Establishing Reliable Robot Behavior using Capability Analysis Tables

Victoria Edwards Loy McGuire
Naval Research Laboratory, Washington DC, USA
Distributed Autonomous Systems Section, Code 5514
victoria.edwards@nrl.navy.mil loy.mcguire@nrl.navy.mil

Signe Redfield
Naval Research Laboratory, Washington DC, USA
Robotics and Machine Learning Section, Code 8234
signe.redfield@nrl.navy.mil

Robots are often so complex that one person may not know all the ins and outs of the system. Inheriting software and hardware infrastructure with limited documentation and/or practical robot experience presents a costly challenge for an engineer or researcher. The choice is to either re-build existing systems, or invest in learning the existing framework. No matter the choice, a reliable system which produces expected outcomes is necessary, and while rebuilding may at first appear easier than learning the system, future users will be faced with the same choice. This paper provides a method to allow for increased documentation of the robotic system, which in turn can be used to contribute in overall robot reliability. To do this we propose the identification of a robot's core behaviors for use in Capability Analysis Tables (CATs). CATs are a form of tabular documentation that connect the hardware and software inputs and outputs to the robot's core behaviors. Unlike existing methods, CATs are flexible, easy to build, and understandable by non-expert robot users. We demonstrate this documentation method with an experimental example using an Unmanned Aerial Vehicle (UAV).

1 Introduction

Autonomous systems can have many hardware components and countless lines of code. It takes time to work through all the raw information about the systems' inner-workings, and in many instances, documentation is limited. Despite this complexity, it has been demonstrated that some robots are able to function reliably in certain settings, but only when the environment is simple, or the level of autonomy is reduced. Reliable, for the purposes of this paper, means that the output of the robot meets the expectations of what the robot should be doing consistently. Improved reliability, is the identification or elimination of undefined output response for a provided input to the robot. When handling a new robotic system, it is easy to pass an input to a robot that could produce an unexpected behavior, especially with little documentation or intuition for what the robot should do.

One way to transfer knowledge to a new user is to spend time with the developer or former caretaker of the robot to learn from that person's experience. Often, this does not or cannot happen. Because of the increasing desire for reliable robotic systems, it has become necessary to develop tools to better facilitate these types of transitions. There are many existing tools to ensure reliability of a robotic system, for example, finite state machines (FSMs) [6], formal methods [27], Domain Specific Language (DSL) [30], and Systems Modeling Language (SysML) [38]. However, each of these tools require expert knowledge of either the robotic system or the methodology to generate useful representations of the system.

R.C. Cardoso, A. Ferrando, D. Briola, C. Menghi,
T. Ahlbrecht (Eds.): First Workshop on Agents and
Robots for reliable Engineered Autonomy 2020 (AREA'20).
EPTCS 319, 2020, pp. 19–35, doi:10.4204/EPTCS.319.3

Tools and methods need to be developed to enable easier transfer of knowledge about existing robot capabilities to new users which leverage existing understanding of how the robot works. A step towards achieving this is to augment documentation to capture not only the software components but also how that software integrates with hardware. First, it is necessary to identify the core behavior as a way to establish a performance baseline for the robotic system. **Core behaviors** are the high level description of what the robot should be able to do reliably with it's current software and hardware infrastructure. The core behavior is meant to capture the intuition of an expert user, and to provide a level of abstraction which is easily understood by a non-expert user. For example, given a ground robot with no sensing, the core behaviors are Drive and Charging. Core behaviors are similar to states in FSMs or modes in formal methods, however, core behaviors are a flexible representation at different levels of abstraction of the system, and do not require the same level of detail required in FSMs or formal methods. While the identification of core behavior may seem trivial, going through the steps of identifying the core behavior ensures that a baseline of performance can be established for the robot based on user intuition. Likewise, having core behavior identified before a transition of technology can help alleviate problems when trying to use an existing system.

For this work, we use a Capability Analysis Table (CAT) [36], which is a documentation tool that captures how the core behaviors of the robotic system corresponds to the hardware and software of an existing autonomous system. The CAT framework is built by a human user of the autonomous system, either during or after behavior development. Once the CAT is developed, it can be used to capture the constraints on inputs to the system, isolate regions of failure, and provide concise representation of the system to new users. To build and use a CAT, a spreadsheet tool is needed, along with knowledge of the inputs and expected outputs of the system (through documentation, investigation of hardware and software, and observation of system performance in the world). CATs will not change the complexity, creation, or design of the system, but can augment documentation to improve the usability of the system.

The main contributions to this work are:

- A method for identification of core behaviors and construction of CATs to aid in establishing a reliable robot.
- A demonstration of how CATs are used to handle unreliable robot behavior.
- Experimental results establishing core behavior, building CATs, and rooting out unreliable behavior for a Quadrotor.

The rest of the paper is as follows: in Section 2 we discussion background material for this paper. In Section 3, we describe the methodology for identifying core behaviors and building a CAT. In Section 4, we demonstrate the proposed methodology for an experimental platform, and in Section 5 we discussion different aspects of the proposed approach. Finally, we conclude and discuss future steps in Section 6.

2 Related Work

There are some existing tools to provide insights into the system's core behavior, which in other areas can also be called states [6, 32] or modes [27]. For example, Finite State Machines (FSMs) [25] are a representation of states and transitions between system states. There are many tools that will test FSMs [6], help build FSMs from state assignment [8], or represent functions as FSMs [3]. Another tool which is closely related to FSMs are Petri nets, which are a mathematical set of rules to capture distributed or asynchronous processes [13, 29, 32]. Fernandez et al. use Petri nets as the foundation for their robotic systems [12]. In the proposed method, they allow a human user to build a Petri net

as the input commands to a robot, and provide tools to handle issues as the robot interacts with the world. Finally, system guarantees and logical models representing a system are captured using formal methods [4, 10, 27, 33]. More recently, Kress-Gazit presented a survey on how formal methods are used with robots [20] and a survey on the challenges of automatic code synthesis for reactive systems [21].

These tools all aid different aspects of making a reliable robot, and each captures a form of core behavior for the robot. FSMs capture the robot states, Petri nets capture asynchronous behavior, and formal methods represents systems logically, however the connection of the core behavior to the inputs and outputs of the system as they relate to the hardware and software of the system is missing. The CAT supports connections between information available to the system and the mutable variables of that system, which specifically allows for the representation of hardware. Additionally, CATs combine data relevant to all of these methods into a concise format, providing the user with a new representation for reasoning over autonomous system performance.

