# A Behavior-based Architecture for Excavation Tasks

**T. Groll, S. Hemer, T. Ropertz, K. Berns**

Robotics Research Lab, Technical University of Kaiserslautern, Germany
E-mail: groll@cs.uni-kl.de, s_hemer@cs.uni-kl.de, ropertz@cs.uni-kl.de, berns@cs.uni-kl.de

**Abstract -**

**The paper describes a software architecture for autonomous excavation tasks. For this, it uses a behavior-based approach which allows adaptation to the changing environment and unexpected events. As part of the presented architecture, basic primitives are defined which solve spatially limited motions. These can be activated as and when required. It is possible to have multiple of them running in parallel. Normally, an excavation task can be split up into several different phases. During these phases, the primitives will be activated as needed. To be reusable, the primitives are designed in a very general way. To show the performance of the presented architecture, an example application is built up which can be used to dig a trench. This application is implemented on a system running on a real backhoe loader.**

**Keywords -**
**primitives; behavior-based control architecture; excavator**

## 1 Introduction

Excavators are commonly used machines on construction sites and in mining environments. One of the main reasons for that is their multi-functionality. They can be deployed for transport or loading of material for example and as well for all kinds of digging scenarios. Targeting this, excavators are available from small machines with weights around 1 t to huge machines with over 200 t. Additionally, a huge set of different attachments is available for the use with excavators. Besides a lot of different kinds of buckets, it is also possible to use tools like drillers with such machines (compare [6]).

Because of the kinematic structure of a common excavator arm, it is necessary to control multiple degrees of freedom at the same time to solve a task properly. This requires some expertise. The basic kinematic structure is comparable to the one of common industrial robots but most of the excavator arms are controlled by hydraulic actuators. So it is comprehensible that the proceeding automation does not stop in the area of construction machines. In the past years, some projects in this field were executed. For example, there are THOR [15] and LUCIE [1].

According to the wide range of possible tasks which can be solved by an excavator, an architecture for an autonomous control system for such a machine should be adaptable. Another big issue while working on construction sites is that here the environment is very unstructured and can even heavily change while solving the task. In most cases, the changing of the environment is the goal of the excavation work. To deal with this situation the behavior of a control system should also be very adaptive.

Such an architecture for the control software of an autonomous excavator is described in this paper. With the presented approach the task of digging a trench as an example application can be executed autonomously. This typical excavation problem can be divided into four subtasks: *moving to trench start, digging, moving to dumping position, dumping*. So the implemented solution should address these phases as part of its architecture. Additionally, it should be possible to exchange the implementation for adaptability. If this is done properly, the same solution can be used to solve different tasks with only exchanging, for example, a single phase implementation. This could be the exchanging of the digging strategy between the excavation of big masses and a solution for creating a more precise shape of the pit. Another problem which is addressed by the architecture is the switching between the subtasks while system operation. This should happen according to the system state measured by the sensor system. In previous research presented in [2], it was shown that expert human operators overlap the solving of the subtasks a little bit. So the architecture presented here also should allow such overlapping. On the lower level of the software stack, primitive motion control modules are applied which can be reused for different tasks or subtasks. Another goal for the architecture design is to be scalable. This means that in future works the same solution could be used to extend the local trenching to an implementation for building up a whole construction pit. To show the performance of the presented architecture, an example application is built up which digs a trench. This is implemented on a system running on a *John Deere 410J TMC* backhoe loader.

Followed by this introduction, similar projects and other architecture techniques are presented. Then in section 3, the developed architecture is explained in a general way. In section 4, the implementation of the architecture for the trenching task is presented. Following this, the experiments and their results are shown. At the end, a conclusion is given which also includes a short explanation of possible future work.

## 2 Related Work

For every system which implements an autonomous excavation task, it is necessary to implement subsystems for solving smaller parts of the work cycle. Additionally, a strategy is needed which controls the execution of the subimplementations. For solving this, different approaches can be found in the literature.

