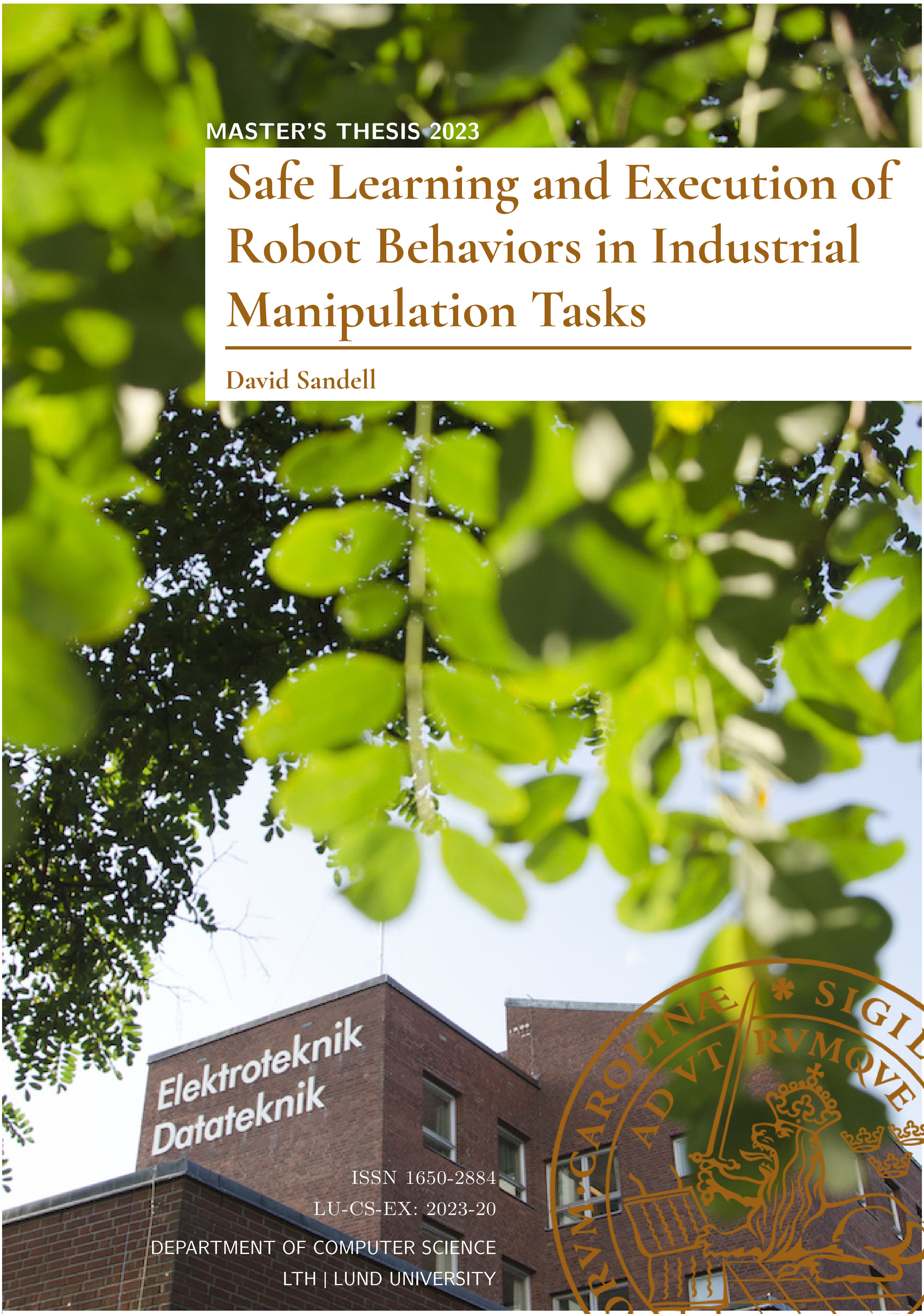


MASTER'S THESIS 2023

Safe Learning and Execution of Robot Behaviors in Industrial Manipulation Tasks

David Sandell

ISSN 1650-2884
LU-CS-EX: 2023-20

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-20

Safe Learning and Execution of Robot Behaviors in Industrial Manipulation Tasks

Säker Inlärning och Utförande av Robotbeteneden i Industriela Manipulationsuppgifter

David Sandell

Safe Learning and Execution of Robot Behaviors in Industrial Manipulation Tasks

20th June 2023

Master's thesis work carried out at
the Department of Computer Science Faculty of Engineering
LTH
Lund University
Box 118
SE-221 00 Lund
Sweden

Subject: Degree Project in Computer Sciences for Engineers (EDAM05)
Examiner: Christoph Reichenbach, christoph.reichenbach@cs.lth.se
Supervisor: Matthias Mayr, matthias.mayr@cs.lth.se
Assistant supervisor: Momina Rizwan, momina.rizwan@cs.lth.se

Abstract

When a robotic arm is performing a task, there is a risk of undesired and possibly dangerous behaviour. Two such dangers can be physical injuries or the destruction of equipment. In most applications today a full safety stop with guide rails is implemented.

A deep dive into robot safety in the form of a literature study was carried out. This newly acquired knowledge was then used to perform various safety rule studies to define some more straightforward classes of situations that could be dangerous and what response is appropriate for each of them.

To reduce these risks, a safety program was created, whose aim is to ensure that when the robot arm violates safety, the dangerous behaviour is stopped, and if the situation allows, the robot arm is returned to an earlier state that is safe. This resulted in both an interface between the robot arm's controller and the safety program, but also a safety program that was able to perform multiple reactions. One of them is the ability to return to the start position. The program is able to follow the rules for speed, force, and position, where the rule for position is defined in terms of spatial volumes. A template for adapting and creating rules and reactions to broken rules was also created. A series of successful experiments of keeping rules were carried out to show the safety program's validity and functionality. These were carried out on two different reinforcement learning scenarios. The novelty of the solution in this thesis was to design and evaluate a system that steers reinforcement learning towards a safe behaviour by integrating rule-based safety monitoring into the training process. The other uncommon behaviour for safety applications, present in this solution, is to have a way to recover to a safe state automatically when rule breaches occur.

Keywords: robot arm, safety, human-robot interaction, reaction, safety rules, rules

Sammanfattning

När en robotarm utför en uppgift finns det risk för oönskat beteende som kan vara farligt. Två sådana faror kan vara fysiska skador eller förstörelse av utrustning. De flesta tillämpningar på säkerhet idag består av säkerhetsstop och säkerhets regler.

En djupdykning inom robotsäkerhet i form av en litteraturstudie utfördes. Denna nyförvärvade kunskap användes sedan för att utföra olika säkerhetsregelstudier för att definiera några fler enkla klasser av situationer som kan vara farliga och vilken respons som är lämplig för var och en av dem.

För att minska dessa risker, har ett säkerhetsprogramet skapats, vars syfte är att säkerställa att när robotarmen bryter mot säkerheten, så stoppas det farliga beteendet, och om situationen tillåter, återgår robotarmen till en tidigare position som är säker. Detta resulterade i både ett gränssnitt mellan robotarmens styrenhet och säkerhetsprogramet, men också ett säkerhetsprogram som kunde utföra flera olika reaktioner, en av dem är en reaktion som återställer positionen för robotens sluteffektor till startpositionen. Programmet kan följa reglerna för hastighet, kraft och position, där regeln för position definieras i termer av volymer. En mall för anpassning och skapande av egna regler och reaktioner för brutna regler skapades också. En serie av experiment utfördes för att visa programmets giltighet och dess funktionalitet detta gave ett positivt resutlat. Det nya med denna lösning är att både ha ett system som kan användas på Förstärkningsinlärning för att säkerställa dess säkera beteende, men också att ha ett sätt att återhämta sig till ett säkert tillstånd automatiskt.

Nyckelord: robotarm, säkerhet, människa-robot interaktion, reaktion, säkerhetsregler, regler

Acknowledgements

This Degree Project in Computer Sciences for Engineers is the master's thesis that concludes my degree in Master of Science in Electrical Engineering with the specialization in Control and Automation. I am grateful for the help I received from different people during this master's thesis.

I would like to thank my supervisor Matthias Mayr for this opportunity to do this master thesis and for his support during the hard work both with ideas, programming errors as well as helping me forward when I got stuck, but also helped guide me through the entire project.

I would also like to thank my other supervisor Momina Rizwan for helping me with the safety studies, helping to guide me with search terms and giving me material to study.

I would like to thank Faseeh Ahmad who, even though he was not my supervisor, also helped me during the programming phase.

A big thanks to my parents as well who have been supporting me through this project.

Terminology and definitions

Action vector
The action vector is a vector that describes an action taken by the robot arm in this case. An action can be, for example, move to this new position.

Clamping in robot structure
This scenario means that a human has gotten a limb stuck in the robot arm structure whilst the robot arm is applying a force on the stuck limb.

Compliance mode
This is a mode where the robot's motor stiffens goes to 0, making it comply with externally applied forces put on it hence why its called compliance mode.

Constrained impact
Constrained impact is an impact scenario where a human gets stuck between an object and the robot arm.

Contact scenarios
Contact scenarios are a descriptive term describing a situation where the robot arm has come in contact with something.

Electric tools
Electric tools can be anything from plasma cutters to heat-guns to tools based on electric motors, or any other tool which gets its energy from an electric source.

End-effector
An end-effector is a device placed at the end of a robotic arm with the purpose of adding some functionality to interact with the environment.

Force vector
This is a vector describing a force.

Hazard of the object
How dangerous an object is to a certain situation.

Joint
Joint in this context refers to, for example, an elbow joint in the robot arm. The robot arm contains many different joints giving it mobility.

Memory mode
Memory mode is a mode where the safety program remembers previous states which can then be used to perform different reactions.

