motor behavior is not linear so it can not start direct at full speed.

C1,3: The model of the youBot, did not reach the end position in exact the given time. Due to the maximum power of the motors. The youBot can not move directly at the given speed.

A2: The control software is adjusted to take care of the error, and compensate the speed to reach the end-position in time.

B2,1: The control software is tested against the linear-model.

C2,1: The model has the same behavior as C1,1.

B2,3: The software is tested against the non-linear model. The speed has an overshoot to compensate for the slower speed than at the start of the movement.

C2,3: The model of the youBot, reached the given position in time.

B2,4: The control software will be used on the physical system. For making the software ready, the communication protocol must be added. To send the right messages, but the control software stays the same.

C2,4: Tested at the physical model, the youBot moves as expected as in the simulation.

Step 3

The feature behaves as expected, the given end location is reached in time. The next feature that is added on top of this feature is to move in the Y-direction.

The structure shown in Subsection 5.3.1 is used to implement the seven features of the system. Only for the other six features, the highlights are noted, and not the complete structure.

5.3.2 Move Y-direction

The movement in the Y-direction is almost the same as the movement in the X-direction. Due to that this is the second feature there is a learning curve. The mistake that were made during the development of the first feature were not made in the implementation of this feature. This results in fewer mistakes during the development and also fewer cycles than in the previous feature.

5.3.3 Move rotation

This looks like the movement in the Y and X direction, the two differences are that the feedback is not in meters but radians due to the fact it is a rotation movement, and the rotation direction of the wheels.

5.3.4 Movement of the base

Since the arm does not influence the movement of the base, and there is no overlap until the skill layer, the service of the base is implemented before the function features of the arm. During the implementation of the movement of the base the three features in the function layer are combined into one system. The calculations for the movement of an omni-wheel platform are explained in (Taheri et al., 2015). Equation (5.1) is the formula that can be used to calculate the speed in the X or Y direction, or the rotation speed. Where r is the radius of the wheels, Vx and Vy the movement speed in the 2 directions, L the length of the base and W the width of the base, and $\omega 1,2,3,4$ the rotation speed of the wheels.

$$\begin{bmatrix} \omega z \\ Vx \\ Vy \end{bmatrix} = r/4 \begin{bmatrix} \frac{1}{L+W} & \frac{1}{L+W} & \frac{1}{L+W} & \frac{1}{L+W} \\ -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} \omega 1 \\ \omega 2 \\ \omega 3 \\ \omega 4 \end{bmatrix} \tag{5.1}$$

The addition in control code is that the movement must be compensated by the current rotation to move constantly in the same direction. Also because there can be a rotation involved in the movement, the X and Y position can not be read from the total encoder count. If there is a rotation of 360° while moving to a position. The encoder of the wheels will result in 0 at the end position. This results in an adjusted feedback loop. The fourth level of detail showed that the environment also influences the behavior of the youBot. During the test on the cyber physical system the wheels have some slip, due to the slip the end-position of the base does not correspond with the desired position. But for confirming the design method, there is no need to compensate for this behavior. If it is compensated, it should at least lead to another design cycle.

5.3.5 Arm inverse kinematics

The first feature of the arm is the inverse kinematics. The goal of the inverse kinematics is to calculate the angles of the arm to reach a given position of the end-effector. For every position of the end-effector there are at most two possible options. At the first try the second option was not calculated, this was solved in the second iteration. By implementing the maximum power of the motors, the behavior changes. Only there is no possibility to compensate for the speed because the driver is already set to reach the specified position as fast as possible. The inverse kinematics of the arm are calculated in (Abuqassem, 2010).