### 2.1 Autonomous Excavators

A lot of projects in which autonomous excavation was implemented use finite state machines (FSM) to fulfill the task. For example in [3] a solution is presented which uses a FSM in a software system which allows a wheel-loader to load a pile of soil on a truck. In the implementation of mass excavation including truck loading for the *Automated Loading Systems* of *Carnegie Mellon University* parameterized scripts are used. These scripts implement the control algorithms for the different joints of the used excavator and are used in parallel. They are synchronized by event-triggered state changes, which allows interpreting the single scripts also as finite state machines. This solution is described in [12] and [13]. Schmidt describes in [15] a control system for the automated excavator THOR which also uses a FSM. Here, the state machine provides the input for a behavior-based system which generates the moving trajectory by activating primitives. Each of this primitives calculates control signals for every possible moving direction. In this works also some algorithms for environmental perception and interpretation are described. For example, he explains a gridmap-based solution to find the best starting point for the next excavation cycle.

On the low level of the control stack for excavation automation operates the solution presented by Maeda in [8] and [9]. Here, an adaptive impedance control implementation is shown which can control the bucket of an excavator along a given trajectory while moving through soil with a lot of unknown disturbances. Targeting the problems occurring because of this disturbances, the system can adopt the control algorithm while executing the excavation cycles and improve the precision of following the trajectory with every new iteration of the digging process. Besides these projects which all implement software system to drive real excavators, Du et al. describe in [2] a system that should improve simulation by including a virtual operator into the tested machine. For this, the developed algorithm should act as similar as possible to the movement a human operator would control. To find out how a human really solves the excavation task, they did detailed research with expert machine operators. Based on this, a detailed description of the excavation cycle while trenching is given. Starting from this knowledge a virtual controller for the excavation simulation was implemented. In the implementation also a state machine with hard state changes is used. However, one conclusion of the tests with the experts was that very experienced human operators let the states overlap a bit.

Also in the work presented in [16] no real excavator is used. There, a high-level task planner is presented which can be used to divide the earthwork of a whole construction site into small portions. The resulting segmentation delivers positions which should be used for local digging by an excavator. An algorithm to reach this location in a good manner is then given by Kim et al. in [5].

### 2.2 Motion Primitives

Using primitives is a good strategy to develop modules that can be reused for different tasks. If the primitives are additionally adaptable in their behavior, the range of possible usage is extended a lot. Addressing this, Ewerton describes in [4] an approach which uses movement primitives with parameters. In this work, he teaches a robot to play golf by demonstrating the basic trajectory to it. With the implemented algorithm the robot was then able to adapt this trajectory to vary the velocity of the golf club when it hits the ball. This is possible because they approximate the given trajectory with parameterized Gaussian functions. To adapt the velocity, and with it the hitting strength, the parameters of the Gaussian kernels are changed.

A similar solution is used for the *Dynamic Motion Primitives* (DMP) introduced by Schaal in [14]. Here, parameterized functions are used to approximate a trajectory which also include Gaussian basis functions.

Another approach for implementing movement primitives is given by Luksch in [7]. There the primitives are described as small systems that generate local control commands. For example, it exist a primitive to control the velocity of an object according to one criterion. The resulting modules can run in parallel and are triggered by stimulation which is given by a hierarchical behavior-based network. This uses an architecture earlier introduced in [10] which is called *integrated behavior-based control* (IB2C).

### 2.3 Robot Architectures

To design robot control software, it is necessary to use an arbitrary software architecture with structure modules and define rules for the execution of those. Weidauer has done so in [17] for a standard industrial manipulator. To provide the possibility of reuse, system manipulation primitives are introduced which implement parts of the system each. To give the execution a appropriate order it is organized by a petri net.

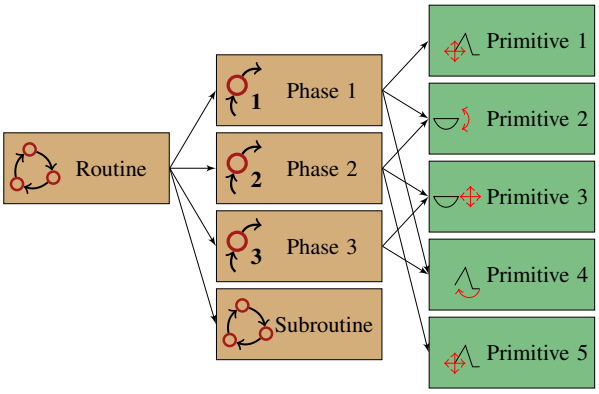


Figure 1. The structure of the behavior-based control architecture

## 3 Architecture

With the presented architecture the previously explained subdivision of the trenching process into subtasks is targeted. Additionally, the implementation of the subtasks should be exchangeable and on the lower level primitives are used to provide a good re-usability for control laws in the different phases and for other tasks.