Mode
A mode describes a setting for how to behave when certain criteria are met.

No human present
No person is detected near the robot.

Normal operations zones
Areas where no extra limits are put on the robot.

Object characteristics
Object characteristics is a descriptive character of a certain object, these can be for example anything describing how soft or hard an object is to how sharp or dull it is.

Object position
Position of an object.

Object safety
Object safety means ensuring safety around an object.

Operator present
An operator for the robot is detected near the robot.

Partially constrained impact
Partially constrained impact is an impact scenario where the human becomes constrained for a short time but is able to move one of the objects away.

Power tools
Power tools are tools usually based on either compressed air, electric motors or some form of combustion engine.

Quasi-static contact
Quasi-static contact is a contact scenario where a human is not able to recoil way form the contact scenario.

Reaction vector
The reaction vector is the same as the action vector. The difference is that the values in it have been changed by the safety program as a reaction to some breach of the safety limits.

Reduced operations
A decrease in allowed movements such as speed or forces.

Return to origin
Return to origin is one of the reaction behaviors for a broken rule created in this thesis. The reaction causes the robot arm to return to the starting position.

Reward system
A reward system refers to a system manly used in artificial intelligence (AI) training applications. The reward is a signal to the AI that the behaviors leading up to the reward were a preferred behavior increasing the likely hood of that behavior happening again in the future.

RL reinforcement learning
RL is a form of neural network training method based on simulated reward systems.

Robot Operation Modes
A mode describes a setting for how to behave when certain criterion are met. Robot operation modes in this sense means modes in which the robot can be operating.

RosMari
RosMari is a programming language with the focus on safety and was one of the inspirations for this thesis.

Safety interface
The safety interface is the program created in this thesis that manages communication between the controller and the safety program.

Safety limits
A descriptive term is the safety limit, describing as it says a limit that ensures safety.

Safety Studies
Safety studies are a set of idea brainstorming about robot safety that are carried out in this thesis.

Safety Study
Safety study is one single brainstorming about robot safety.

Safety system
Is a descriptive term meaning a program or a set of programs that ensures safety.

Singularity scenarios
Is when the robot's number of degrees of freedom is reduced in some amount by its configuration in space.

State vector
A vector describing the state the robot is in. The state in this case describes the state of a robot arm a state can be for example a measured position.

Stop operations
The robot is forced to stop.

Stop zones
Areas where the robot is forces to stop of it enters them.

Taxonomy
Taxonomy is a system of categorizing or classifying groups of objects and things in science.

Transient contact
In a Transient contact scenario the human is able to recoil away from the contact scenario.

Tool safety
Tool safety means ensuring safety around a tool.

Tools

Tools can be for example anything from a saw to a drill to a gripper or possible a welder that can either be used by the robot arm or exist in its environment.

Unconstrained impact
Is as it is called an impact scenario where the human is not constrained from moving away.

Visitor present
A visitor is detected near the robot.

Volumes
Volumes is a term used to describe a virtual created 3d-shape that in this thesis has some safety functionality attached to it.

Warning zones
Areas where extra safety limits are added to the robot if it is in them.

Contents

1 Introduction

1.1 Background and Motivation

The research question of this thesis is how to formulate and implement safety constraints on the movements and forces of a robot arm in a way that stops dangerous behaviours from occurring. In the area of robotics and industrial machines, safety is an important aspect. Many machines and robots work with great forces, high speeds and are heavy. This makes these machines dangerous to work with. For example, an operator of such a machine can accidentally give a command that can be catastrophic to equipment and be very expensive to repair. It could harm humans in the case where human-robot interaction (HRI) is possible. This also applies if the control is given to some sort of artificial intelligence (AI) system. The usual solution to this is to implement both physical and software guide rails and safety stops to make sure that if dangerous behaviour is detected, the machine shuts down. It also means that the production stops, if the machine is unattended, which often is unacceptable. To find a better solution would be of great interest to this field.

1.2 Project Aims and Main Challenges

It is assume that the robot of this thesis is a compliant robot arm which is a robot arm that has a springiness effect to an externally applied force. The robot arm is assumed to weight about 28 kg and have a length of about 1 m. It is further assumed that the work area can be populated with both humans and various objects that the robot can come in contact with. Another assumption is that the main task of the robot is to insert a Small cylindrical peg attached to the end of the robot arm in to a cylindrical hole placed in the work area. The final assumption is that the robot arm is going to try to do things that are not safe neither for any objects in the work area but also not for any humans present in the work area.

The aim of this master thesis work is to ensure safety of a robot application in a working environment. The goal is to have a safety system for both humans that can be in the work area but also objects present that can be fragile as well as the robot arm it self.

A subsequent aim is to define a format for safety that makes sure that a controller handling the arm is not able do anything that can destroy equipment or harm anyone. This is to be done by establishing rules and limits for this format, where the first step is to define what is “safe” and what is “not safe”. For example, the arm could only be allowed to move in a specified spatial volume. Similarly, a forbidden volume can be defined. Additionally, limiting forces and speeds might be constraints worth considering.

Another aim is to implement a safety system based on the format for safety with the rules and limits, as well ass adding appropriate reactions for the breach of these. A reaction For example, could be that the movement is stopped. But often a better reaction is to return to a previous step that is inside of the rule conditions. It is also relevant to be able to return to the start of the task if a failure occurred. This is because obstacles in the workspace could mean that a straight line back to the start position is not always an option.

The last aim is to be able to use the safety system in a reinforcement learning (RL) application to ensure a safe behavior while training.

This project will not, however, cover programming any artificial intelligence (AI) or machine learning (ML) algorithms. This project will not cover any other robots but the *Kuka iiwa* robot arm (although the developed solution is general enough to be compatible with other robots). It will only cover the physical safety of objects and humans but not computer security.

The main research questions of this thesis is: Is it possible to create a safety system that is capable of ensuring a safe behavior by a robot arm controlled by an artificial intelligence? Is it possible to run a safety system on a training artificial intelligence model? Is it possible to have an automatic recovery behavior for a robot arm when safety is breached?

1.3 Approach and Methodology

The thesis is divided into 4 main parts, see Figure 1.1. The first part covers the literature studies, which are performed to get a better understanding of safety in an industrial environment and in human-robot interactions. Then a summary report is created as a basis for future steps.

The second part are the safety studies which are based on the literature studies for developing a structure for defining safety in the context of the robot. The structure is then scrutinized to ensure its rigidity. When a rigid structure for safety is met the work moves on to the next step. However, If the structure is found flawed, more literature studies are to be carried out until a good level is met.

The third part is the program implementation. Which consists of creating a program which resembles the structure for safety, developed in the previous step, as closely as possible. Minor testing of the program is carried out to see that all parts work as intended and if more features are needed until a fully working version is created.

The final step is to validate the program by testing it in simulation to see if unsafe behaviors are being stopped correctly and that safe behaviors are allowed to continue. Data about the performance of the tests are then to be gathered for further analysis.

The chapters are structured as follows: Section 2 describes the methodology. Section 3 describes the results of the literature studies. Section 4 describes the safety studies. Section 5 describes the safety program and Section 6 contains the validation of the

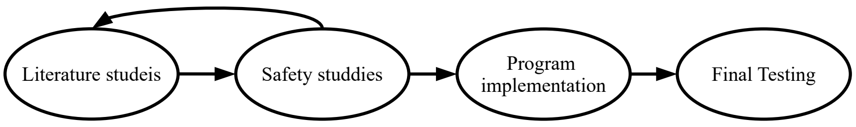


Figure 1.1: A Flowchart displaying the method of this thesis divided in to four steps

program. Section 7 contains the discussion and conclusion, as well as future work.

2 Methodology

The first part of the project was to get acquainted with the tools and software that were going to be worked with in the production of the safety system. The second step was to conduct literature studies of both robot safety in general, but also safety in robot-human interaction as well as safety when training and running RL on robot systems. This was done to have better understanding of safety in the field of robotics. After the literature studies, some safety studies in the form of thought experiments were carried out and documented, however the initial resulting safety studies were found flawed as they did not pass the scrutinization. Thereafter, new iterations of literature studies and safety studies continued until a satisfactory safety manual and document was created as well as a few descriptive charts, see section 4. This was done to have the base for the program.

In the final step, the implementation of the two programs, the safety interface and the safety system, was to be created, the safety interface was programmed using C++ and the safety system was done in Python. The programming was done in incremental steps with small goals along the way towards the main goal.

2.1 Literature Studies

It was important to acquire knowledge in the field of safety in industry and robotics in order to have the appropriate background to be able to create a capable safety system. The search started with interviewing knowledgeable people in the field at the university for guidance of were to beguine the literature studies. Here a few articles on the idea of the RosMari safety system programming language were acquired. RosMari is a programming language with the focus on safety, that one of the supervisors had created. The focus in the articles was on the idea of separating the safety system into a separate process but also some safety system examples.

One idea when doing the literature studies was to find some sort of taxonomy tree structure to base the safety studies on. This, however, proved more difficult than expected. Dialogue with the people at the university about their knowledge in the field and articles that would be of interest were continuously held. It was also decided to go and ask the university library for help, which resulted in an interesting article[1]. The results were documented and summarized in a minor report and discussed with the supervisors.

2.2 Safety Studies

The safety studies were an exploratory brainstorming activity based of the literature studies where the goal was to creating a structure or taxonomy that would single out important aspects of safety. The purpose behind the creation of a structure or taxonomy was to have a blueprint to follow when making the safety program.

Safety studies were carried out a few times after some literature studies. The first safety study was to divide according to categories found in the literature studies following a tree based structure with a root base branching out in to more detailed categories. The resulting structure was however not as well grounded in literature studies as was initially thought.