In Figures 5.3 and 5.4 is the top view and planar view of the youBot arm shown. According to (Abuqassem, 2010) the angles of the joints can be calculated with equations 5.2, 5.3, 5.4, and 5.5 . This equations are used to calculate the angels of the joints. Where ψ is given as input to the control software. The fifth angle is not calculated this is only a rotation of the head and

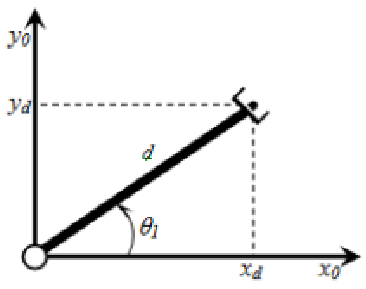

Figure 5.3: Top view of the youBot arm. (Abuqassem, 2010)

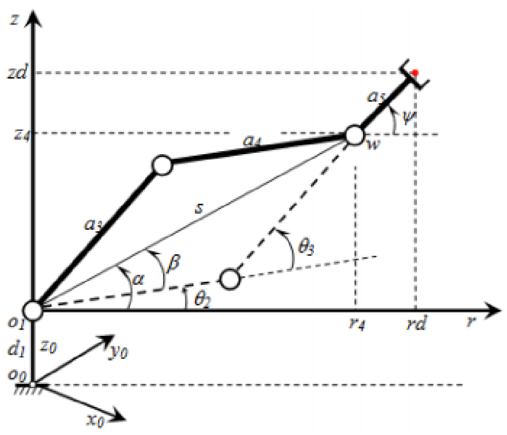

Figure 5.4: Planar view of the youBot arm. (Abuqassem, 2010)

does not change the location of the end-effector. This joint is kept constant by the drivers of the youBot, so it does not need control software to keep it in the same position.

$$\theta_1 = atan2(y, x) \tag{5.2}$$

$$\theta_2 = \alpha \pm \beta \tag{5.3}$$

$$\theta_3 = atan2(s^2 - a_3^2 - a_4^2, 2a_3a_4) \tag{5.4}$$

$$\theta_4 = \psi - \theta_2 - \theta_3 \tag{5.5}$$

5.3.6 Movement of the arm

The feature of the arm in the service layer is the combination of the control of the arm, as in the inverse kinematics, and the addition of an offset for the base position of the arm. This results in the different calculations for the angles of the joints. With the implementation there was not a second iteration needed to solve issues.

5.3.7 Stabilization of the end-effector

For the stabilization of the end-effector a prediction of the movement of the base is needed to compensate it with the movement of the arm. First the control software is tested against the kinematic model. The non-ideallness of the motors is ignored, this makes it easier to calculate the prediction of movement. There is extra control code added to work with the model with power limitations. When this is added, the control does not work for the kinematic model anymore.

When testing it on the cyber-physical system, the arm keeps the end-effector in the same position, when there is not a rotation involved while moving the base. The rotation adds slip to the movement, this will result in a disturbance of the position of the end-effector. The level of slip is not measurable at the sensors, so compensation for this is hard to implement. This will also exceed the goal of the thesis. If there is a compensation implemented there is another development cycle added. This cycle is shown in the evaluation in Section 5.4. The slip can be reduced in the control software. In (Stonier et al., 2007) (Kilin et al., 2017) a method is described to compensate for the slip of an omni-wheel platform.

5.4 Evaluation of the implementation

The design method describes a way for feedback on the implementation. The number of iterations taken to design a separate feature will be a grade for the implementation. If there are a lot of redesigns needed for a feature, it would suggest that the order of implementing the features was sub-optimal.

The resulting structure of different models and their detail level is shown in Figure 5.5. In this figure the three models are shown, the combination model does not start at the kinematic level. But at the second level of detail, the movement limitations. The limits are included in the part of the arm and tested for the arm. The behavior of the combination does not change, except for the points that can be reached by the end-effector.