### 3.1 Structure

In general, the architecture structures the implementation hierarchically. So the overall control system solving the task is called a routine. A routine consists of phases that represent the subtasks a routine can be split into. In every phase a set of primitives will be activated. It is possible that a primitive is actually used by multiple phases. This structure is graphical shown in figure 1.

To adapt the implementation of a subtask, for example to switch from mass excavation to precise trench shaping, it is only necessary to exchange one single phase while the remaining implementation can stay the same. Besides the option of using a set of primitives directly, it is possible to use a subroutine as implementation of a phase.

### 3.2 Phase

Subtasks, that a routine can be split into, are represented as phases. This structure element is used for arbitration; a phase should become active if specific criteria of the system state are fulfilled. These can include measured data from the environment like the actual trench shape as well as the machine state, for example the actual angle of the bucket according to the machine frame. During the development process for the implementation of the trenching task, it appeared that a phase should become active if some conditions are fulfilled and should be held active while some other criteria are in the appropriate range. Targeting this, the developed architecture defines a set of preconditions that has to be fulfilled at the moment of phase activation. Additionally, an invariant, represented as a set of conditions, has to be fulfilled during the phase is running.

For the trenching process, four phases are defined according to the subtasks introduced before: *move to start, digging, transport to dumping and dumping*. In the *move to start* phase the bucket should be controlled to the start position of the next digging cycle. This is the position at which the bucket should begin to retrieve the next portion of soil out of the trench. To determine this position, rating functions are used which operate on grid maps of the desired and actual surface. These functions are introduced in [15]. Additionally to reaching the position, this first phase should curl the bucket to an angle which allows to break into the soil in an appropriate manner. During the *digging* phase, the bucket should move through the soil to scoop up as much soil as possible. This leads to a trajectory which moves the bucket from an open angle to a closed one. Also, the bucket should move into the direction of the excavator base. When the bucket is filled, the soil has to be brought to the dumping position. This can be a target pile or the loading area of a truck, for example. The transfer to this position is done in the *transport to dumping* phase. Arrived there, the material has to be dumped out of the bucket. For this, the *dumping* phase is responsible. When the bucket is empty again, the digging cycle should start with the next iteration.

To formulate the conditions, mainly the pose of the bucket given in Cartesian space $\Pi_{bucket}$ is used. This includes the bucket position $P_{bucket}$ and its orientation $\Omega_{bucket}$.

$$\Pi_{bucket} = (P_{bucket}, \Omega_{bucket}) \tag{1}$$

While digging and dumping, the main movement should be a curling of the bucket to gather soil into it or drop it. Addressing that, the angle of the bucket according to the basis frame of the arm $\theta_{bucket}$ is used here to measure the progress. Additionally, some target or threshold values are defined to formulate criteria for fulfillment of the conditions. These are $\Pi_{start}$ for the desired start pose of the next digging process, $\theta_{lift}$ for the angle that should be used while lifting the bucket out of the trench, $P_{dump}$ for the desired position of the bucket while dumping and $\theta_{dump}$ for the bucket angle in which it is fully open. In table 1 the used phases with their preconditions and invariants are listed. With a clever definition of the preconditions and invariants, it is possible to let the execution of the phases overlap appropriately.

| phase | precondition | invariant |
|---|---|---|
| move to digging | $\theta_{bucket} \leq \theta_{dump}$ | $\Pi_{bucket} \not\approx \Pi_{start}$ |
| digging | $\Pi_{bucket} \approx \Pi_{start}$ | $\theta_{bucket} \leq \theta_{lift}$ |
| move to dumping | $\theta_{bucket} \approx \theta_{lift}$ | $P_{bucket} \not\approx P_{dump}$ |
| dumping | $P_{bucket} \approx P_{dump}$ | $\theta_{bucket} > \theta_{dump}$ |

Table 1. The phases of the digging cycle with their preconditions and invariants

### 3.3 Motion Primitive

On the lowest level of the presented architecture, primitives are used. They implement local control laws which generate velocity vectors for the TCP. Finally, these local velocity vectors are combined to an overall control vector. This will be given to the low-level system which calculates the control commands for the machine. A primitive can generate a fixed velocity as well as a dependent one which is calculated based on a sensor data input vector.