After more literature studies, a new tree structure was created, this time dividing according to *normal operations*(the robot follows the task at hand with no extra safety limits), *warning mode*(the robot has some extra limits but is still able to continue working) and *disaster mode*(the robot is stopped or shut down). After reviewing the resulting structure it was again decided to do another safety study

The final literature study was the basis for the two final safety studies, one for defining the most important features for categorizing safety and the other for the creation of the schematic of the safety program. The idea of the categorization was to go through each possible main situation for the robot setup creating a volume-based safety system. And the idea for the schematic of the program was to create a tree of data structures to visualize their respective attributes, see section 4.3.

2.3 Programming

According to the literature studies and safety studies a program was created. This was done with normal programming procedures with both bug testing and fixing. The main goal of this program was to ensure the safety that was defined in the safety studies and that the robot arm would always be stopped from acting in a dangerous way. Two programs were created, one being the safety program itself and the other being a interface program. The two programs had to communicate with each other, this meant creating a communication system. Thankfully the basis for this already existed in the ROS package. In order to test the capabilities of the programs and check their different functionalities, they were both run on the simulation. This was also done interactively for each new ability that was added until a final well-working program was acquired. The final testing that was performed was to examine if the safety program would stop the dangerous behaviors. This was done by inducing dangerous behaviors by the AI and seeing if this was appropriately stopped. It was further also checked that good behaviors were allowed to continue without intervention from the safety program.

3 Safety of Robot Systems

This part consists of the Literature studies which were carried out in two main steps. These steps were carried out to increase the knowledge in the area of safety in robotics for the purpose of having a firm base to build a safety system on. The first of the two steps was the different contact scenarios which were studied for the purpose of getting a better understanding of the physical limitations for how safe contact is defined. The second step focuses more on different taxonomies and contains the summary of the literature study.

3.1 Robot Contact Scenarios Literature Study

The article [2] was one of the inspirations for the first iteration of safety studies as it contains a form of taxonomy for different scenarios of contact and accidents. The article [2] was interesting in the sense that it contained a taxonomy which was partially dividing according to the following structure; near singularities(which is when the robot's number of degrees of freedom is reduced by its configuration in space), in singularity, as well as, dynamic contact, quasi-static contact and sharp or blunt contact. These examples were interesting due to the similar nature of the contact scenarios to the robot arm in this thesis which also can cause such contacts to occur. Another inspiration was a paper which contained a few examples of how safety limits work [3]. Here an autonomous mobile robot is designed to work on mink farms and the robot is to drive around on the farm. One problem encountered in this article [3] is making sure that the robot does not fall over, which prompted limits on the measured tilt angle as well as speed limits. Another limit was on the time spent in each farmhouse due to the exhaust from the robot which in high consentration could be dangerous for the animals at the farm. This same robot was also able to take corrective measures for each breach of a limit. This shows the multi-variability of safety and that there are many things to take into consideration to ensure safety in different applications.

When looking for human safety in human-robot interaction, it is important to follow values for safe human-robot contact. Two articles that stood out in this area was [4, 5], in both it was concluded that for a robot arm of a similar size and weight as the one used for this thesis project (a *Kuka LBR iiwa*), the safety limit for speed was to be applied at a maximum safe velocity of 0.25 m/s. However, in [5] some examples concluded a much lower speed of 0.1 and 0.2 m/s, whilst it was also argued in the same article that the speed limit of 0.25 m/s is a bit too confining. This was debated in light of the possibility of different setups and functionalities of the task at hand.

In this thesis a quick hand calculation was carried out as a simple validation for the robot arm that safety was to be applied to. Some estimations of the weight of the robot arm and of the thickness of the skin at the temple of a human head, yielded a very similar resulting velocity as a safety limit. The arm was modeled as a solid bar with equal mass density along the arm, which rotates around one of the ends, and then impact an object with an estimated skin thickness of 0.2mm, which is the maximum allowed deceleration distance, simulating the head of a human. In this case the maximum allowed force is $\mathrm{F_{Max}}$=110 N according to Table 3.1, if it is assumed that the hit is from a 1cm^2 area hitting at the temple. This 1cm^2 area is assumed reasonable for the geometry of the robot arm in this thesis.

First are the estimated physical values of the thickness of the skin of a human head l_{skin}, as well as the length of the robot arm l_{arm}, and the weight of the robot arm m_{arm}. Here F_{Max} is the maximum allowed force on the head at the impact point.

$$l_{skin} \approx 2\ mm \tag{3.1}$$

$$l_{arm} \approx 1\ m \tag{3.2}$$

$$m_{arm} \approx 28\ kg \tag{3.3}$$

$$F_{Max} \approx 110\ N \tag{3.4}$$

The max Energy allowed in a collision with the estimated breaking length of 2mm and a max force of 110 N gives in Equation 3.5

$$E_{human} = F_{Max} * l_{skin} \approx 0.22\ J \tag{3.5}$$

This is very close to the number 0.23 J presented in the article [6] for the maximum allowed energy transfers for an impact scenario to the forehead.

Equation 3.6 is the equation for the moment of inertia of a symmetric beam fastened at one end, acquired from [7].Where I is the inertia, m the mass, and l the length, of the arm.

$$I_{arm} = m_{arm} * \frac{l_{arm}^2}{3} \tag{3.6}$$

Equation 3.7 is the energy equation for a rotating symmetric beam fixed in one end point [7]. E is the rotational energy, I is the inertia, m the mass, l the length and ω is the rotational velocity.

$$E_{arm} = I_{arm}\frac{\omega^2}{2} \tag{3.7}$$

Combining Equation 3.6 with Equation 3.7 gives the following Equation 3.8

$$E_{arm} = m_{arm} * \frac{l_{arm}^2}{3} * \frac{\omega^2}{2} = \frac{m_{arm} * l_{arm}^2 * \omega^2}{6} \tag{3.8}$$

Calculating the maximum allowed rotational velocity for the robot arm combines Equation 3.8 with Equation 3.5 which gives the Equation 3.9 which in turn results in Equation 3.10

$$E_{arm} = E_{human} \Rightarrow \frac{m_{arm} * l_{arm}^2 * \omega^2}{6} = E_{human} \Rightarrow \tag{3.9}$$

Body region	Specific body area	Quasi-static contact
		Maximum permissible pressure N/cm^2
Skull and forehead	Middle of forehead	130
	Temple	110
Face	Masticatory muscle	110
Neck	Neck muscle	140
	Seventh neck muscle	210
Back and shoulder	Shoulder joint	160
	Fifth lumbar vertebra	210
Chest	Sternum	120
Abdomen	Pectoral muscle	170
	Abdominal muscle	140
Pelvis	Pelvic bone	210
Upper arms and elbow joints	Deltoid Muscle	190
	Humerus	220
Lower arms and wrist joints	Radial bone	190
	Forearm muscle	180
	arm nerve	180

Table 3.1: A list of biomechanical limits describing the max tolerable forces for different human contact scenarios, from [6]

$$\omega = \sqrt{\frac{E_{human} * 6}{m_{arm} * l_{arm}^2}} \approx 0.22\ rad/s = 12\ ^{\circ}/s \tag{3.10}$$

Translating angular velocity to linear in Equation 3.11.

$$1\ m * 0.22\ rad/s \approx 0.22\ m/s \tag{3.11}$$

Compared to the values used in prior work, 0.25 m/s is very close to the calculated value of 0.22 m/s Equation 3.11 velocity. However, even though these values are a base for making sure that the right safety limits are put in place, it is still not necessarily enough. In the article [2] the risk of simply trusting the sensor values was highlighted. A difference of as high as 100 N between the sensor values and the measured value was recorded. Instead, it was suggested to perform physical measurements of the various movements of the robot and the maximum values that appear, to provide as a risk assessment for safety.

3.2 Taxonomy Literature Study

To find any taxonomy-based standard for safety has yielded no major result. The search lasted for 3 weeks on the university's search engine LUBsearch, the university library and Google scholar. What has been found that was relevant, although not standard, are the following taxonomy figures, Figure 3.1 and Figure 3.2.

3.2.1 Different Types of Failures That Can Occur

The first taxonomy tree that was found, see Figure 3.1, was based on the different causes of breaches in safety. It did not focus on the situation of the breach of safety, but the focus was on who or what caused the breach to occur and who is responsible for the breach. This however, is not quite what this thesis is about as in this thesis the focus is on identifying the situations that have a high probability of being dangerous.

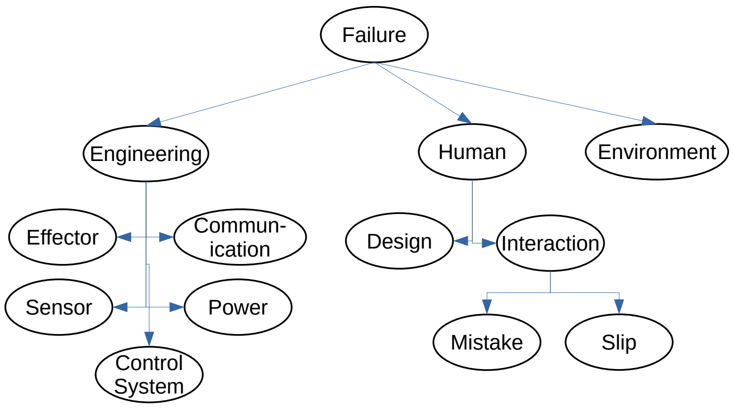


Figure 3.1: A Taxonomy describing causes to failures which can come form three main sources: Engineering failures, environmental circumstances and Human errors. According to [5].