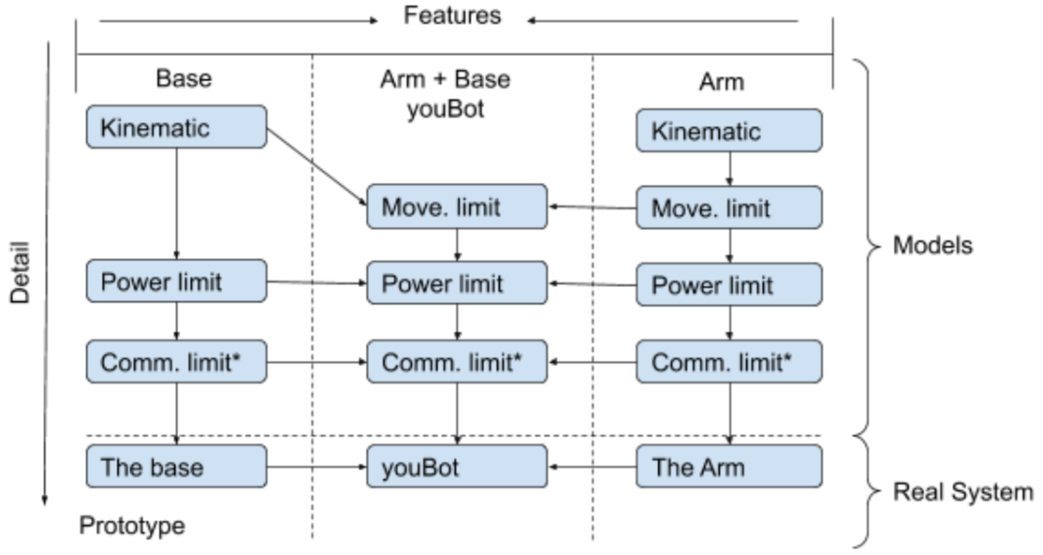

Figure 5.5: The overview of the detail levels of the models. The implemented features will start outside and will be combined in the centre. * not implemented.

The movement in the Y-direction does not need a second iteration to get the working correct. Compared to moving in the X-direction the Y-direction takes a cycle less. The mistakes made in the X-direction were not made in the Y-direction, due to a learning curve. The number of iterations of each feature is shown in Figure 5.6.

In the matrices in Figure 5.6 are not a lot of redesigns visible for a feature. This would suggest that the detail levels are implemented in the correct order. The design errors were found in the

$$\begin{bmatrix} B1,1 & B2,1 \\ B1,3 & B2,3 \\ & B2,4 \end{bmatrix}$$

A: Move x direction

$$\begin{bmatrix} B1,1 \\ B1,3 \\ B1,4 \end{bmatrix}$$

B: Move y direction

$$\begin{bmatrix} B1,1 \\ B1,3 \\ B1,4 \end{bmatrix}$$

C: Move rotation

$$\begin{bmatrix} B1,1 & B2,1 & \\ & B2,3 & B3,3 \\ & B2,4 & B3,4 \end{bmatrix}$$

D: Movement of the base

$$\begin{bmatrix} B1,1 & B2,1 \\ B1,2 & B2,2 \\ & B2,3 \\ & B2,4 \end{bmatrix}$$

E: Inverse kinematics

$$\begin{bmatrix} B1,1 \\ B1,2 \\ B1,3 \\ B1,4 \end{bmatrix}$$

F: Movement of the arm

$$\begin{bmatrix} B1,2 & B2,2 & \\ B1,3 & B2,3 & B3,3 \\ & B2,4 & B3,4 \end{bmatrix}$$

G: Stabilization of the end-effector

Figure 5.6: Evaluation matrices according the design method. A,B,C, and D are the features of the base. E and F are the features of the arm. G is the combination of the arm and base, the youBot.

feature that is currently under development, and not in the previous feature. This also confirms that the order of features is chosen correctly. The way to specify the different features proposed in Chapter 3 works for this case study.

6 Reflection and evaluation of the SDM

After the case study on the youBot the design method should be evaluated. The reflection part of the SDM is already done in Section 5.4. In Chapter 3.2 some points should be reflected. Some parts of the design method are implemented in another way than expected. This will be discussed in the reflection part.

The second part is the evaluation with use of the structure mentioned in 4.3. The use of the evaluation metrics described there should help to compare the design method with other methods.

6.1 Reflection of the SDM

For the reflection the findings of (Morsinkhof, 2019) are reviewed. Also some parts are done differently than (Broenink and Broenink, 2019) explains. First the findings of (Morsinkhof, 2019) will be reviewed.

6.1.1 Recommendations of (Morsinkhof, 2019)

According to (Morsinkhof, 2019), it is sometimes better to implement multiple features at once. The mistakes he found were not while testing a separate feature. Most of the mistakes he found are caused by combining multiple features into one system. The combination of multiple features is also described in Chapter 3 as a feature. The higher-level features are mostly a combination of lower-level features with the addition of control code. There should not be a problem with a lower-level feature when the interface between the features is well specified when implementing a higher-level feature. Testing multiple features at once can lead to errors, that can not be located easily. The advice is to simulate every feature separately and continue to the next feature when the tests hold.