A tool to help identify how core behaviors relates to the hardware and software of the system is Systems Modeling Language (SysML). SysML is a visual semantic framework to build models of complex systems that is an extension of Unified Modeling Language (UML) [7, 15], specifically, SysML breaks down systems into four components: structures, behaviors, parameters, and requirements [38]. These models have been used to verify systems and used with different formal method approaches [18, 22, 31]. SysML models are a powerful modeling infrastructure to model all the details of the system and require expert knowledge to build, while the CAT is a flexible methodology which is meant to benefit any user of a specific platform at certain levels of abstraction.

A similarly related area is Domain Specific Languages (DSL), which are defined as represnetations with specific vocabulary to be used by domain experts and provide some component of machine readable syntax [30]. Several DSLs have been built for robotics systems [9, 14, 28]. In addition, a DSL has been coupled with design processes which can improve the verification and re-usability of a robotic system [39]. Finally, a DSL has been used to map robot software to formal specification techniques. These approaches are modular in nature and provide improved insights on how to design and build systems, but do not focus on documentation tools for existing systems. Likewise, these tools require the user to have domain knowledge, while CATs are intended to be a tool that can be used with limited domain knowledge. DSLs can take into account hardware constraints for a robotic system, but often they focus on the verification of the software. Because robotic systems exist with both hardware and software it is critical to start building tools which consider the failure of both components and the potential for the failure at the intersection of hardware and software. A survey on verification of autonomous systems presents more resources on formal methods, modeling tools, and other representations to try and assure robot behavior [26]. It is an active area of work to build a formal representation of CATs. This work is focused on a documentation tool to enable more reliable use of inherited robotic systems.

In addition to existing tools, capability analysis has been used in several fields including in statistics as the study of performance of a system [40] and in computing metrics for power consumption [23]. In statistics, data is taken from a system, and a variety of metrics are computed (e.g., mean, standard deviation, and range) to statistically quantify performance. This analysis is organized in a spreadsheet of data cells called capability charts [5, 40], or capability indices [19]. Alternatively, capability metrics for power consumption, are automatic tools developed by Landgren et al. [23,24] to allow for faster output of performance and conditions that the network will perform under. Unlike the charts in capability analysis for power and system performance, the CATs discussed in this work is a tabular representation of core behavior, while connecting system inputs and outputs.

Finally, we will use CATs to perform analysis similar to root cause analysis (RCA) or fault tree analysis for a robotic system. RCA is the identification of what the problem is, how the problem hap-

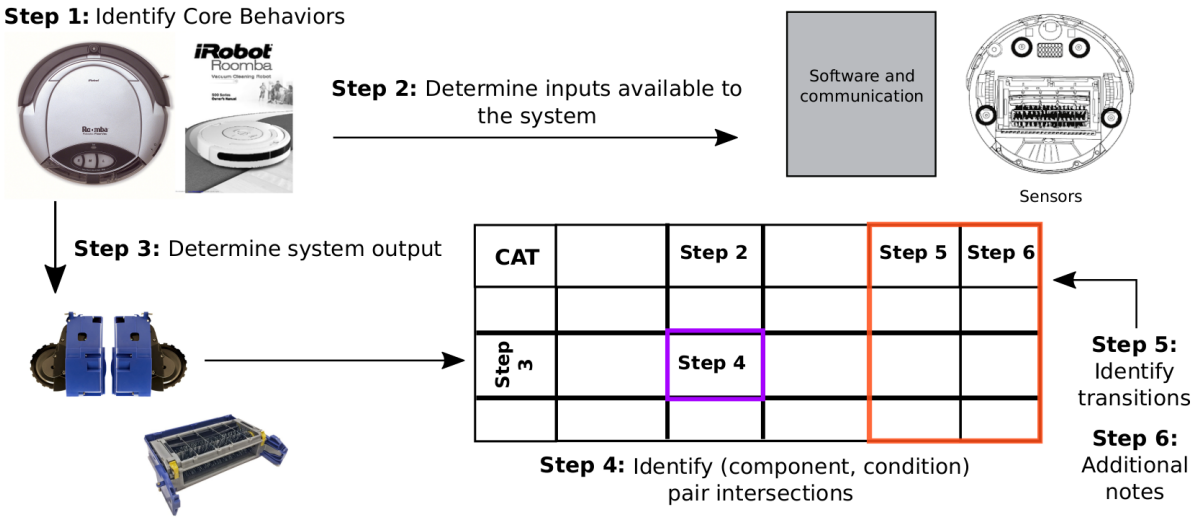


Figure 1: There are 6 main steps to building a CAT, each of which correspond to text in Section 3.2. The Roomba images in this figure are from the iRobot website [1].

pened, and why the problem happened, and this information is represented in maps or causal factor charts [11, 37]. Fault tree analysis is a methodology to determine the faults that may happen considering how different combinations of software, hardware, and human inputs may disrupt the system [2, 35]. A tool more commonly seen in robotics to generate explanations of behavior was proposed in Raman et al., who used a temporal logic approach to generating explanations of robot behavior [34]. Likewise, Hayes et al. provided explanations of robot behavior to aid in human interactions with the robot [16]. We will use CATs to provide explanations of unreliable behavior, and provide a systematic way to isolate why the undesirable behavior is occurring. It is more likely to see unwanted behavior when transitioning hardware and software, which is why an easy to use tool like CATs is critical to root out the negative behavior.

3 Methodology

In this section, we will describe the identification of core behaviors, a methodology to develop CATs, and outline how CATs can be used to isolate unreliable robot behavior. The example used throughout the Methodology will be of an iRobot Roomba [17].

3.1 Identifying Core Behaviors

A core behavior is the expected behavior of the system from basic inspection, without a need for large amounts of detail about how the behavior is implemented. Identifying core behaviors does not require expert knowledge of the system, and as the users understanding of the system evolves the core behaviors can be refined. Core behaviors for some robots may be obvious, and in other cases may require a more in depth exploration of the system or consultation with an expert. Sources of information which can help shed light on the core behaviors are: code documentation, hardware manuals, inspection of the code and hardware components, consulting an expert, and, when necessary, experimental trials. Of these options,

running experimental trials is the most dangerous, especially when proper inputs and outputs are not identified for the system. For the Roomba example, we used a specification manual to identify the core behaviors [17].