3.2.2 Types of Human-Robot Contact Scenarios

Another taxonomy tree that was found see Figure 3.2 was one which was based on the dangerous situations that can occur. However, it does not cover situations other than where a human is involved and focuses only on the human aspect of safety. It also goes into more detail in the human safety aspect for some areas which were later seen in this thesis as unnecessary due to them being sen as equally disallowed and so they could be lumped together.

3.2.3 Different Perspectives on Safety

The following are citations from [1] containing different human-robot interactions from different sources that are summarized in [1]. The most applicable scenarios for this thesis have been hand-picked.

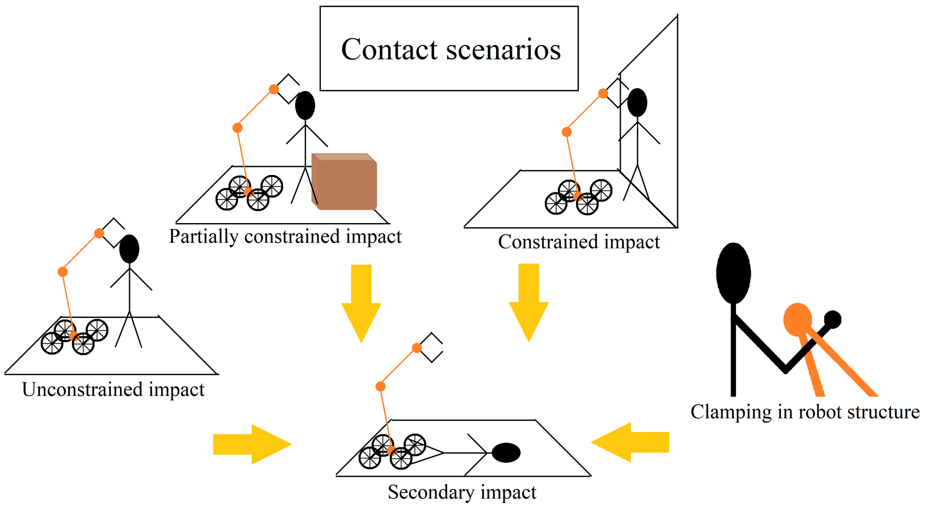


Figure 3.2: Different scenarios of contact between robots and humans classified according to [5]. The figure on the bottom left represents an unconstrained contact scenario where the person is able to recoil. The middle bottom represents a secondary possible contact scenario after the first contact. The bottom right image represents a clamping contact scenario where the person gets stuck in the robot structure. The top left image represents a contact where the person is partially constrained by objects in the environment. The top right scenario is a contact scenario where the person is constrained between a solid object and the robot. All scenarios can lead to a secondary impact scenario.[5]

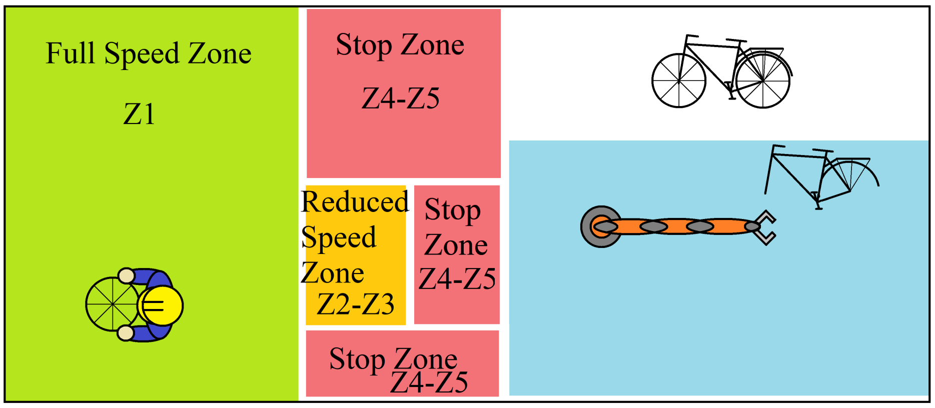


Figure 3.3: An example of a setup with different work zones. Each zone has a different mode of operation. This is described both in this figure-image but also in the Table 3.2 accompanying it. The figure is inspired by [8]

In [1] a literature review was found on the human-robot collaboration (HRC) subject. The parts most interesting for the robot arm in this thesis application were then added to a collection. These are in the text that follows in this subsection:

The first interesting example was from [1] also found in [9], was an industrial safety application where safety was divided into, "human action restriction, robot behaviour modification, injury classification, injury minimisation and collision avoidance" [1]. This gave an insight into an example of categories of hazard avoidance to keep in mind when setting up the safety rules later.

		Action strategy in HRC		
	Robot action		Human action	
	If operator enters	If visitor enters	Operator	Visitor
Warning zone (Z1)	Warning signal	Warning signal	Enter with awareness	Stay with caution
Speed reduction zone (Z2)	Speed reduction	Stop	Continue individual process	Not allowed leave safely
Speed reduction zone-the head area (Z3)	Stop	Hard stop, manual restart needed	Continue individual process	Not allowed leave safely
Collaborative zone (Z4)	Collaborative task	Hard stop, manual restart needed	Collaborative task	Not allowed leave safely
Stop zone (Z5)	Stop	Hard stop, manual restart needed	Continue individual process	Not allowed leave safely

Table 3.2: The Table describes the different modes of operation and the different apropreate actions to take in the different zones described in Figure 3.3. The table is from [8].

Another example was [10] where a gripper was used in human-robot collaboration and the conclusion was that the main danger was crushing and pinching actions from the gripper on the human [10]. This was interesting due to the fact that it gave examples of the most common dangers when dealing with grippers. However, these dangers are still applicable to the robot arm in this thesis as pinching and crushing are still very real dangers.

In [1, 11], different robot designs are discussed. Mentioned are, the compliance of the robot, how human-friendly the physical design is, for example: Rounded edges instead of sharp edges, if the system is well monitored with cameras and sensors etc., if stiffness is variable and if the real-time system is of high enough quality or fast enough in its reactions for the application [1]. Even though not many of these are applicable to the thesis, they are in any case important things to take into consideration for the current equipment to ensure that it is not stretched to its limits.

Further, reactions were discussed, Both a stop reaction and a reaction for moving away from the human as well [1, 12]. Preventative measures where the robot is trying to avoid the human, correcting the robot's path is mentioned. Also warning the human in question of possible dangers, and stopping if needed are methods mentioned[1, 13]. These reactions are quite similar to the proposed recovery behaviour in this thesis. However, the difference is that a recovery is a self-correction of an already broken rule. Here the reaction is of a more preemptive nature.

Then a solution is proposed that divides the robot workspace into different zones with different parameters for the robot arm to work under. This follows the layout of Figure 3.3 and Table 3.2. Here a quote follows form the paper [1]: *"Stop and reduced speed zones are configured during the layout design of the HRC cell, considering the task of both the human and the robot. Conventionally, the size of different zones is a static environment without changes during operation. However, in a symbiotic HRC cell, the robot's speed and task are dynamically changed and monitored by the system. Consequently, the size of the zones can be dynamically modified as well. For instance, when the moving speed of the robot decreases, the potential energy transferred to the human is lower"* [1]. This is a solution that supports the prior suggestion in the introduction, of having volumes or zones as it is called here, were the robot is to have different rules in different zones. The article further includes the concept of having a way to recover form broken rules.

The same paper then continues to discuss how it acts if different tools are used. It is mention that sharpenss of the tool should affect the operation limits and that the speed in a certain case described should decrease by 50% and that the allowed energy transfer is decreased by 75% [1]. This is another thing to consider in the application. however, the energy of the end-effector was not available.

3.2.4 Comment on Safety Studies

The application of the robot arm in this thesis project requires more than just human safety. It also requires object safety and tool safety to not destroy any equipment. Although not necessarily a new field due to the nature of safety, the implementation of object safety can be a branch of the safety for humans but, with a lower safety priority than the safety for humans in the program.

3.2.5 Summary of the Taxonomy

What follows is a concluded taxonomy created after summarizing the most common features encountered in the diffident articles after the literature studies. This was done as finding proper taxonomies which contained the sought after sub Taxonomies for this thesis, were not found. The different taxonomies that were created are:

- Different contact scenarios, see Figure 3.2
    - Quasi-static contact
    - Transient contact
    - Sub-scenarios
        - Secondary impact
        - Unconstrained impact
        - Clamping in robot structure
        - Partially constrained impact
        - Constrained impact
- Different objects
    - Tools
        - Power tools
        - Electric tools
    - Object characteristics
        - Object position
        - Sharpness/minimum contact area
        - (Hazard of the object)

- Different safety measures(preventative)
  - Stop operations
  - Reduced operations
    - Forces
    - Speeds
    - Energies
- Different volumetric definitions
  - Normal operations zones
  - Warning zones (many different safety measures possible)
  - Stop zones
- Different humans with different authorization
  - Operator present
  - Visitor present
  - No human present

A minor modification of [3, 2, 6, 8]

Talking to two researchers familiar with the field here at Lund university, both agree that there currently is no common safety standard, but rather that every company and robot implementation has its own taxonomy for safety, depending on their specific situation. Which is the same conclusion that was made during the literature studies. However, this does not mean that any or some of these taxonomies can not be a basis for a new definition and make the taxonomy as flexible and modular as possible for more general use cases.

One conclusion that can be made form [1] is that it is useful to have zones of operations of which the robot's behaviour is determined by, for example, if the human is present in a zone. These zones are also very adaptable in size and safety thresholds depending on the different scenarios. For example this can depend on if a human is in the zone, the human task in the zone, the tools that are used, the objects in the zone and then what limitations are appropriate for the zone for the set scenario. For example, stop zone, reduced forces and speeds zone, invisible wall zone.