At the end of the design method there will be an evaluation of the process. According to (Morsinkhof, 2019) it is better to do an evaluation cycle more often. During the implementation process it was not necessary to evaluate more than once. The feedback can be taken for the next project, and in Section 5.4 there were no mistakes found in the design process. But the advice from (Morsinkhof, 2019) can be useful. When there is a mistake in the order of the detail levels of the model, it already can be visible in the iterations of the first feature. Continue with the other features can result in a lot of iterations for all features. So it can be better to reorder the detail levels.

The last finding of (Morsinkhof, 2019) is that the model must be detailed enough to find the mistakes. Otherwise there can be errors found when implementing the software on the physical system. The code will not be according to the first time right principle. In the case study the slip of the base was not included in the model. The slip that occurred for the first time when testing the movements of the base in the service level caused a displacement for the position of the youBot. This was not expected when preparing the design process. The mistake was found relative quickly, but the simulation did not show the problem. The influence of the slip can be reduced, but only for a certain environment, if it is used on another floor it should be compensated in another way. Adding this detail level to the model will be hard and should be redone for every environment. Other mistakes during the development were found while the model was not capable to describe the real-world behavior in full detail. So, if detail levels increase in small steps problems can be found. But some will be unnoticed while simulating.

6.1.2 Difference between execution and the SDM

In the method itself, the control software of a feature is designed against the first level of detail in the model. This level is not used in the evaluation part of the method. In the Section 5.3. The first detail level is also used in the development cycles. The way for handling the control software in the first level of detail is the same as in the other levels, the structure stays the same. When developing a feature, it is only allowed to proceed to the next level of detail when the test passes. When a test on a higher detail level does not hold, the control software is adjusted and tested again to the first level of detail. When the test holds the next detail level is tested. When keeping track of how often a redesign is needed for the first detail level, it can be concluded whether the preparation went well. If all features are well specified, probably a few iterations will be required for the first level. So, the iterations of the first detail level can say something about the preparations.

The review of the method shows some differences in Figure 5.6 D and G than the design method describes. In the right upper corner there are empty places. This is caused because a higher detail level of the model showed a mistake in the control software, that can be compensated on that level of detail, but result in a code that does not work for the first detail level. In this case the movement for the base. When adding the maximum power of the motors to the model the position does not correspond with the calculations anymore. By redesigning the control software to take care of the behavior the motors the control software for the kinematic model is not adjusted. Testing another time to the first level of detail will not create another result for the test. So, during the implementation this detail level can be skipped a time.

Because the youBot consists of two separate parts. The detail levels of the models are also slightly different. Some of the detail levels are skipped in one of the systems. For example the movement limits of the motors. In the arm the motors can not rotate freely, this differs from the base. Therefore in the base the movement limits are either ignored or skipped. The order of the implementation does not change by skipping some detail levels. So, the order for the complete system stays the same, but in some parts a detail level is not used.

6.2 Evaluation of the SDM

The second part is the evaluation were it is possible to compare the design method with other methods. During the research there was no time to execute another method. So, the metrics mentioned in Section 4.3 are used to get an overview of the design method (Broenink and Broenink, 2019).

Due to the fact that the method is not compared with other, but only the metrics are used to say something about the SDM, the metrics will have a scale from 1 to 3. The words that will be used for the scale are depending on the metrics but normally means, Negative, Neutral, and Positive.

The first part is about the phases in the design process. The four phases in the design process are design, code, testing, and implementation. In Table 6.1 the score for the design phases are noted. The design part was not that simple and needs a lot of work to prepare for the other phases. When the design part is done the correct way the other phases of the development are taking less time, and will be more simple to do. This is visible in Table 6.1. The implementation at the end of the physical system did not take much effort because of the simulation of the parts before testing them on the system itself.