Primitives will be activated during the phases. It is possible that one phase uses multiple primitives and one primitive can be used by multiple phases. This improves the re-usability of the structure elements of the system. The standard case of every phase is that a set of primitives is active in parallel.

To solve the trenching task the following primitives are used:

**Rotate**: This primitive is used to rotate the swing to a given target angle. To do this, it generates a rotational movement around the z-axis. The desired velocity is proportional to the angular distance to the target angle. Here, the maximum angular velocity in the experiments on the machine were 0.25 $\frac{\text{rad}}{\text{s}}$.

**Curl bucket**: Similar to the first primitive, it generates an angular velocity which curls the bucket to a given target angle. This is a rotation around the y-axis of the backhoe plane coordinate system (maximum velocity: 0.5 $\frac{\text{rad}}{\text{s}}$).

**Move to position**: The target of this primitive is to move the bucket to a given position in the two-dimensional backhoe plane. To reach this, a 2D velocity vector is generated which is directed to the target point and has a length proportional to the remaining distance. In the experiments, this movement is done with a maximum velocity of 1 $\frac{\text{m}}{\text{s}}$.

**Close bucket**: This one curls the bucket with a configured fixed angular velocity in the closing direction (velocity: 1 $\frac{\text{rad}}{\text{s}}$).

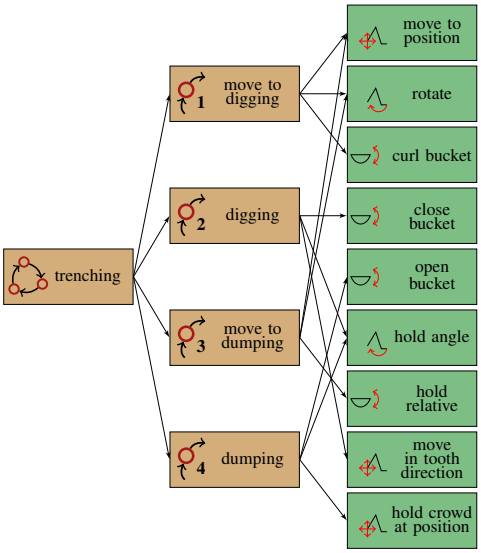


Figure 2. The architecture of the trenching process

**Open bucket**: Similar to the previous primitive, this curls the bucket in the opening direction (velocity: 1 $\frac{\text{rad}}{\text{s}}$).

**Hold angle**: With this, the backhoe plane should be held at a fixed angle. This is done by generating an angular movement around the z-axis of the arm base coordinate system if the actual angle differs from the configured one. Here, the value of the generated velocity is proportional to the angular error.

**Hold crowd at position**: By activating this primitive, the crowd joint will be held at a given position on the backhoe plane. This is done in a similar way as in the primitive explained before but here a linear velocity will be generated according to the error instead of an angular one.

**Move in tooth direction**: From this, a velocity vector for the TCP with a fixed length is generated which targets into the direction of the tooths of the bucket. To be able to generate a vector which targets into the right direction, the mounting angle of the tooths has to be configured. Additionally, the actual bucket angle according to the machine coordinate system is used to generate the movement (velocity: 1 $\frac{\text{m}}{\text{s}}$).

**Hold relative**: This primitive generates an angular velocity which holds the bucket in a configured angle relative to a given reference plane.

In figure 2, the whole architecture for the trenching process is shown.

# 4 Implementation

For the implementation of the architecture, a framework is used which defines special modules that can be used together as a behavior-based control network. This is introduced in [10] as IB2C. A short introduction to this is also given in the next section.