Core behavior is identified by understanding the basic functionalities of the robot. Questions should be asked, such as: *If the robot is turned on, what will the robot do? Given this input what output is expected?* Identifying core behaviors captures the users intuition of what the robot should do, and allows for a standard representation. Additionally, core behaviors abstract away the need to have intimate knowledge of the robot's subsystems. The key at this level of abstraction is to select behaviors which highlight the functions that must persist for the robot to do anything meaningful in the world. The early generation the Roomba has 3 core behaviors: Drive, Clean, and Charging. These are the three basic states of the robot's expected overall behavior.

Once identified, a set of baseline experiments can be established to ensure the persistence of the core behavior. In doing so, the robot has a performance benchmark which can be tested and evaluated before more advanced capabilities are added. This can enable principled experimentation of new algorithms and more reliable robot behavior. Establishing this baseline is critical to knowing if the system is performing reliably, and can be used throughout the lifespan of the robot.

For the Roomba example, a set of baseline experiments which evaluate the core behaviors are:

- the robot increases power and does not move when charging,
- the robot drives with no warning light,
- the robot vacuums with no warning light.

Each of these experiments evaluates specific underlying components of the core behaviors which we will identify in Section 3.2

3.2 Building a Capability Analysis Table

In this section, we will define a methodology for how to build a Capability Analysis Table (CAT) for an existing system. The same information used in the identification of the core behaviors can be used to build a CAT. Figure 1 is a basic outline of a CAT. For more details on how to build a CAT for design purposes, the reader is directed to [36].

Step 1 in building a CAT is to identify the core behaviors of the system, outlined in Section 3.1. We will build an example CAT for the Roomba, Figure 2. The Roomba core behaviors were identified to be Drive, Clean, and Charging.

Step 2 is to identify the inputs to the system, which includes both inputs from sensors, inputs generated within software, and communication which is provided to the robot. These inputs make up the column headers, and are named by the user. Consistency and clarity in naming will aid in future use. The main goal is a documentation aid which incorporates both hardware and software constraints. The Roomba has a variety of inputs, but the key required inputs for the core behaviors defined above are: battery level, warning light, type of cleaning, and velocity. Note that many more inputs might be available, but at this level of abstraction all of these inputs are necessary in some way for the core behaviors to exist.

Step 3 is to identify the expected outputs or actions that are expected at the core behavior level of the system. These outputs make up the left-hand side row headers. In systems which have both hardware and software components, it is important to consider how outputs are represented in software for the different core behaviors. For the Roomba example: Drive has an output of wheels moving, Clean has an output of pick up dirt, and Charging has an output of increasing the power level.

Roomba CAT											
	Battery Level		Warning Light		Type of Cleaning		Velocity		Change in State		Notes
	Condition	Component	Condition	Component	Condition	Component	Condition	Component	Condition	Component	
Wheels move	has value	Drive	Not active	Drive			Available	Drive	Previous Component = Charging	Drive	Drive controls the wheel motors
Pick up dirt	has value	Clean	Not active	Clean	Available	Clean			Previous Component = Charging	Clean	Clean controls the vacuum behavior
Increase power level	has value	Charging					None	Charging	Previous Component = Clean	Charging	Robot placed at a power station

Figure 2: Given an early iRobot Roomba system based on information from [17], we built a CAT which describes the robot's core behavior. The three core robot behaviors are: Drive, Clean, and Charging.

Step 4 is taking the defined inputs and outputs that have been defined and filling in the cells of the CAT. To do so, start with considering a corresponding output and which core behavior contributes to that output. Each column is composed of a (Condition, Component) pair, which represent the constraints the input may require for the specific core behavior. Along each column put the core behavior in the Component grid cell, and the corresponding constraint in the Condition grid cell. For each row, every column with an entry must meet the specified condition to see the desired output. Blank spaces along a row represent inputs that are not necessary for the core behavior to generate the desired output. CATs are a tabular representation of a set of conditions which can be represented logically.

For example consider the Roomba CAT in Figure 2. Let us use the core behavior Drive and the output wheels move. Along the row, Drive will be placed in the Component grid cell for the following columns: Battery level, warning light, and velocity. This implies that for the output, wheelsMove, all of these conditions must be true. This can be represented logically as:

$$\text{Drive} = (\text{batteryLevel} \wedge \text{warningLight} \wedge \text{velocity} \wedge \text{changeInState}) \implies \text{wheelsMove} \tag{1}$$

This type of logical representation can be built for every row, where all the input columns that intersect the row with an entry are a component, condition pair which is required by a logical AND condition for the output to exist. To achieve a logical OR statement, one output will have two or more sub-rows, which is seen in the more complex CAT in Figure 7. Each column with a component entry along a sub row will be combined using the logical AND as done in Equation 1. The resulting sub row statements will be combined for a single output using the logical OR.

Step 5 is to identify the conditions which will transition the system into the core behavior for that row, similar to a tabular FSM. Transitions between states may be dependent on time or it may be possible to have asynchronous execution of behaviors. Timing constraints can be handled in the condition sub-column, and it is a direction of future work to consider how CATs can be used in real time incorporating these complexities.

Step 6 is the final step, and it is the process of adding notes to the final column to provide any additional information to the user. For the Roomba example, the CAT in Figure 2 is filled in to reflect the transitions between behaviors and a discussion of each core behavior. Once built, CATs can be used as a documentation aid to help transition robot capabilities to new people, maintain up to date system

Clean CAT											
	Dirt Detector (left or right)		**Type of Cleaning**		**Battery Level**		**Incoming Command**		**Change in State**		**Notes**
	Condition	Component	Condition	Component	Condition	Component	Condition	Component	Condition	Component	
Vacuum Motor spins	Move toward highest value	Spot	hit dirty areas more	Spot	has value	Spot	message in = Spot	Spot	Previous Component = General	Spot	spot is extra vacuuming at a region
	vacuum all area	Max	vacuum for all battery time	Max	Use all power available	Max	message in = Max	Max	Previous Component = Charging	Max	Max is maxing out the battery to vacuum an area
	vacuum all area	General	vacuum normal cycle	General	has value	General	message in = General	General	Previous Component = Charging	General	General is just a normal vacuum cycle

Figure 3: The Clean Roomba core behavior decomposed into a CAT.