The second part is about the overall process of the design method. The scores are shown in Table 6.2. The risk factor is very low. This is due to that every separate part is tested and simulated. If a part fails it is only a small part of the system that needs a redesign. This makes the risk of a complete system that is not working in the end low. The expertise required to use this method is not high. The person who is using it should have an understanding of testing and

Table 6.1: Evaluation metrics of the separate phases.

Phase	Simplicity	Development time
Design	Hard	Long
Code	Easy	Neutral
Testing	Easy	Short
Implementation	Easy	Short

simulating techniques. Due to testing every single phase, the containment of an error is located to one phase. This makes debugging quicker. The resources that are needed are programming and simulating environments and a prototype of the physical system. The prototype is needed to test the last cycle of the separate features. The flexibility of the structure is not tested during the case study. Because every part is tested separately and the next feature was implemented when the tests hold. There was no need for a change in a phase before.

Table 6.2: Evaluation metrics of the overall process.

Metric	Score
Risk factor	Llow
Expertise required	Low
Phase containment of an error	High
Resources required	Low
Flexibility	-

The last part is about the final system Table 6.3. This part is about the working and structure of the code. The code that is created during the development should be readable, maintainable, adaptable and of course the code should not have too much overhead that decreases the performance. The readability of the code is well done. Due to the implementation of the features separate, automatically the code is split into multiple functions. This increases the readability of the code. It is harder to say anything about the maintainability, the code is structured in a way that the lower-level features are used by higher-level features. When a lower-level is changed a little bit, it could influence the behavior of a higher-level feature. This is not tested during the case study, because the final code does not need an update to maintain the functionality. The performance of the code is fine. Because the simulation uses and generates the same signals as the physical system, there is not an overhead needed to test the code in a simulation. To give the system a new goal it is needed that it is adaptable. Looking at the structure, the separate features can be reused in the structure for the new goal. Only the higher-level features are changing. But when a part of the physical system is replaced, al higher-level features that are built on top of the feature that is replaced also may be replaced or redesigned.

Table 6.3: Evaluation metrics about the final system

Metric	Score
Readability	Good
Maintainability	-
Performance	Good
Adaptability	-

The first part of the evaluation shows that if the design phase is done correctly, the other phases will be easy and fast to realize. Good preparation before starting the code phase is necessary and takes a long time. But the time spends in testing and implementation can be reduced. Looking at the overall process there is not much expertise required and are not a lot of the resources required. The physical system must be available for testing the last level of detail. When building a prototype the system must already be in a state that it can be tested. Due to the division of features in the implementation the readability is good. The overhead is low and can result in good performance. For maintainability and adaptability the score is neutral, but this is not tested in the implementation phase.

7 Conclusion and recommendations

7.1 Conclusion

The SDM can be used to implement the control software for cyber-physical methods. The structure of the feature-based development cycles is good to implement the features in a structured way. With the use of the SDM, there were only a few redesigns needed for the features. Looking at the evaluation in Section 6.2 the time spent on testing is the same as the time spent on coding. But the time spent on preparation/design was longer. So, a longer preparation is needed for this design method but the time spent on debugging is not 50 % of the total development time. Thus the proposed design method can be a good system to implement cyber-physical system.

The addition of the specification of the features mentioned in Chapter 3 can be used to split the cyber-physical system into feature-based parts. By adding this to the method it became clear how to specify the different parts of the system. When the structure is implemented well the time spent on design can be reduced.

More evaluation cycles during the design process can reduce the mistakes made during the development of the control code. In the case study it was not useful to add another evaluation cycle. But still it can be useful for detecting that the order of detail levels incorrectly. By detecting the mistakes early in the design process, they can be solved for the other part of the design.

The metric proposed in Chapter 4.3 is used, but for a good comparison and evaluation of the metric it should be used on multiple design methods. For the design method in this thesis the evaluation shows the difference between the different phases of development. The other parts in the comparison are not that useful by only using it on this method.

7.2 Recommendations

To use the design structure of the SDM the physical system must be available before implementing the first feature. The last step for each feature is to test is on the physical system. With rapid development methods the prototype is not always available before creating the control code. To improve the work flow when the physical system is not available, it is better to skip the testing on the physical system to the end of the design phase.