## 4.1 IB2C

In IB2C, a basic module type is defined that can be used to compose a net which makes a control system for robots. Such a node has a set of well-defined inputs and outputs, figure 3 shows a schematic representation. An IB2C module is defined with the three-tuple $B = (f_a, f_r, F)$ where $r = f_r(\iota)$ is the target rating function, $a = f_a(r, \iota)$ the activity function and $\vec{u} = F(\vec{e})$ the transfer function. Additionally, each module has a set of input and output signals which include the meta data: Activity $a \in [0; 1]$, target rating $r \in [0; 1]$, stimulation $s \in [0; 1]$ and inhibition $i \in [0; 1]$ as well as the input vector $\vec{e} \in \mathbb{R}^m$ and the output vector $\vec{u} \in \mathbb{R}^n$. With the transfer function $F$, the output vector will be calculated from the input vector according to the internal control law. Using the meta data, the influence of the module in the whole system is calculated and coded in $a$. This is done by calculating the activation $\iota$ of the module from the meta data inputs. In difference to the original introduction of IB2C, in the presented approach the following equation is used to calculate the activation.

$$\iota = \min(s, 1 - i) \tag{2}$$

With this value, the target rating $r$ can be calculated by the user-defined function $f_r$. It should express how much the calculated output vector $\vec{u}$ should influence the system if the module is fully active. From this values, the activity $a$ of the module can be calculated.

$$a = \min(\iota, r) \tag{3}$$

With this equation, it can be ensured that the activity has in average case the value of the target rating $r$ and at maximum the value of the stimulation $s$. To combine the control vectors generated by different IB2C nodes, a fusion module is used. For the combination, it uses the meta data of the corresponding modules. From this, the fusion module calculates a new output vector in which every input vector is included according to its corresponding activity and target rating. To perform this, different fusion methods can be used; for example a maximum function or a weighted sum. If the maximum function is used, the output of the module with the highest activity is directly used as the output of the fusion module. By using the weighted sum solution the input vectors of the fusion are multiplied with their corresponding activities and summed up.

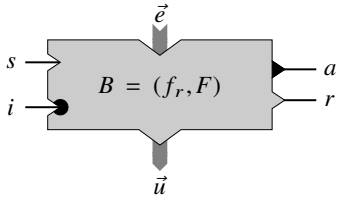


Figure 3. The basis IB2C module

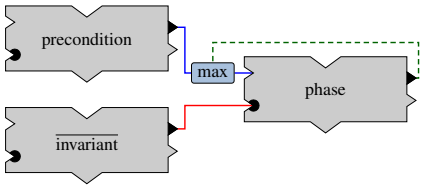


Figure 4. Phase arbitration with IB2C

## 4.2 Phase Arbitration

For the implementation of the conditions used for the arbitration of the phases, IB2C modules are used. Also, the phase itself is implemented as such a behavior-based node. It is not possible to use the condition modules as simple stimulation for the phase, because it should become active if the precondition is fulfilled, but stay active until the invariant is not fulfilled anymore. So it should not become active if the invariant is fulfilled but the precondition is not and it should stay active if the criteria for the precondition were in the right range but are not anymore. Addressing this problem, the pattern shown in figure 4 is used for the arbitration. To hold the maximum activation given by the precondition, a self-stimulation of the phase module is used. Without inhibition, this leads to an activation for the phase $\iota_{phase_{i=0}}$ which is the maximum of the previous activities given from the precondition. The set of the previous activities of the precondition is defined as $A_{pre} = \{a_{pre_0}, a_{pre_1}, ..., a_{pre_t}\}$ where $t$ is the actual time step.

$$\iota_{phase_{i=0}} = \max(A_{pre}) \tag{4}$$

When the work is done, the phase has again to be deactivated. For this the invariant is used which is connected to the inhibition input of the phase module. With this implementation, the activity of the invariant lowers again the activation of the phase. So the invariant condition has to be implemented in a way that it becomes active if the invariant is violated. This pattern allows also to implement the condition with fuzzy activity functions which lead to a partial activation of the phase.