Drive CAT													
	Bump Detection		**Battery Level**		**Cliff Sensor**		**Warning Light**		**Velocity**		**Change in State**		**Notes**
	Condition	Component	Condition	Component	Condition	Component	Condition	Component	Condition	Component	Condition	Component	
1 wheel spin forward, 1 wheel spin backward	TRUE	Turn	has value	Turn	FALSE	Turn	Not Active	Turn	positive and negative velocity	Turn	Previous Component = Forward	Turn	The robot turns when it hits an obstacle
Both wheels spin forward	FALSE	Forward	has value	Forward	FALSE	Forward	Not Active	Forward	Positive velocity	Forward	Previous Component = Turn, Reverse, or Charging	Forward	The bump detection
Both wheels spin backward	FALSE	Reverse	has value	Reverse	TRUE	Reverse	Not Active	Reverse	Negative velocity	Reverse	Previous Component = Forward	Reverse	The robot moves backward when there is a cliff

Figure 4: The Drive Roomba core behavior decomposed into a CAT.

information, and provide tools to troubleshoot negative robot behavior.

3.2.1 Handling Levels of Abstraction

As robotic systems grow in complexity, CATs can establish different levels of abstraction, highlighting different levels of detail. The more CATs built the more details available about the relationships between the hardware and software of the system. The ability to abstract at different levels simplifies the raw information that must be represented at any one level, and improves the users ability to use the system reliably.

Consider that more detail is needed about a core behavior. The abstraction can be decomposed into lower level robot behaviors using the steps outlined in Section 3.2. The first step is to select the core behavior to be further decomposed. Next, for that behavior, a set of new core behaviors which are at a lower level of abstraction are selected. Finally, using the steps in Section 3.2 a lower level CAT is built.

Using the Roomba example, we select Drive and Clean as behaviors which are composed of more detailed core behaviors. First, we decompose Clean into these new core behaviors: Spot, Max, and General. These key behaviors are types of Vacuuming protocol for the Roomba. The output of these behaviors is to have the vacuum motor spin and depending on different incoming commands, and the

Roomba Advanced Behavior CAT													
	Battery Level		Warning Light		IR Sensor		Pose Estimate		Incoming Command		Change in State		Notes
	Condition	Component	Condition	Component	Condition	Component	Condition	Component	Condition	Component	Condition	Component	
Map	has value	SLAM	Not Active	SLAM	Available	SLAM	Available	SLAM	message in = use SLAM	SLAM	Previous Component = Charging	SLAM	Building a map happens as Drive and Clean happen

Figure 5: To update a CAT either modify an existing CAT or build a new CAT which can sit on-top of or in parallel with core behavior already established. This CAT is for the new SLAM Roomba feature and captures the necessary sensor data and has a map as the resulting output.

amount of dirt detected, different core behaviors will be expressed. This lower level CAT can be seen in Figure 3.

Next, consider Drive as a core behavior. This can be broken down into three new core behaviors: Turn, Forward, and Reverse. The output of these new behaviors are different combinations of wheels turning and the inputs are based on different sensor values. For example, to Turn the bump detection sensor must be true, and the cliff sensor must be false in addition to having battery and a velocity value, as soon as the condition of bump detection is false the system can transition back to the forward component. The same reasoning can be worked through for each row of Figure 4

In the life cycle of a system, it is possible to have many updates to the hardware and software components. It is important to keep the CAT up to date for it to remain useful to future users,. Likewise, CATs can be built to inherit properties of lower level CATs as more complex behaviors are added.

For example, in newer iterations of the iRobot Roomba, Simultaneous Localization and Mapping (SLAM) occurs on board the robot. To add this feature to the CAT, we make a Roomba Advanced Behavior CAT which has SLAM as a core behavior and map as an output. The necessary inputs are battery level, warning light, IR sensor, pose estimate, and incoming command. This CAT can be seen in Figure 5. While SLAM is a well understood capability, this CAT is able to abstract away code and specific hardware details, and capture the necessary components to communicate to the user how certain outputs are achieved. The CAT in Figure 5 was built separate of the core behavior CAT, however, in this case, the added complex behavior does not inherit properties from the core behavior CAT and could have been a new row in the original CAT, Figure 2. It is possible that the IR sensor is also used in one of the original core behaviors. To add a new sensor to the Roomba CAT, Figure 2, append a new column to the table for the new sensor, and update the entries along each row for core behavior that use the new information.

3.3 Rooting out Unreliability

Failures are going to occur in the lifetime of an autonomous system. The key is being able to identify the failure points, and capture if the issue is human error, environmentally caused, or hardware/software related.

First, in cases where a failure has occurred, identify which core behavior was violated. Once a core behavior has been isolated, consider if it is clear that a condition was broken at the highest level CAT. If a condition at this level is the culprit, then resolve the condition for that component and move on. If the behavior was violated but a condition was not broken, consult lower level CATs to determine which lower level behavior condition was violated. This process can be done iteratively until the broken condition is discovered.

Figure 6: An AscTec Pelican Quadrotor

Number of Launch Files	147
Lines of Code	37,036
Hardware Pieces	> 20

Table 1: Parts making up the AscTec Pelican Quadrotor

For example, if while the Roomba is in the Drive core behavior the robot stops and the battery is below the threshold constraint then the robot should transition to the Charging behavior. This is a state transition whose trigger is represented in the highest level CAT and should be resolved at that level. However, if the robot stops and the warning light is active then lower level CATs need to be consulted, examples of lower level CATs are in Figure 3 and Figure 4

It is possible for the CAT to be incorrectly constructed because this is a human generated tool. As errors are found and more knowledge is gained about the system, the CAT can be easily fixed for future by updating the entries.

4 Experiment with Quadrotor

We identified core behavior, established baseline experiments, built CATs, and isolated failure cases for the Ascending Technologies Inc. Pelican Quadrotor (AscTec Pelican Quadrotor), Figure 6. New users inherited this robotic system with an existing software infrastructure, and it was necessary in a short amount of time to have the robot operational.