For the specification of the features there is a structure proposed in this report. For the detail levels of the model a structure is not available. The levels that are used for the youBot are comparable with de levels of the mini-segway described in the SDM. So, there should be a research for a structure that can be used to define the detail levels of all kinds of cyber-physical systems.

The structure for the specification of features is used for the youBot and the mini-segway. For a good evaluation of this structure it should be implemented on more different systems.

A Overview of the models

During the research three models were used, with multiple detail levels. In total seven different models, the models of the different parts are structured and are showed in Figure A.1

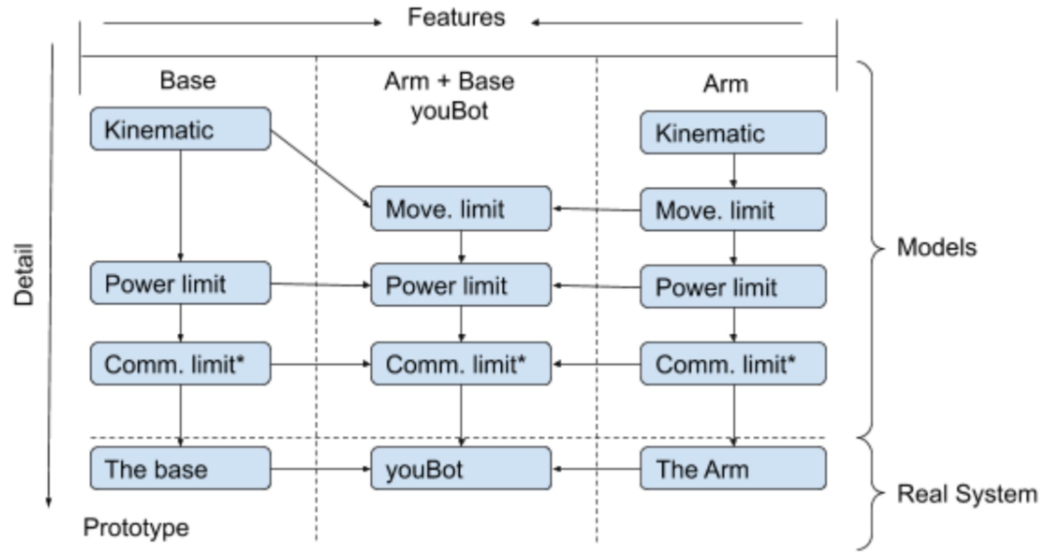

Figure A.1: The overview of the detail levels of the models. * not implemented.

The models are designed in 20-sim and can be used directly to simulate the behavior of the system. The features listed in Figure A.2 make the use of the models to confirm there behavior. For each feature the different detail levels of the model are used. They are all stored separately in the file system.

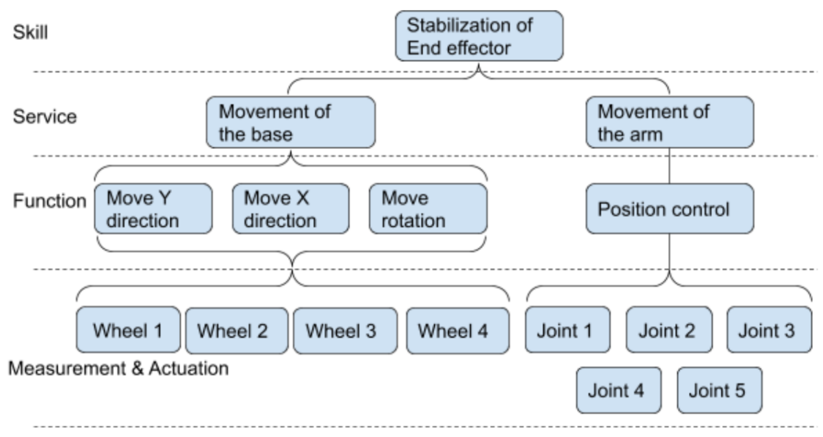

Figure A.2: Specification of the features

So for example the feature Position control has three separate files,

1. Against the first level of detail, the kinematic model.
2. Against the second detail level, the model with movement limits.