## 4.3 Conditions

In the trenching implementation, two kinds of conditions are used. One is to proof if an input angle is in a

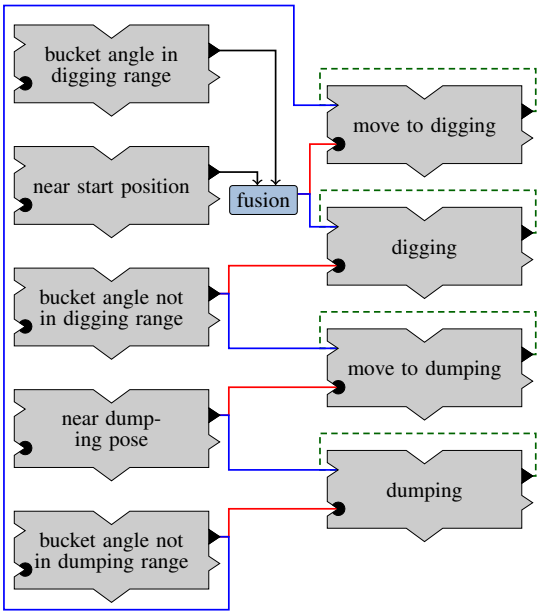


Figure 5. The phase arbitration of the trenching process

configured range and the other one is to test if a given position is in the near or behind a plane defined in Cartesian space. For the first condition type, three parameters are defined. With the start angle $\theta_s$ and the end angle $\theta_e$ an angular area is defined in which the condition should be fully active. The third parameter $\theta_a$ defines an angular distance to the start angle in which the condition starts to become active. This leads to the following activity function for the condition. Here $\theta_{in}$ is the measured input angle which should be tested.

$$r_{cond_a} = \begin{cases} 1, & \text{if } \theta_s < \theta_{in} < \theta_e \\ \frac{\theta_s - \theta_{in}}{\theta_a}, & \text{if } (\theta_s - \theta_a) < \theta_{in} \leq \theta_s \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

With the second condition, a given input position $P_{in}$ is tested if it is in the near or behind a plane. It starts to become active if the distance $d_p$ of $P_{in}$ to the plane is smaller than the activity distance $d_a$ and fully active if the input position moves behind the plane. From this the activity function of this condition $r_{cond_p}$ can be formulated as given in the following equation.

$$r_{cond_p} = \begin{cases} 1, & \text{if } d_p < 0 \\ \frac{d_p}{d_a}, & \text{if } 0 \leq d_p < d_a \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

To test if the bucket has reached the start position or the dumping position respectively, the second condition type is used. The target plane is defined with the target position as supporting point and a normal which directs into

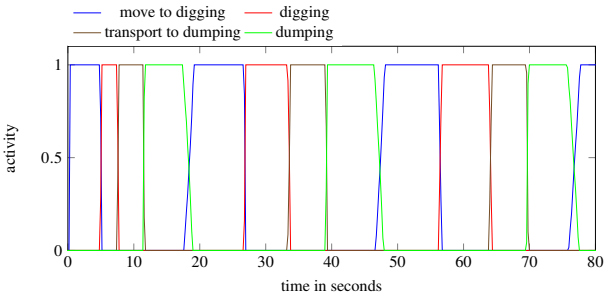


Figure 6. The activity of the different phases while trenching

the starting position of the movement. Using the angle checking condition, three modules are implemented. One for checking if the bucket angle is in the right angle range for digging, another one if it is not in this range anymore and a third one which checks if the angle has passed the right angle for dumping. In figure 5 the network for the phase arbitration of the trenching process is shown.

### 4.4 Adaption Layers

With the described method, the trajectories are generated in a very free way. To ensure given constraints, the generated velocity vector has to be adapted according to the environment and the desired target trench. This is done by two additional behavior-based subnets which implement an adaption of the velocity vector.

To prevent the bucket from hitting an obstacle, which also includes the remaining soil of the ground, an IB2C network is developed which adjust the velocity vector of the TCP if the distance to the next object is too small. For this a grid map is used which represents the measured environment by containing a height information in every cell, representing the surrounding. If the cell in the configured safety distance in the desired moving direction contains a value that is higher as the current z-position of the bucket, the net blocks the moving in this direction.

Similar to the obstacle avoidance there is a net implemented which prevents the bucket to hit the borders of the desired target trench. It works in the same way as the obstacle avoidance but the gridmap which is used for the adaption does not contain measured obstacles, it contains a representation of the desired trench shape.