3.2.6 Discussion on the Literature Studies

I believe that the most sensible way of implementing a program for safety for a robot arm is to define operation volumes with operation limits set by the physical environment. I do not however, think there should be predefined safety limits for different hazards categories, which you would just choose from for your application in a volume. For example: object characteristics, tools used, clamping hazards and impact hazards. But rather that every different scenario would have to be evaluated separately to ensure that the safety is thought through thoroughly. However, differentiating between volume definitions like stop, warning, normal operations, and main volume (where all objects should be encompassed by subvolumes) and defining a few safety constraints and operation modes that can be useful in different scenarios, is a more sensible approach. Also defining the main contact scenarios as a suggestion like human contact and object contact but with no deeper description is a good idea. The last safety situation is when the Jacobian matrix of the robot state approaches the singularity or, as said before, when the degrees of freedom of the robot is decreasing.

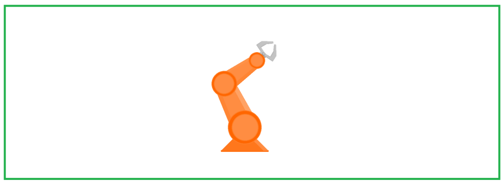
The system that was implemented have these different classifications seen in Figure 3.4. The main volume must cover all subvolumes, the main volume is a border in which the robot should operate, it is not supposed to leave it to ensure safety outside of the main volume. Main volume operation can be defined in different ways, but usually it is a cap on speed and forces, to ensure the well-being of the robot. The reactions defined on the border to "out of bounds" can also vary. For example, the robot can use some kind of invisible wall function, go back recovery behaviour, or even a stop command with an operator needed for recovery. The operations inside the main volume can also be variable depending on input, for example, if a sensitive object is added or removed.

The subvolume is a smaller volume inside the main volume it usually covers an object but not necessarily. This volume should be a zone where a specific operation mode is defined. For instance, speed, force and energy or invisible wall operations or other recovery behaviour. A subvolume can be a stop, warning or other operation area defined by the safety of both objects and human presence. All objects in the main volume should be covered by some sort of subvolume to ensure that the main volume is free from objects.

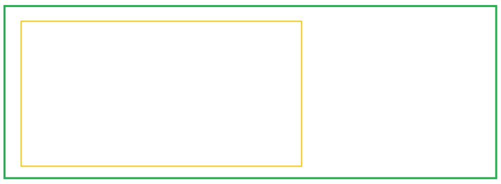
Each volume definition must contain a safety behaviour for all possible objects or humans that can enter it. It must be able to change depending on how the circumstance changes, especially for the human case. Here the word "object" is synonymous with a tool, an unknown object and other robots present.

Each situation both inside and outside of a volume must have a definition of how to treat a contact or collision. Note that depending on the physical and software-wise characteristics it is not always possible to determine if the robot has collided with a human. In the cases that contact is detected but a human contact can not be ruled out, the program should always assume that a human contact has occurred. How to treat a contact depends on the application, whether the contact is necessary, unimportant, undesirable, or hazardous. The next step is to design some kind of recovery behaviour for contact.

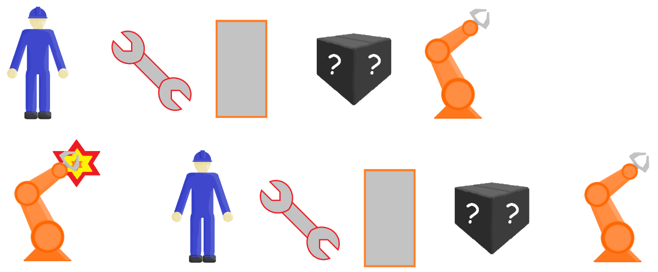
In summary, all safety evaluations for a system must be done separately. All possible hazards and safety operation parameters have to be defined for each possible hazard in the system. After this the defined operations can then be put into subvolumes.



(a)



(b)



(c)

Figure 3.4: In this figure from a)-c) contain all the different objects that exist in the state set. This figure also describes the different objects encountered in the figures in section 4. The green rectangle in a) and b) is representing the outer border of the allowed robot's movements and is called the main volume, and the orange rectangle in b) is representing a sub-volume where other predefined safety rules and safety limits for the robot govern. Underneath the volumes, in c) comes a list of the different objects that can exist in and outside the volume. A blue person represents a human. The wrench represents different tools and the grey rectangle with the orange border represents a known object with a known position. The black box with a question mark on it represents unknown objects with unknown positions and the orange robot arm represents the robot. Underneath this is a similar list but with a robot arm in the beginning with a red yellow star representing a collision. And the same objects as before representing objects that the can be collided with.

4 Results of Safety Studies

The safety studies are a set of thought experiments conducted based on the previous literature studies. The goal of these safety studies was to define a system for categorization safety, in order to have a base for creating a program for robot safety. These were done in 3 main iterations were the first two are described in the section 4.1 and the final safety study is defined in the section 4.2 describing the finalization. The last part contains a schematic map to follow when programming see section 4.3.

4.1 Initial safety studies creating tree structure taxonomy

In the first safety study conducted, the objective was to define all resulting states from the different taxonomy categories based on the first literature studies which were divided into these following categories: A singularity (decrease in the degree of freedom), a form of contact (dynamic and quasi-static), and manipulator characteristics (sharp vs blunt). In difference to [2] the robot arm of this thesis had a scenario which included situations for not only human safety but for object safety as well. There were more possible variables like if a human is present, if there are objects that might be delicate, as well as volumetric limits. These were kept in mind as more details on each of these categories were added. To minimize the taxonomy tree structure the contact subcategories of, sharp, blunt, dynamic, quasi-static, as well as pinching, were lumped together into the supercathegory contact characteristics. The definition of "sharp," what that really means, and where the line between sharp and blunt is drawn, has also been questioned due to the definition being ambiguous. Instead, it was seen as a stabbing incident, which meant that it was not treated differently depending on dynamic and quasi-static but rather as an unacceptable state. As well as contact scenarios, the singularity scenarios were also seen as a categorisation for the taxonomy tree. singularity scenarios was divided into limit angles, volumetric limits as well as limited by objects in the volume. The category of volumetric limits was divided into physical limits and virtual limits. This was done with the thought, that the robot can not only hit a firm object, but it might also be that some virtual area would be a limit. But exactly how to treat it was not thought more of at this stage.

In a meeting with the supervisors, the resulting taxonomy tree was reviewed and discussed. Is it a good idea to start off basing everything on singularity as the first category or if human presence should be the main one, what are the sources for doing so and the missing and lacking reactions for each state and its multidimensionality? The decision was to do more literature studies.

After more literature studies, the focus was more on the reactions of each state. A new, more refined state tree was also created. Which tried to simplify the states by combining some of the previous states into one category and also removing those seen as unnecessary. First, the identification of the safety modes are as follows: normal operations, warning mode, and finally catastrophe mode as primary reactions which contains a list of appropriate reactions for each mode. It then branches in to primary reaction which has a certain set of states, here some of these are, human presence, then object safety, and volume safety. This in turn branches in to more intricate sub-reactions. As mentioned before due to the focus on the reaction rather than the state. This meant that a lot of research went to real physical limits for speeds and volumes and forces etc. For the warning mode there is a greater restriction on movement force and speed than in normal mode. In normal mode, the robot is free to move as fast and with as much force as the task allows it to. While the disaster mode is divided into a few more states. The first branching for disaster mode is divided the same as warning mode but with an immediate stop reaction. Whilst the human part of the disaster mod is divided into two parts. One part describes how to deal with blunt objects involved with a contact scenario, which is a warning mode, that can lead to a stop mode if the parameters for further human safety can not be satisfied. The thought was to prevent a continuous hitting of a human whilst still not being too restrictive when the objects are seen as more or less harmless. Whilst the other is an immediate stop mode due to the immediate danger this state can be to a human. This is due to the object in this state being sharp or semi-sharp. As a clarification, blunt and sharp and semi-sharp are characteristics defined in this thought experiment as something an operator or an engineer is to define for each separate object in the robot workspace.

When presented to the supervisors some critics were if the basis for this division was grounded in some standard taxonomy. That was not the case, as for now no standard had been found but instead only examples. Another topic was that this, even though very interesting, now instead of covering states, covered the reaction and that there must be a way to cover both dimensions together well. In discussions with the supervisors, it was previously mentioned that from their knowledge there was no standard but it was still decided to do some more studies on the matter.

4.2 Safety Study, Creating a State Space

In this part of the safety study the goal is to create a state space which covers the significant states the robot can be in. This is to have a template to follow when creating the safety program to minimize the risk of forgetting to take into consideration an important danger.

Each section for a state was divided into two parts. The first part for each type of situation goes through the different states and explains what they represent and what is included in the state. The second part describes the proposed reactions tied to each state for what correct behaviour can be and what parameters can be defined. The basis for this second part came form the previous literature studies, as well as discussion with knowledgeable people in the field, as well as experience gained while reading the literature studies. For a deeper explanation of the elements in Figure 4.1 and Figure 4.2, see Figure 3.4.

The common recurring suggested reaction category to the states is the *suggested limit operations* as we will call it in the future and has the following proposed limitations for the state it occurs in:

- Suggested limit operations
    1. Maximum end-effector speed
    2. Maximum end-effector force
    3. Maximum joint speed
    4. Maximum joint force
    5. Maximum energy limit of moving mass
    6. Compliance mode
    7. Maximum physical or virtual limitation X (custom limitation)

4.2.1 Robot Safety State Space

a) Inside the main volume

In this state, see Figure 4.1:a, the robot is operating normally with the appointed settings from the operator or programmer. There are no foreseeable limitations to the robot and therefore it can operate freely. All obstacles inside the main volume (the main volume is the volume which covers every allowed position for the robot) should be encompassed by subvolume definitions if possible.