3. Against the third detail level, the model with power limitations.

This is done for every model. The model of the stabilization of the end effector against the first detail level, The movement limits, is shown in Figure A.3. The model has three parts, the physical system, the drivers, and the code. The drivers are already implemented in the software of the youBot so can also be included into the physical model, but for keeping the model with the control code the same way as is described in (Broenink and Hilderink, 2001) this layout was chosen.

By using the signal generators to specify the position of the end effector, and give a speed of the base, the simulation can be done. A example of the simulation is shown in Figure A.4. In this Figure there are three graphs, the first is the location of the base. In real world coordinates, the start point of the simulation is (0,0). The second plot is the angles of the arm joints in radians. In the thirth plot the position of the end effector is shown. When the Z location is at 0.8 M the specified location can not be reached.

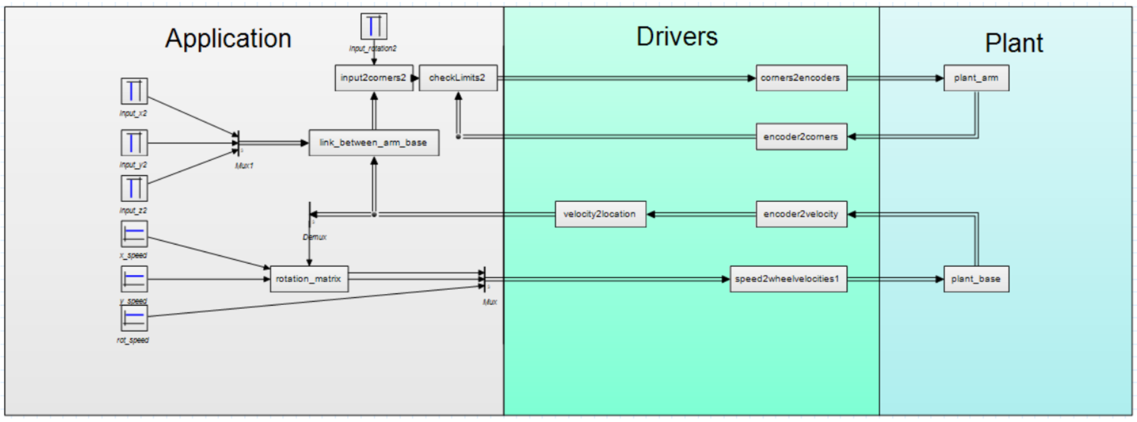

Figure A.3: Model + control code for the stabilization of the end effector. The detail level of the model includes the movement limitations.

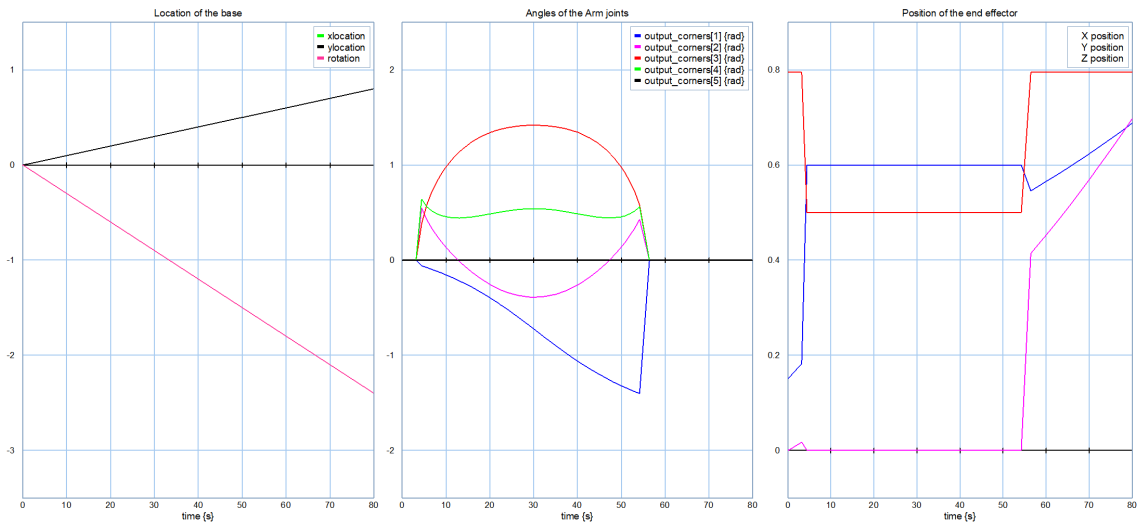

Figure A.4: The plot of the signals of the stabilization of the end effector code.