4.1 Identifying Core Behaviors and Baseline Tests

To identify the core behaviors of the AscTec Pelican quadrotor and the accompanying inherited software, we went through the provided information outlined in Table 1. This information was gathered through the inspection of: hardware manuals, software documentation, and physical attributes. In Table 1, the number of launch files corresponds to all files with a .launch file type within the repository for the history of the robot. This repository has had many managers and has been in operation for over 10

years, collecting significant code rot over time. Likewise, the total number of lines of code for the system, shown in Table 1, consists of both on board robot firmware and ground station code written in: C++, embedded C, and python. Note that not all of this code is used to control the robot, and upon first inspection it was unclear how the software connected to the hardware to allow the robot to fly. Finally, in Table 1, the robot hardware consists of 4 motors, 4 Electronic Speed Controllers (ESCs), 4 propellers, 1 low level autopilot, 1 high level autopilot, sensors on both autopilots, connections between autopilots, connection from autopilot to a high level on board computer, WiFi dongle, and the frame of the robot. The hardware components were determined with physical inspection, and is a rough estimate of components where some parts like sensors on the autopilot and the frame are grouped together, and things like motors and ESCs are listed out. For a more detailed estimate, hardware manuals needed to be consulted. Note that counting out all of the sensors on the autopilot and the individual frame components would only increase overall number of robot components, increasing the overall complexity of the system. Table 1 highlights that a robot with relatively simple capabilities is comprised of many raw components. However, the basic intuition for what the robot should be able to do is: Takeoff, Fly, and Land. With further inspection, and some trial and error experimentation (with high damage cost to the robot) we realized that Idle and Emergency were also necessary core behaviors.

The establishment of core behaviors allowed for the development of a set of experiments to establish baseline performance of the robot. We used two experiments: a "hover" experiment and a "square" experiment. In the hover experiment, the robot took off, hovered at 1 m for 5 seconds, and then hovered at 2 m for 5 seconds and landed. In the square experiment, the robot flew in a 1.5m square at a fixed altitude. Both test cases contributed to testing the core behavior Fly along with transitions between different core behaviors like Takeoff to Fly to Land.

4.2 Building a Capability Analysis Table

Following the steps in Section 3.2, we built a CAT for the AscTec Pelican Quadrotor's identified core behaviors:

1. Identify core behaviors: Takeoff, Land, Fly, Idle, Emergency. Develop a set of test cases to evaluate core behaviors: hover experiment and square experiment.
2. Identify available inputs (sensors, communication, or inputs) to the system: Pose, battery, flying, arm motors, velocity vector.
3. Identify expected outputs or actions of the system: Velocity commands in positive or negative Z, Velocity commands, no input, robot on, arm motors, error.
4. Example of entering behaviors across the row: Takeoff you have outputs of arm motors and positive altitude, which requires input from pose, batter, arm motors, and velocity vectors. In each Condition, Component pair, the constraint on the input is listed in the Condition column and the core behavior is listed in the Component column. Flying is left blank because this input is not necessary for this behavior. The remaining CAT entries can be seen in Figure 7.
5. Complete the Change in Component column with the transitions between different components.
6. Finally add notes in the Notes column to describe the component and any conditions on that row.
7. To break the CAT in Figure 7 into sub components, consider the Fly behavior. We will repeat the procedure outlined in Section 3.2 to construct a more detailed description of the Fly behavior, this new CAT is in Figure 8.

AscTec Pelican CAT															
	Pose (fcu/odom)		**Battery**		**Flying**		**Arm Motors**		**Velocity Vector ****		**Incoming Command**		**Change in State**		**Notes**
	Condition	Component	Condition	Component	Condition	Component	Condition	Component	Condition	Component	Condition	Component	Condition	Component	
Arm Motors Command / **Positive Altitude Command**	Pose Available	Takeoff (/takeoff)	>= 10 V	Takeoff (/takeoff)			Yes	Takeoff (/takeoff)	Command Matches 6DOF	Takeoff (/takeoff)	Command=Takeoff	Takeoff (/takeoff)	Previous component = Idle	Takeoff (/takeoff)	Takeoff when battery >=10 and motors armed
Velocity Vector **	Pose Available	Fly (/cmd_vel)	>= 10 V	Fly (/cmd_vel)	Yes	Fly (/cmd_vel)			Command Matches 6DOF	Fly (/cmd_vel)	Command = Fly	Fly (/cmd_vel)	Previous component = Takeoff (/takeoff)	Fly (/cmd_vel)	Fly when command received and battery >=10
Negative Altitude Command	Altitude Available	Land (/land)	< 10 V	Land (/land)	Yes	Land (/land)					Command = Land	Land (/land)	Previous component = Any	Land (/land)	When Battery <= 10
	Altitude Available	Land (/land)		Land (/land)	Yes	Land (/land)					Command = Land	Land (/land)	Previous component = Fly (/cmd_vel)	Land (/land)	Only Land when Flying and commanded to do so
No Input / **Robot On**	Pose Available	Idle	>= 10 V	Idle	No	Idle	No	Idle	No	Idle	Command = None	Idle	Previous component = Land (/land)	Idle	When no commands have come in yet or all commands have executed through land
Error	Pose Not Available	Emergency									Command = Any	Emergency	Previous component = Any	Emergency	If no pose data
					No	Emergency					Command = Any	Emergency	Previous component = Fly, Land	Emergency	If not flying is false
							No	Emergency			Command = Any	Emergency	Previous component = Takeoff, Fly, Land	Emergency	If the motors are not armed
									Command Invalid	Emergency	Command = Any	Emergency	Previous component= Takeoff, Fly, Land	Emergency	If the incoming velocity is invalid

Figure 7: A CAT built for the AscTec Pelican Quadrotor at the core behavior level of abstraction.