These are the proposed reactions and limitations for this state:

- Volume limits (space limit functions, space limit functions for joints)
- Maximum end-effector speed
- Maximum end-effector force
- Maximum joint speed
- Maximum joint force
- Maximum energy limit of moving mass
- Maximum physical or virtual limitation X (custom limitation)

b) Inside a subvolume

While inside a subvolume, see Figure 4.1:b, the limit of speed, force, volume limits and restricted area in the subvolumes, apply to the robot. These limits are set by the operator or programmer to the desired values. Subvolumes can also be placed inside of each other to create intricate behaviour if that is desired. If multiple subvolumes occupy the same space, the subvolume with the highest safety settings overrules the others in its occupied volume. Subvolumes are quite useful to cover different objects and tools. Especially where operations need to be changed due to, for example, safety around an object, or where stronger forces are allowed, for a specific task. "Compliance

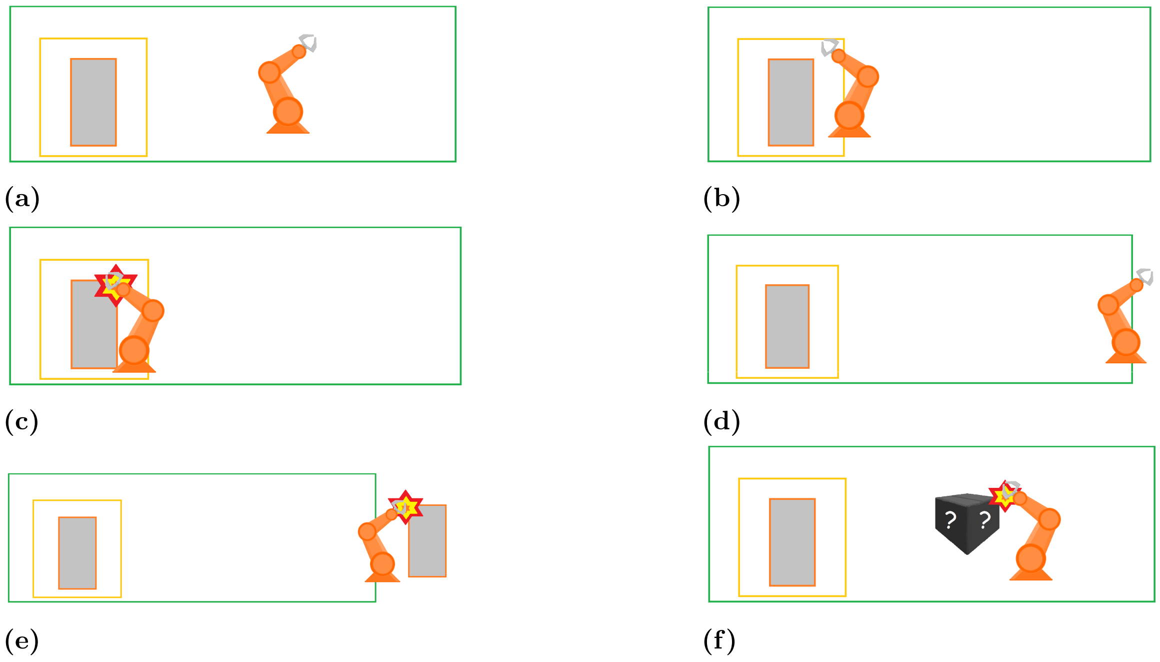


Figure 4.1: Figure 4.1:a The robot arm is inside the main volume. Figure 4.1:b The robot arm is inside a subvolume. Figure 4.1:c The robot arm has come in contact with a known object. Figure 4.1:d The robot arm has left or partially left the main volume. Figure 4.1:e The robot arm has come in contact with an object whilst also being outside or partly outside the main volume. Figure 4.1:f The robot arm has come in contact with an unknown object.

mode" is when the robots motor stiffens goes to 0, making it comply with forces put on it hence why its called "compliance mode".

These are the proposed reactions and limitations for this state:

1. Stop command and call for operator
    - (a) (Optional) Compliance mode
2. Stop command and recovery behaviour (If recovery behaviour fails return to step 1.)
    - (a) Return to the previous step outside subvolume
    - (b) Move in the normal direction to the border surface away from the subvolume
    - (c) Move away from the center of the subvolume
    - (d) Recovery algorithm X (custom algorithm)
3. Suggested safety violation operations or subvolume operations
    - (a) Suggested limit operations
4. Invisible wall behaviour
    - (a) Forces in the normal direction to the subvolume surface in the subvolumes direction set to 0
    - (b) Forces towards the centre of the subvolume set to 0
    - (c) Invisible wall function X (custom function)

c) Contact or collision in a subvolume

Contact with an object in the subvolume, see Figure 4.1:c, can be either positive or negative depending on the intended use and the object's characteristics depending on safety. This is defined by the programmer or operator of the system for the specific subvolume.

These are the proposed reactions and limitations for this state:

1. Stop command and call for operator
   (a) (Optional) Compliance mode
2. Stop command and recovery behaviour (If recovery behaviour fails return to step 1.)
   (a) Return to the previous step inside or outside subvolume
   (b) Moves away from the centre of the subvolume until the outside
   (c) Move in the normal direction to the border surface away from the subvolume
   (d) Recovery algorithm X (custom algorithm)
3. Suggested contact operations
   (a) Suggested limit operations
4. No change in safety behaviour

d) Out of bounds or outside main volume

In this state, see Figure 4.1:d, the robot has left the main volume or operating area and is therefore out of bounds. Once the robot is in this state, it is difficult to predict what will happen next and contact with any object or human is possible making this state a safety risk. Some safety measure that ensures that a human is not present in the reachable area of the robot or close to the main volume must be present. However, in some cases, the environment around the main volume is not populated with either objects or humans that the robot can reach and is therefore somewhat safe to move through for the robot. The purpose of the main volume is to encompass the entire working area and all operations of the robot.

These are the proposed reactions and limitations for this state:

1. Stop command and call for operator
   (a) (Optional) Compliance mode
2. Stop command and recovery behaviour (If recovery behaviour fails return to step 1.)
   (a) Return to the previous step inside bounds
   (b) Move towards the centre of volume until inside
   (c) Move in the normal direction to the border surface in direction of the main volume
   (d) Recovery algorithm X (custom algorithm)
3. Invisible wall behaviour
   (a) Forces in the normal direction to the volume surface in out-of-bounds direction set to 0
   (b) Forces away from the centre set to 0
   (c) Invisible wall function X (custom function)

e) Out of bounds contact or collision

In the case, see Figure 4.1:e, the robot has exited fully or partially the main volume or operating area and has come in contact with or collided with some unknown object or possibly a human, we can no longer guarantee safety and must therefore come to an emergency stop. In the case that the volume outside of the main volume is known, contact is still unexpected, therefore an emergency stop is mandatory.

These are the proposed reactions and limitations for this state:

1. Stop command and call for operator
    (a) (Optional) Compliance mode

f) Contact with unknown object

In this case, see Figure 4.1:f, the robot is in contact with an unknown object or an object with an unknown position. Contact with an unknown object is usually unacceptable and needs human intervention. However, in some cases these objects are not unknown objects, but rather it is their position that is unknown, or somewhat unknown and contact is not only allowed but desired for normal operations. An example of a situation where the object is not unknown but the position is, is were a box is to be moved by the robot arm from one point to the other but the exact position of the box is impossible to know. In this case, the treatment of this object should be defined under the limits for the main volume.

These are the proposed reactions and limitations for this state:

1. Stop command and call for operator
    (a) (Optional) Compliance mode
2. Suggested contact operations
    (a) Suggested limit operations
3. No change in safety behaviour

4.2.2 Robot-Human Safety State Space

a) Human in the main volume

If a human is present in or close to the main volume, see Figure 4.2:a, all operations must be adapted to only operate in a way that can ensure the person's safety. What is safe is different from case to case and must be evaluated for every specific situation. One example that can determine the settings is any tool used by the robot, if it is sharp or blunt. Another example is the specification of the objects in the environment of the main volume operations. There are many more factors needed to determine the safety and these two examples are not guidelines.

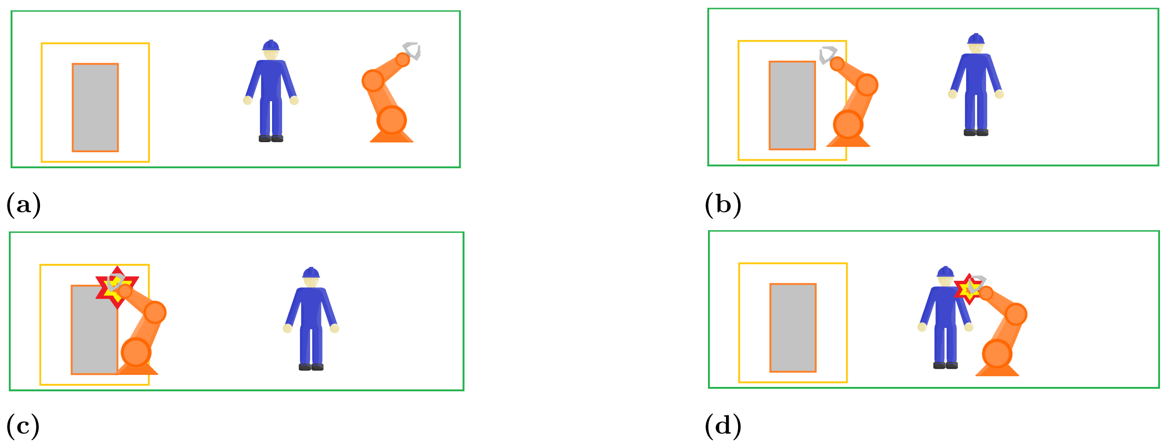


Figure 4.2: Figure 4.2:a The robot arm is in the main volume while a human is also present in the main volume. Figure 4.2:b The robot is in a subvolume while a human is present somewhere in the the main volume, possibly also the subvolume. Figure 4.2:c The robot arm has come in contact with an object whilst a human is present somewhere in the main volume. Figure 4.2:d The robot arm has come in contact with or collided with a human inside the main volume.