## 5 Experiments

To proof the performance of the developed approach, an experiment was executed. For this, the presented approach for autonomous trenching was implemented on a *John Deere 410J TMC* backhoe loader. This is a machine with an overall weight of around 8 t. Additionally to the

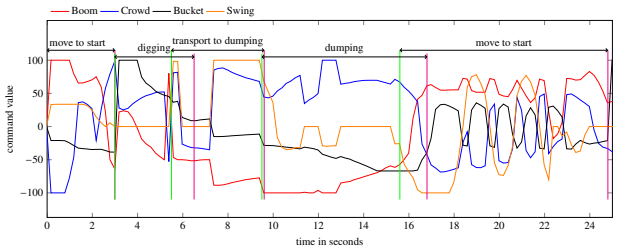


Figure 7. The generated joystick commands while trenching

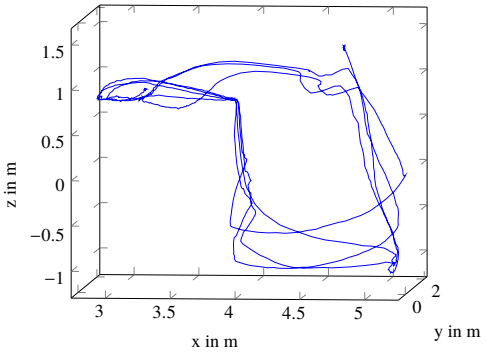


Figure 8. The trajectory of the bucket while digging

real machine, a simulation environment was used for testing. To do this, a realistic simulation model of the *John Deere* backhoe loader was built up. With the usage of the simulation platform V-Rep (see [11]), it was possible to represent the dynamics of the vehicle. For the experiment and the results presented in the following part the real machine was used. In the implemented solution a target trench could be configured with length, depth and starting position. Here, a length of 1 m and a depth of 1 m are used. The starting position was given to (5 m, 0 m) which represents a point in the local coordinate system of the used machine. The origin of this coordinate system is located in the swing joint which is basically the origin of the backhoe arm. It could be shown that the implemented control system was able to dig the configured trench. In figure 8 the trajectory of the bucket recorded while this process is plotted. Here, it can be seen that the resulting trajectory is not as smooth as it would be if a well adjusted industrial robot would execute it with a closed loop control. But these uncertainties are originated from the constant adaption of the behavior-based system. For example at the boundaries of the desired trench dimensions a behavior becomes active which adapts the control vector to keep the bucket inside this boundaries. Another adaption behavior prevents the bucket from colliding with the ground and other obstacles. Furthermore, for the trenching process the small uncertainties are not relevant because they are not detectable in the end result. Additionally, the activity of the phases implemented with the presented approach was recorded. This data is visualized in figure 6. There, it can be seen that one excavation cycle lasts around 20 s to 30 s, which is approximately in the same range as Du et al. have measured with human experts in [2]. Additionally, with the architecture used here, it is even possible to overlap the different phases. This can also be seen in the data showed in figure 6.

The machine used for the experiments can be controlled with joystick commands which are directly translated into flow rates for the hydraulic cylinders. These commands are in a range between −100 and 100. Using the solution presented here, the calculated control velocity vector is converted to appropriate joystick commands with the help of the inverse kinematic. It is possible to calculate them directly because the machine which is used here has only four joints which leads to a uniquely defined system. So four command values are generated, one for each joint. Figure 7 shows the commands generated by the control system during the trenching process. In this figure additionally, the active phases are marked.

## 6 Conclusion and Outlook

With the presented approach, an architecture is defined which can organize a control system in phases. For the arbitration of the phases, conditions are used. Because of this structure the system is similar to a finite state machine. But with the help of the used behavior-based implementation the conditions have fuzzy like activity functions. So it is possible that multiple phases are active at the same time. This leads to a smoother state change as in a standard FSM. Additional this adds a property to the control system which let it more behave like expert human machine operators.

Although with the used behavior-based method, it is possible to build up reactive systems which can adapt in a proper way to high variety in the environment. To target new requirements, it is very easy to add additional primitives which can implement local control systems to address additional problems. Overall the usage of this primitives provides the possibility of re-using the same control law to solve similar subproblems in different tasks.