Fly CAT													
	Pose (fcu/odom)		**Velocity Vector**		**Default Z height**		**Set Point Set?**		**Incoming Command**		**Change in State**		**Notes**
	Condition	Component	Condition	Component	Condition	Component	Condition	Component	Condition	Component	Condition	Component	
Positive or negative X velocity command	Altitude Available	Move in X	X has a value, Y = 0	Move in X	Z command has a stable value holding altitude	Move in X			Command = Must be holding altitude, Y is 0 and X is not equal to 0	Move in X	previous component = Fly	Move in X	Move in X direction with the specified velocity at a provided altitude
Positive or negative Y velocity command	Altitude Available	Move in Y	Y has a value, X = 0	Move in Y	Z command has a stable value holding altitude	Move in Y			Command = Must be holding altitude, X is 0 and Y is not equal to 0	Move in Y	previous component = Fly	Move in Y	Move in Y direction with a specified velocity at a provided altitude
Positive or negative Z Command (in meters)	Altitude Available	Hover	No velocity commands available	Hover	Z is set to a default value in software	Hover			Command = No velocity available	Hover	previous component = Fly	Hover	Move to default Z height until input velocity given
	Pose Available	Move in Z	Z has a value, X = Y = 0	Move in Z			Goal Value exists and is set	Move in Z	Command = Positive or negative Z	Move in Z	previous component = Fly	Move in Z	Move in Z direction with a specified velocity (do not move in x or y)
Positive or negative X, Y velocity command	Altitude Available	Move in XY	X and Y have value	Move in XY	Z command has a stable value holding altitude	Move in XY			Command = Must be holding altitude, X,Y command not equal to 0	Move in XY	previous component = Fly	Move in XY	Move in XY at a specified velocity with a provided altitude
Positive or negative X, Y, Z velocity command	Pose Available	Move in XYZ	X, Y, and Z have value	Move in XYZ					Command = Positive or negative X, Y, Z	Move in XYZ	previous component = Fly	Move in XYZ	Move in XYZ at a specified velocity

Figure 8: A CAT for the Fly core behavior. This CAT breaks down flying into different core behavior which provide details on different combinations of movement the robot can execute.

Number of Core Behaviors	11
Number of Outputs	12
Number of Inputs	13
Number of (Condition, Component) pairs	70

Table 2: Details on CAT components

Table 2 outlines the number of elements that make up the CATs for the AscTec Pelican Quadrotor. The two CATs built are a concise representation of all of the raw information in Table 1. The CAT reduced a large amount of raw data into a clear format representing the core behavior of the robot to allow for test and evaluation of key properties. Now that the CATs exist, they can be used as additional documentation for the robot's core behavior for future users to work with the robots reliably.

4.3 Evaluation

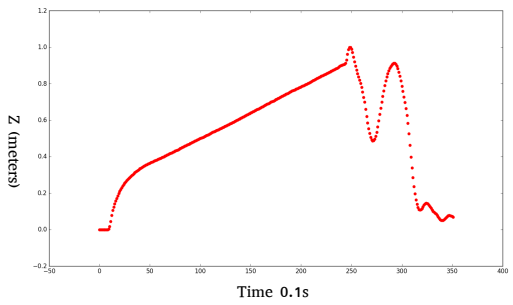
The proposed benchmark experiments in Section 4.1 were run to see that the robot was performing core behavior as expected, Figure 9. These tests where important to run to establish that the robot behaved reliably, especially, after a long period between experiments. Figure 9(b) shows the completed trajectory of a successful hover experiment, Figure 9(c) shows the completed 3D trajectory of the square experiment and Figure 9(d) shows the 2D projection of the 3D trajectory completed in the square experiment. These tests ensure that UAV the core behaviors are reliable before adding more complex algorithms or putting the robot in an environment with greater uncertainty.

4.4 Repair and Redesign

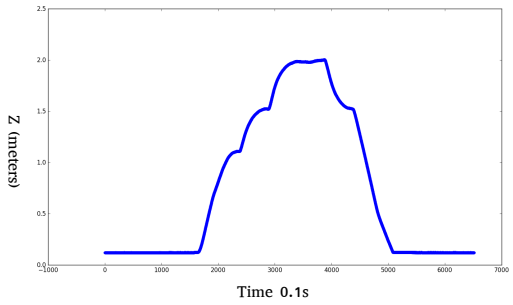
When first using the AscTec Pelican quadrotor several problems were encountered: hardware, software, and human input error. For example, after takeoff, the robot would occasionally drift toward the ceiling. Figure 9(a) shows an example of this happening where the robot drifted toward the ceiling until the safety pilot took over at ≈ 23.0 s.

The steps in Section 3.3 where followed to identify the core behavior that was failing: in this case it was the Fly behavior. The CAT in Figure 7 was inspected and after ensuring that all conditions for the FLY behavior were met, we went to the CAT in Figure 8 which breaks down the Fly core behavior. Within this CAT we identified Hover as the core behavior that was failing. It was determined that if no velocity commands were provided, and a default altitude was not set the robot would drift toward the ceiling. Thus a condition for Hover was broken, resulting in the unexpected behavior. This problem was resolved, and a successful Hover experiment is shown in Figure 9(b).

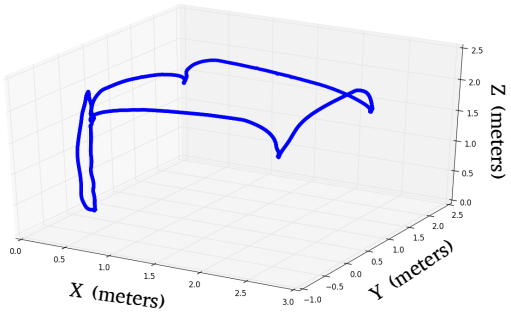
It is possible for the CAT to be incorrectly built, especially if the person is a new user and building the CAT while learning the system. The CAT provides a framework to systematically isolate the users intuition as it relates to what is known about the hardware and software. If the CAT is incorrect, it is possible to fix the table by further investigating the raw information which makes up the robot. Over time and with increased understanding of the system, the CAT will become more complete, descriptive, and provide meaningful information to future users.



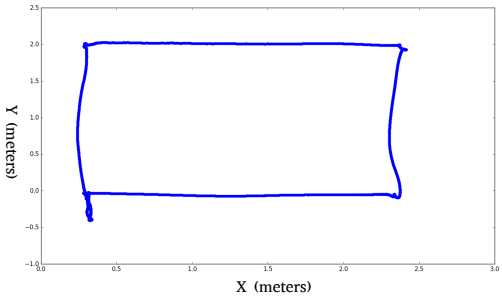
(a) Hover Experiment Failure



(b) Hover Experiment Success



(c) Square Experiment 3D view



(d) Square Experiment 2D view

Figure 9: Figure 9(a) shows the z height in meters of an unsuccessful hover experiment where the robot took off and drifted toward the ceiling. Figure 9(b) shows the resulting z height in meters for a successful hover experiment. Figure 9(c) shows the 3D path of the robot through space during a square experiment. Figure 9(d) is the 2D projection of the robots path through space.