To use the application on the youBot the control code must be changed to make the use of the drivers that already are implemented on the software of the youBot. The features of the base and arm are tested on the youBot. The stabilization of the end effector was not possible to run on the youBot due to a defect in joint 3 of the Arm.

Bibliography

Abuqassem, M. R. M. (2010), Simulation and Interfacing of 5 DOF Educational Robot Arm, *Simulation and Interfacing of 5 DOF Educational Robot Arm*.

Bischoff, R., U. Huggenberger and E. Prassler (2011), KUKA youBot - a mobile manipulator for research and education, pp. 1 – 4, doi:10.1109/ICRA.2011.5980575.

Broenink, J. and Y. Ni (2012), Model-Driven Robot-Software Design using integrated Models and Co-Simulation, in *International Conference on Embedded Computer Systems, SAMOS 2012*, Eds. J. McAllister and S. Bhattacharyya, IEEE Computer Society, United States, pp. 339–344, ISBN 978-1-4673-2296-6.

Broenink, J. F. and G. H. Hilderink (2001), A structured approach to embedded control systems implementation, in *Proceedings of the 2001 IEEE International Conference on Control Applications*, Eds. M. W. Spong, D. W. Repperger and J. M. I. Zannatha, IEEE, pp. 761–766, ISBN 0-7803-6733-2, doi:10.1109/CCA.2001.973960.
http://www.ce.utwente.nl/rtweb/publications/2001/pdf-files/042R2001.pdf

Broenink, T. and J. Broenink (2019), Rapid development of embedded control software using variable-detail modelling and model-to-code transformation, in *33rd International ECMS Conference on Modelling and Simulation 2019*, pp. 151–157.

Calantone, R. J. and C. A. Di Benedetto (2000), Performance and time to market: accelerating cycle time with overlapping stages, **vol. 47**, no.2, pp. 232–244.

Chandra, V. (2015), Comparison between Various Software Development Methodologies, *International Journal of Computer Applications*, **vol. 131**, pp. 7–10, doi:10.5120/ijca2015907294.

Greg Law (2014), Speeding up software debugging.
https://www.embedded-computing.com/embedded-computing-design/speeding-up-software-debugging

Kilin, A., P. Bozek, Y. Karavaev, A. Klekovkin and V. Shestakov (2017), Experimental investigations of a highly maneuverable mobile omniwheel robot, **vol. 14**, no.6, p. 1729881417744570, doi:10.1177/1729881417744570.

KUKA (2015), Wiki page of the youBot.
http://www.youbot-store.com/wiki/index.php/Main_Page

Lee, E. A. (2008), Cyber physical systems: Design challenges, in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, IEEE, pp. 363–369.

Morsinkhof, N. (2019), Testing the structured method to design embedded control software.
http://essay.utwente.nl/78683/

RobMoSys (2019), Separation of Levels and Separation of Concerns.
https://robmosys.eu/wiki/general_principles:separation_of_levels_and_separation_of_concerns

Sonjara inc (2019), How to Evaluate a Code Base.
http://www.sonjara.com/blog?article_id=132

Spil, T. (2016), User-input device based command and control of the youBot using a RaMstix embedded board.

Stonier, D., S.-H. Cho, S. Choi, N. Kuppuswamy and J.-H. Kim (2007), Nonlinear Slip Dynamics for an Omniwheel Mobile Robot Platform, pp. 2367–2372, doi:10.1109/ROBOT.2007.363673.

Taheri, H., B. Qiao and N. Ghaeminezhad (2015), Kinematic Model of a Four Mecanum Wheeled Mobile Robot, *International Journal of Computer Applications*, **vol. 113**, pp. 6–9, doi:10.5120/19804-1586.

TRINAMIC (2011), TMcM-1632 Ethercat Manual.