In the future the system will be extended to solve a more global task like building up a larger excavation pit. This will also include a transfer of the whole machine to have a larger range. A possibility to do this is also given with the presented architecture with the definition of subroutines which can be used as phase implementation. So the presented local trenching process can be such a subroutine in a larger control system. To be able to give the trench a better shape it should be possible to switch to a

more precise digging technique which is more optimized for shaping than for mass excavation. Then there are at least two possibilities for digging available which makes it necessary to choose the appropriate one for the given requirements. Targeting this, a planning system should be designed in future work. This should be as autonomous as possible. So, the first step towards this can be to add some properties to the primitives and phase implementations which can help the planning system to choose the right solution for solving the task. For example, one possibility for such a property can be the desired digging shape which is expected as a result of the appropriate module.

## References


[1] David A. Bradley and Derek W. Seward. The development, control and operation of an autonomous robotic excavator. Journal of Intelligent and Robotic Systems, 21(1):73–97, January 1998.

[2] Yu Du, Michael C Dorneich, and Brian Steward. Virtual operator modeling method for excavator trenching. Automation in Construction, 70:14–25, 2016.

[3] Ahmed Elezaby, Mohamed Abdelaziz, and Sabri Cetinkunt. Operator model for construction equipment. In Proceedings of the International Conference on Mechtronic and Embedded Systems and Applications (MESA), pages 582–585. IEEE, 2008.

[4] Marco Ewerton, Guilherme Maeda, Gerhard Neumann, Viktor Kisner, Gerrit Kollegger, Josef Wiemeyer, and Jan Peters. Movement primitives with multiple phase parameters. In Robotics and Automation (ICRA), 2016 IEEE International Conference on, pages 201–206. IEEE, 2016.

[5] Sung-Keun Kim, Jongwon Seo, and Jeffrey S Russell. Intelligent navigation strategies for an automated earthwork system. Automation in Construction, 21:132–147, 2012.

[6] Horst König. Maschinen im Baubetrieb. Springer, 4. edition, 2014.

[7] Tobias Luksch, Michael Gienger, Manuel Mühlig, and Takahide Yoshiike. A dynamical systems approach to adaptive sequencing of movement primitives. In Proceedings of the 7th German Conference on Robotics (ROBOTIK), Munich, Germany, May 21-22 2012.

[8] Guilherme J Maeda and David C Rye. Learning disturbances in autonomous excavation. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pages 2599–2605. IEEE, 2012.

[9] Guilherme J. Maeda, David C. Rye, and Surya P. N. Singh. Iterative autonomous excavation. In Kazuya Yoshida and Satoshi Tadokoro, editors, Field and Service Robotics: Results of the 8th International Conference, pages 369–382, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[10] Martin Proetzsch, Tobias Luksch, and Karsten Berns. The behaviour-based control architecture iB2C for complex robotic systems. In Proceedings of the 30th Annual German Conference on Artificial Intelligence (KI), pages 494–497, Osnabrück, Germany, September 10-13 2007.

[11] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: a versatile and scalable robot simulation framework. In Proc. of The International Conference on Intelligent Robots and Systems (IROS), 2013.

[12] Patrick Rowe and Anthony T. Stentz. Parameterized scripts for motion planning. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robotic Systems, volume 2, pages 1119–1124, Grenoble , France, September 7-11 1997.

[13] P.S. Rowe. Adaptive motion planning for autonomous mass excavation. PhD thesis, Carnegie Mellon University, 1999.

[14] Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In Adaptive motion of animals and machines, pages 261–280. Springer, 2006.

[15] Daniel Schmidt. Shaping the Future - A Control Architecture for Autonomous Landscaping with an Excavator. RRLab Dissertations. Verlag Dr. Hut, München, 2016. `http://www.dr.hut-verlag.de/978-3-8439-2816-8.html` ISBN-13: 978-3-8439-2816-8.

[16] Jongwon Seo, Seungsoo Lee, Jeonghwan Kim, and Sung-Keun Kim. Task planner design for an automated excavation system. Automation in Construction, 20(7):954–966, 2011.

[17] Ingo Weidauer, Daniel Kubus, and Friedrich M Wahl. A hierarchical extension of manipulation primitives and its integration into a robot control architecture. In Robotics and Automation (ICRA), 2014 IEEE International Conference on, pages 5401–5407. IEEE, 2014.