5 Discussion

Building a CAT is more easily done by a person that has intimate knowledge of how a robot works because there is less need to consult documentation or other sources of information about the system. However, non-expert robot users can build a CAT by using basic intuition and inspecting the raw information about the system. For example, the Roomba core behavior CAT, Figure 2, is able to capture the core behavior of the system with limited technical details. The only information available was an online specification [17].

CATs have the power to condense large amounts of raw information and grow flexibly with additional capabilities. The complexity of CATs will vary system to system, but the abstraction allows CATs to describe the robotic system at the necessary level of detail for the documentation. Developing CATs may take time, but investing in the documentation of hardware and software relationships can ease burdens on future users. Likewise, building a CAT provides a method which can translate intuition into: core behaviors, a set of benchmark tests, and a way to eliminate unreliable behavior.

An added advantage of CATs is that they can also help provide clear explanations for errors which may occur in the system. New users face errors that experienced users may not encounter due to years of developing intuition for how robotic systems work. A CAT can reduce the barrier to entry to allow new users to trouble shoot problems before hitting a wall or needing more expert assistance.

Finally, the identification of core behaviors at the correct level of abstraction allow for benchmark performance evaluations to be established. Having such performance measures when changing to a more complex environment or adding new capabilities will provide insight into what the robot should do, and if the robot is physically capable of doing the task in the new environment. For example, consider a Huskey robot with a 7DOF robot arm attached. Both the Husky ground robot and the 7DOF robot arm will have individual CATs which capture the core behavior of the separate capabilities. A new table is necessary to now capture that the robot can both move and grasp simultaneously, expanding what the set of core behaviors of the robot.

6 Conclusion and Future Work

To establish reliable robot behavior we used Capability Analysis Tables to connect the robot's core behaviors with the necessary hardware and software inputs and outputs of the system. Unlike existing methods, CATs connect raw information about the robotic system's core behaviors ensuring: more robust evaluation of future algorithms, an established set of requirements, and a way to root out unreliable behavior. We use CATs to address the need for user friendly documentation to capture the ever growing complexity of autonomous systems. In providing users with a more concise representation of the autonomous system, more reliable behavior can be established.

Future work will consider how a core set of behaviors might span across many vehicle types. For example, do all UAVs share a set of core of behaviors? Is a potential set: Takeoff, Fly, Land, Emergency, and Idle? It is our hypothesis that this generalization can be made for different robot vehicle types, and that the combination of hybrid vehicle types will require additional CATs to illuminate the new behavior achieved. The specific hardware and software details for each robot would differ from platform to platform, but establishing a set of core robot behaviors could allow for a ubiquitous performance evaluation for robots in a certain category.

In addition, it is a direction of future work to make CATs machine readable and perform automatic updates to the table, along with consistency checking. The issue remains of addressing keeping CATs

human readable, and not imposing syntax that turns the tool into something like DSL or SysML which require advanced knowledge of the specification methods to use. There is active work considering how to map CATs to existing formal methods tools which will help ensure the validity of the current table structure. We are also working on how a CAT could be coupled with robot logs to do run time monitoring of the system. These directions will continue to improve the CAT methodology and evolve the tool past a documentation aid for ensuring reliable robot behavior.

References

[1] (2020): iRobot Corporation, *Roomba*. Available at `https://store.irobot.com/default/parts-and-accessories/roomba-accessories/`.

[2] R. G. Bennetts (1975): *On the Analysis of Fault Trees*. *IEEE Transactions on Reliability* R-24(3), pp. 175–185, doi:10.1109/TR.1975.5215143.

[3] A. W. Biermann & J. A. Feldman (1972): *On the Synthesis of Finite-State Machines from Samples of Their Behavior*. *IEEE Transactions on Computers* C-21(6), pp. 592–597, doi:10.1109/TC.1972.5009015.

[4] B. Bohrer, Y. K. Tan, S. Mitsch, A. Sogokon & A. Platzer (2019): *A Formal Safety Net for Waypoint-Following in Ground Robots*. *IEEE Robotics and Automation Letters* 4(3), pp. 2910–2917, doi:10.1109/LRA.2019.2923099.

[5] P. Castagliola & K. Vannman (2007): *The efficiency of the EWMA capability chart*. In: *2007 IEEE International Conference on Industrial Engineering and Engineering Management*, pp. 1389–1393, doi:10.1109/IEEM.2007.4419420.

[6] T. S. Chow (1978): *Testing Software Design Modeled by Finite-State Machines*. *IEEE Transactions on Software Engineering* SE-4(3), pp. 178–187, doi:10.1109/TSE.1978.231496.

[7] L. Delligatti (2013): *SysML Distilled: A Brief Guide to the Systems Modeling Language*, 1st edition. Addison-Wesley Professional.

[8] S. Devadas, Hi-Keung Ma, A. R. Newton & A. Sangiovanni-Vincentelli (1988): *MUSTANG: state assignment of finite state machines targeting multilevel logic implementations*. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 7(12), pp. 1290–1300, doi:10.1109/43.16807.

[9] S. Dhouib, S. Kchir, S. Stinckwich, T. Ziadi & M. Ziane (2012): *RobotML, a Domain-Specific Language to Design Simulate and Deploy Robotic Applications*. In: *Third international conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR12)*, pp. 149–160, doi:10.1007/978-3-642-34327-8_16.

[10] A. C. Dias Neto, R. Subramanyan, M. Vieira & G. H. Travassos (2007): *A Survey on Model-based Testing Approaches: A Systematic Review*. In: *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22Nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, WEASELTech '07, ACM, New York, NY, USA, pp. 31–36, doi:10.1145/1353673.1353681.

[11] A. Mark Doggett (2005): *Root Cause Analysis: A Framework for Tool Selection*. *Quality Management Journal* 12(4), pp. 34–45, doi:10.1080/10686967.2005.11919269.

[12] J. L. Fernandez, R. Sanz, E. Paz & C. Alonso (2008): *Using hierarchical binary Petri nets to build robust mobile robot applications: RoboGraph*. In: *2008 IEEE International Conference on Robotics and Automation*, pp. 1372–1377, doi:10.1109/ROBOT.2008.4543394.

[13] P. Freedman (1991): *Time, Petri nets, and robotics*. *IEEE Transactions on Robotics and Automation* 7(4), pp. 417–433, doi:10.1109/70.86074.

[14] N. Gobillot, C. Lesire & D. Doose (2014): *A Modeling Framework for Software Architecture Specification and Validation*. pp. 303–314, doi:10.1007/978-3-319-11900-7_26.