These are the proposed reactions and limitations for this state:

1. Stop command and call for the operator or wait until the main volume is clear of any humans
   (a) (Optional) Compliance mode
2. Suggested Human safety operations
   (a) Suggested limit operations

b) Inside a subvolume, while human in the main volume

Just like when a human is present in the main volume, the operations for the main volume need to be safe. The same is the case for subvolumes that a human can enter, see Figure 4.2:b. These volumes are also useful to cover different objects and tools with which usually need a harder safety limit. Sometimes making the subvolume a restricted area while a human is present is important to ensure safety. However, it might be that a subvolume is safe even though a human is present due to the fact that the area is unreachable for the human.

These are the proposed reactions and limitations for this state:

1. Stop command and call for operator or whait until the main volume is clear of any humans
   (a) (Optional) Compliance mode
2. Stop command and recovery behaviour, move away from subvolume (If recovery behaviour fails return to step 1.)
   (a) Return to the previous step outside subvolume
   (b) Move in the normal direction to the border surface away from the subvolume

    (c) Move away from the center of the subvolume
    (d) Recovery algorithm X (custom algorithm)

3. Suggested human safety operations
    (a) Suggested limit operations

4. Invisible wall behaviour
    (a) Forces in the normal direction to the volume surface in the subvolume direction set to 0
    (b) Forces towards the centre set to 0
    (c) Invisible wall function X (custom function)

c) Contact or collision while human in operating area

The robot has collided with something in a subvolume while a human is present in the main volume, see Figure 4.2:c. This could be an object, an unknown object or even a human. This is unacceptable because there is an extra risk of clamping a human between the object and the robot. In some cases as mentioned, human contact can be defined separately but the extra risk of clamping is still there. Also, compliance mode in this case is not necessarily possible. This can be due to the fact that the weight of the load on the end-effector might vary, causing it to fall on someone if compliance mode is applied.

These are the proposed reactions and limitations for this state:

1. Human contact or collision, is hazardous
    (a) Stop operations and call for the operator until the human leaves and the operator acknowledge.
        i. Compliant mode if safe to do so.
    (b) Safety stop X (custom function)

2. Human contact or collision, is allowed
    (a) Stop operations, compliant mode and safe operations (then move on to c)
        i. Suggested limit operations
    (b) Stop move, move away from contact area (then move on to c)
        i. Suggested limit operations
    (c) If contacted twice in a row while a human is in the main volume (then move on to step 1.)
    (d) Human-contact behaviour X (then move on to step 1.)

3. Human collaborative mode
    (a) Collaborative behaviour X (custom behavior)

d) Human contact or collision

Human contact is when the robot has come in contact with an object or human inside or at the border of the main volume while a human is detected near the main volume, see Figure 4.2:d. If it is possible to differentiate between object contact and human contact in some other way, that will be the definition of human contact. In most cases, human contact is unacceptable and demands a safety stop. However, in some rare cases human contact is intended and follows under the human safety operations defined.

These are the proposed reactions and limitations for this state:

1. Human contact or collision, is hazardous
    - (a) Stop operations and call for the operator until the human leaves and the operator acknowledge.
        - i. Compliant mode if safe to do so.
    - (b) Safety stop X (custom function)
2. Human contact or collision, is allowed
    - (a) Stop operations, compliant mode and safe operations (then move on to c)
        - i. Suggested limit operations
    - (b) Stop, move away from contact area (then move on to c)
        - i. Maximum end-effector speed
        - ii. Maximum end-effector force
        - iii. Maximum joint speed
        - iv. Maximum joint force
        - v. Maximum energy limit of moving mass
    - (c) Compliance mode
    - (d) Maximum physical or virtual limitation X (custom limitation)
    - (e) If contacted twice in a row while a human is in the main volume (then move on to step 1.)
    - (f) Human contact behaviour X (custom behavior) (then move on to step 1.)
3. Human collaborative mode
    - (a) Collaborative behaviour X (custom behavior)

4.3 Schematic of the Proposed Programming Database Structure

After the safety studies a summary and a simplification was needed for a more programming-friendly model. This was done by creating the following data base structure see Figure 4.3. This structure needed to have the ability to be flexible enough to cover all the different states created in Section 4.2 but also simple enough to be easy to use. It was here decide to have a root defined by a volume which would hold a shape and govern a sub category called mode, the mode would contain all limits and reactions discussed in Section 4.2.

The database structure was done in the following way: Firstly, the definition of subvolumes, which contains a few parameters which define the data sets such as an identifier, as well as a priority number for keeping track of overlapping subvolumes. This is then followed by the volume limits itself and finally a list of mode indexes which contains all work modes that exist within a certain volume. It also contains a separate coordinate system which can be changed on the fly during the operations. This is due to the fact that some objects in the area can move, for example, if there are two robots working together. Each subvolume also contains a few triggers, mostly volumetric but it can be defined as a plane, that activate the subvolumes, and its modes of operation.

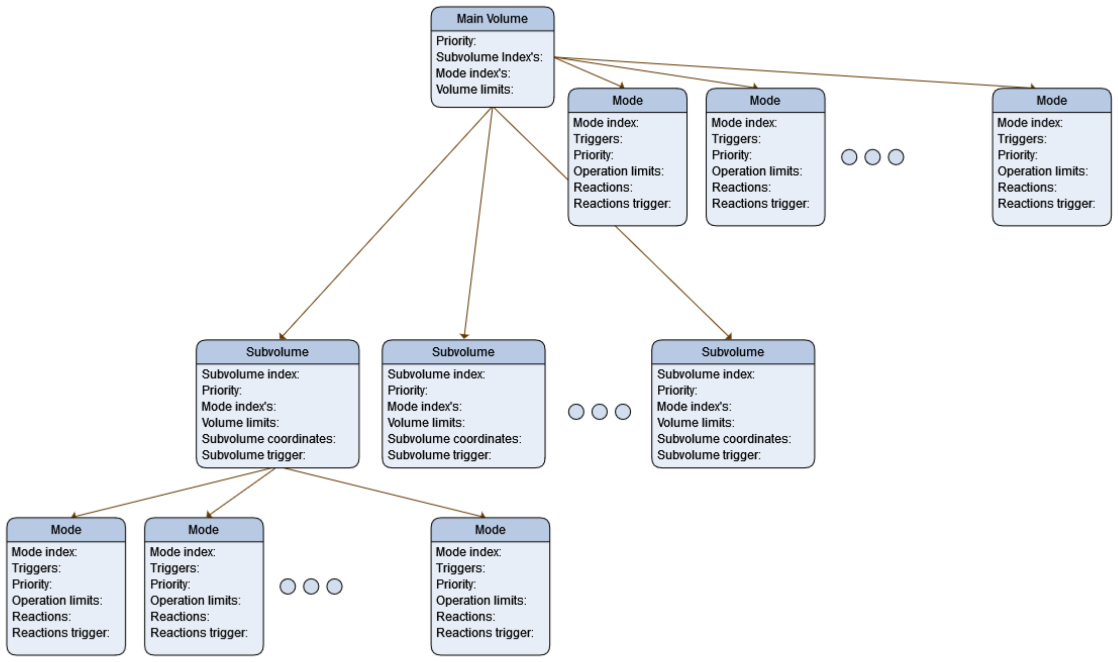


Figure 4.3: The first box, centre middle top, describes the main volume which should contain different subvolumes and the different modes of operation the main volume can be in. The modes that are connected to the subvolumes and the main volume are operational modes for limits and rules.

The highest authority is the main volume, the main volume is where the robot is to be operating, it is to encompass the entire volume of where the robot can go. It is different to the subvolumes in the sense that it does not contain any triggers. This is because when no triggers from other subvolumes are active, the operations go back to the main volume operations.

The main and sub-volumes contain the modes of operations. These modes have different contents, with each mode having a main or sub-volume, an identifier, a priority marker and a trigger. This is due to the fact that even though the volume is defined in the main or sub-volume content, time is a dimension of its own and at certain times or at certain states, some things can be allowed but in other times and states they are not. For example, an object can be picked in one please and pushed in another, allowing different kinds of contact in different times. Each mode contains a set of operation limits, which can be, speeds, forces, angles, and so on. *Reactions* is a list of possible reactions to a certain breach of operation limits. That can be, stopping and calling for an operator, in the most aggressive reactions, but sometimes a simple slow down, or even move back, can suffice.

Reaction triggers are triggers that cause a reaction. they are very similar to operation limits, however, the operation limit does not necessarily trigger a reaction. This can be due to the mode not being triggered. For example, pick an object, place an object or do not touch an object mode. If the robot is in touch mode but the operation limits for pick mode trigger, this should not lead to a reaction and thus the *reaction trigger* does not fire.

5 Safety Program Implementation

5.1 Studying the Tools and Software

The following tools were studied at the beginning of the project:

ROS or robot operating system is a middleware that is used for communication. Communication is done through something called a master which manages the different messages going back and forth between the different programs [14]. This middleware is used to connect the different programs bot for the internals of SkiREIL but also for connecting the safety program to the robot controllsystem. It was of course important to learn this program in order to be able to use the communication features present. To learn ROS a tutorial from ROS's own website [15] was followed.

SkiROS2 is a skill-based platform [16] for complex behavior trees [17]. This is an important part of the SkiREIL package used in this project. SkiROS2 was important to studdy as it contains the AI decition system and this was to be contained by the safety system. SkiROS2 was studied by reading the instructions on the website [17].