[15] M. Hause (2006): *The SysML modelling language*. In: *Fifteenth European Systems Engineering Conference*, 9, pp. 1–12.

[16] B. Hayes & J. A. Shah (2017): *Improving Robot Controller Transparency Through Autonomous Policy Explanation*. In: *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, pp. 303–312, doi:10.1145/2909824.3020233.

[17] iRobot (2005): *iRobot Roomba Serial Command Interface (SCI) Specification*. iRobot Corporation. Available at `https://web.archive.org/web/20131202230831/http://www.irobot.com/images/consumer/hacker/Roomba_SCI_Spec_Manual.pdf`.

[18] Y. Jarraya, A. Soeanu, M. Debbabi & F. Hassaine (2007): *Automatic Verification and Performance Analysis of Time-Constrained SysML Activity Diagrams*. In: *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, pp. 515–522, doi:10.1109/ECBS.2007.22.

[19] S. Kotz & N. L. Johnson (2002): *Process Capability IndicesA Review, 19922000*. *Journal of Quality Technology* 34(1), pp. 2–19, doi:10.1080/00224065.2002.11980119.

[20] H. Kress-Gazit, M. Lahijanian & V. Raman (2018): *Synthesis for Robots: Guarantees and Feedback for Robot Behavior*. *Annual Review of Control, Robotics, and Autonomous Systems* 1(1), pp. 211–236, doi:10.1146/annurev-control-060117-104838.

[21] H. Kress-Gazit & H. Torfah (2018): *The Challenges in Specifying and Explaining Synthesized Implementations of Reactive Systems*. In: *Proceedings 3rd Workshop on formal reasoning about Causation, Responsibility, and Explanations in Science and Technology, CREST@ETAPS 2018, Thessaloniki, Greece, 21st April 2018.*, pp. 50–64, doi:10.4204/EPTCS.286.5.

[22] R. Laleau, F. Semmak, A. Matoussi, D. Petit, H. Ahmed & B. Tatibout (2010): *A first attempt to combine SysML requirements diagrams and B*. *ISSE* 6, pp. 47–54, doi:10.1007/s11334-009-0119-y.

[23] G. L. Landgren & S. W. Anderson (1973): *Simultaneous Power Interchange Capability Analysis*. *IEEE Transactions on Power Apparatus and Systems* PAS-92(6), pp. 1973–1986, doi:10.1109/TPAS.1973.293577.

[24] G. L. Landgren, H. L. Terhune & R. K. Angel (1972): *Transmission Interchange Capability - Analysis by Computer*. *IEEE Transactions on Power Apparatus and Systems* PAS-91(6), pp. 2405–2414, doi:10.1109/TPAS.1972.293398.

[25] D. Lee & M. Yannakakis (1996): *Principles and methods of testing finite state machines-a survey*. *Proceedings of the IEEE* 84(8), pp. 1090–1123, doi:10.1109/5.533956.

[26] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon & M. Fisher (2019): *Formal Specification and Verification of Autonomous Robotic Systems: A Survey*. *ACM Comput. Surv.* 52(5), doi:10.1145/3342355.

[27] D. M. Lyons & M. A. Arbib (1989): *A formal model of computation for sensory-based robotics*. *IEEE Transactions on Robotics and Automation* 5(3), pp. 280–293, doi:10.1109/70.34764.

[28] A. H. Miyazawa, P. F. De Oliveira Salazar Ribeiro, W. Li, A. L. C. Cavalcanti, J. I. Timmis & J. C. P. Woodcock (2019): *RoboChart: modelling and verification of the functional behaviour of robotic applications*. *Software and Systems Modeling*, doi:10.1007/s10270-018-00710-z.

[29] T. Murata (1989): *Petri nets: Properties, analysis and applications*. *Proceedings of the IEEE* 77(4), pp. 541–580, doi:10.1109/5.24143.

[30] A. Nordmann, N. Hochgeschwender & S. Wrede (2014): *A Survey on Domain-Specific Languages in Robotics*. 8810, doi:10.1007/978-3-319-11900-7_17.

[31] S. Ouchani, O. At Mohamed & M. Debbabi (2014): *A formal verification framework for SysML activity diagrams*. *Expert Systems with Applications* 41(6), pp. 2713 – 2728, doi:10.1016/j.eswa.2013.10.064. Available at `http://www.sciencedirect.com/science/article/pii/S0957417413008968`.

[32] J. L. Peterson (1977): *Petri Nets*. *ACM Comput. Surv.* 9(3), pp. 223–252, doi:10.1145/356698.356702.

[33] A. Petrenko (2001): *Fault Model-Driven Test Derivation from Finite State Models: Annotated Bibliography*, pp. 196–205. Springer Berlin Heidelberg, Berlin, Heidelberg.

[34] V. Raman & H. Kress-Gazit (2013): *Explaining Impossible High-Level Robot Behaviors*. IEEE Transactions on Robotics 29(1), pp. 94–104, doi:10.1109/TRO.2012.2214558.

[35] A. Rauzy (1993): *New algorithms for fault trees analysis*. Reliability Engineering and System Safety 40(3), pp. 203 – 211, doi:10.1016/0951-8320(93)90060-C.

[36] S. A. Redfield, E. I. Leonard & J. Lennon (2019): *Task Specification and Behavior verification for UUV Behavior Design*. Autonomous Underwater Vehicles Design and practice.

[37] J. Rooney & L.N. Hauvel (2004): *Root Cause Analysis For Beginners*. Quality Progress 37, pp. 45–53.

[38] R. Steiner S. Friedenthal, A. Moore (2006): *OMG Systems Modeling Language (OMG SysML) Tutorial*. INCOSE, Object Management Group. Available at `https://utilities.omg.org/news/meetings/workshops/SBC_2007_Presentations/00-T1_Friedenthal.pdf`.

[39] C. Schlegel, A. Steck, D. Brugali & A. Knoll (2010): *Design Abstraction and Processes in Robotics: From Code-Driven to Model-Driven Engineering*. pp. 324–335, doi:10.1007/978-3-642-17319-6_31.

[40] D. Wang, H. Ting, C. Chao & T. Koo (2013): *Process capability analysis on autoregressive process*. In: *2013 10th International Conference on Service Systems and Service Management*, pp. 78–80, doi:10.1109/ICSSSM.2013.6602525.