SkiREIL is a package [18] which contains the programs that were to be worked on in this thesis; SkiROS2 is one of the policies that is supported by SkiREIL. SkiREIL is a framework that combines reinforcement learning methods with skill-based behavior trees.

RosMari is a safety system programming language [3] that operates separately from the other programs. The reason behind that is to ensure that the safety program is always able to be executed no matter what happens to the main system. Another purpose of RosMari is also to simplify programming in order to minimize the risk of code errors. RosMari was created in a research project on the university inspired by the article [3]. This article [3] was studied to understand the implication of the safety program and its functionality. RosMari was an inspiration for this thesis with its features of having separate processes and programming simplicity.

5.2 Program Environment Description

This section describes the environment the programs for this thesis was created in. the reason behind this is to ensure understanding of how and why things were done.

The SkiREIL framework that was to be worked on already contained a control system for the movement of the arm. SkiREIL also contains a part for decision making, this is the machine learning part, determining new positions to move to as well as giving the robot commands for force and speed. A constant force is to be applied in the

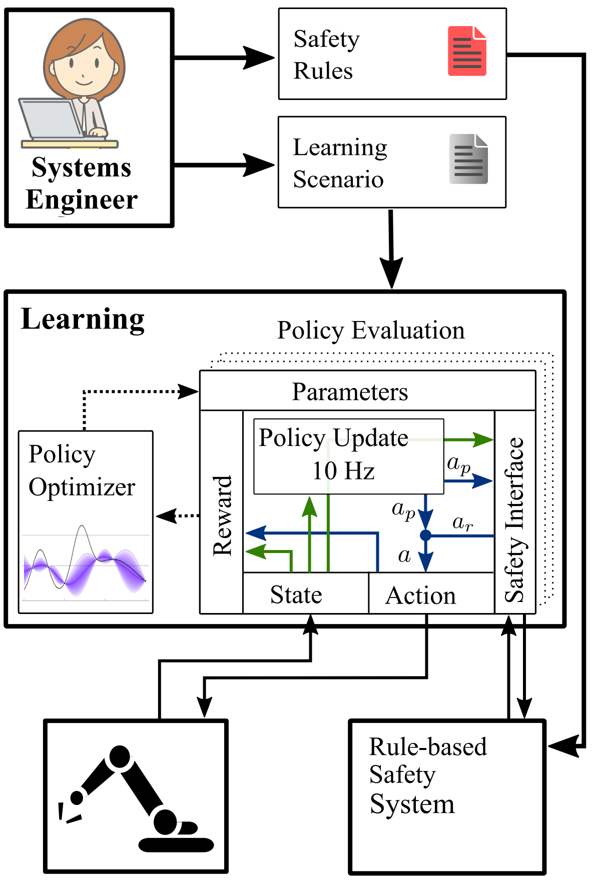


Figure 5.1: This shows the architecture of the SkiREIL system, the parts being worked on have been the safety interface and the rule-based safety system or what is called the safety program. The idea is that a system engineer will be able to work on the safety rules of the safety program and define the learning scenario. The image is taken from [19].

commended direction when a force command is given. The program works with a state vector as well as an action vector when updating the desired commands and these were sent back and forth between different parts in SkiREIL. This was done to ensure that the orchestra of internal simulation, controller, output signals, and decision-making, were synced up correctly to ensure the functionality of the program and the robot arm.

As can be imagined from the introduction, the task here is to intercept the state vector and the main update of the action vector and ensure that safety is preserved. As mentioned this state vector contains end-effector position, end-effector orientation, forces, joint position and joint orientation, but here only the end-effector position and orientation will be of interest. From the action vector, however, both reference position, reference orientation, as well as reference forces and reference stiffness will be needed [20].

Now to what has been added during the thesis. To make sure that safety is always ensured, a safety program will be running separately on an isolated thread. Therefore two things were added: First a safety program to handle the safety rules, and secondly an interface program between the SkiREIL library and the safety program that manages the communication between the two.

5.3 The Safety Interface

See Figure 5.1 for a detailed image of the program.

At the unchanged state of the SkiREIL program, to send the state vector and the action vector to the safety program, just as they are, makes no sense. This is because the two vectors at the unchanged state can be hard to understand for someone working with the safety program. Therefore a Safety interface class was constructed which would divide up the state and action vector into more palatable chunks that we can call physical traits, like the previously mentioned positions, orientations, reference values of position, orientation, force and stiffness. Also, the speed value of the arm was needed, this was however not part of the state vector that was being acquired and was instead added separately to the existing chunks.

The configuration of the safety scenario needs to be uploaded to the safety program in which it will operate. It will need limitations and rules to be operated on. This is done by a separate function, but still in the safety interface part of SkiREIL. This function receives the configuration which is defined in JSON with all safety limits and rules. It then converts it to a string and sends it over to the safety program over ROS service message. This is done first in a separate function in the safety interface to minimize unnecessary calls during the safety checking.

After the setup of the safety scenario has been communicated to the safety program, the state vector and action vector are divided up into their physical traits and prepared to be sent to the safety program according to their respective data type over a ROS service message. Following this, the safety interface waits for a reply from the safety program. Here exists a fail-safe: If a reply can not be established, the safety interface will return a failed safety check to SkiREIL which in turn throws an exception and subsequently brake any further robot actions.

If the communication works properly, a response will be received from the safety program consisting of 3 parts. The first part is a string which contains the message about the breach situation. The second part contains a possibly altered version of the action vector, lets call it the *reaction vector*. this *reaction vector* is then used in the safety interface to overwrite the current action vector. The third part contains a set of booleans connected to different reactions predefined in the safety interface.

The response from the safety program depends on the situation and exists in three different cases. Case one, if no rule has been broken, an empty string will be received by the interface program from the safety program, as well as a *reaction vector* equal to the action vector, and a batch of false Boolean's standing for that no reaction should occur. The following two cases are cases where a rule is broken. In case two the string, will contain a text describing the breach and which rule that has been broken. And a change in the *reaction vector* triggering the reaction of the robot, but with false reaction booleans. In the third case, as before, the string contains the breach info, but the *reaction vector* being equal to the action vector, it is the reaction booleans that will return true and trigger their respective pre-programmed reactions to the robot.

Pre-programmed reactions are the reactions that exist in the safety interface. These reactions work by checking their designated reaction boolean value coming from the safety program. If such a reaction boolean is true the pre-preprogrammed reaction ensues. These reactions that are hard coded in the safety interface are the slowdown reaction and the set forces to 0 reaction. The slow down function is a very simple function that sets the reference position values to the current position values, thereby forcing the controller to want to stay in the position of the previous step.

The safety status is checked for every RL step. In the current set of experiments, this is 10 Hz. This value can of course be adjusted to a desired value for the custom-chosen scenario.

5.4 The Safety Program

The safety program first receives the JSON configuration with all the limits and rules and saves them in a Python dictionary. Secondly, it receives all physical traits from SkiREIL over the ROS service message. Each time the safety program receives the physical traits, it pairs each physical trait with its corresponding rules and limits and compares them to check if any rule is broken. For example, absolute speed is paired with every speed rule and checked for any breach of value. The safety program is able to receive updates on its environment at any time from the JSON file during the robots operation, this adds the multidimensionality of time to the safety program if so is desired.

Rules can be somewhat complex. A set of rules that needs to be breached together to trigger a reaction is called a *block*. So each *block* contains multiple rules. For example, one *block* can contain one rule for speed, another for force and one for a physical breach volume for the position of the end-effector. In this case, if the speed exceeds its maximum allowed value but the other limits are not breached the situation will not trigger a reaction. One scenario in which such a rule can be applied could be because outside this volume a higher speed is allowed or because it is the combination of speed and force in this volume that would be undesired. In any case, all rules need to be broken simultaneously in a *block* for a reaction to trigger.

A *block* does not only contain rules, each *block* also contains a unique ID number for the breach, a “limit category” information message string in case of a breach and the reaction itself as a string, as well as a priority number. For the priority number, a low number means that the *block* is having a higher priority, in case multiple competing overlapping rules trigger their reactions simultaneously.

After all traits have been checked against their corresponding rules, a list of all detected breaches that satisfy a *block* is stored and then sorted according to each *block's* priority number. Then these *blocks* are filtered leaving only those with the highest priority number. At this point, all the different *blocks* reactions are stored in a dictionary of reactions as well as all information connected to the *blocks* in question. Here lies an exception; If any of the *blocks* contain the reaction “stop”, this will be the only reaction allowed in the dictionary of reactions. Subsequently if a “stop” is detected, only one *block* will continue to the next part.

In the next part, all dictionary keys (keys are keywords that can be used in a dictionary data structure to receive a variable) which are the reaction keywords from each reaction will cause certain changes to the *reaction vector* that are to be sent back to safety interface as well as setting reaction booleans true and false accordingly. One information string about the breach is also created which contains both the information message string, the id and the reaction for each *block* in one string, and if multiple *blocks* are triggered at once the strings get added together into one big message. Finally, all physical traits as well as the message string and the reaction booleans are sent back to SkiREIL.

There are two reactions that cause the change in the reference value. The first one is the “stop” reaction and the second is the “return to origin” reaction. The “stop” reaction saves the first pose that caused the breach and sets the reference pose trait to this position every time the safety program is called. The “return to origin” reaction uses the recording of all the different physical traits since the start of the episode to find its way back to a point close enough to the start point. This is done by setting the reference pose trait to the recorded path taken.

However, when the SkiREIL is training the RL model, multiple threads of SkiREIL can be running simultaneously. Having a memory of the states of the robot arm is then a problem as mixing up the saved states of the different threads is bound to happen. Therefore the safety program is to be run in a different mode without a memory of previous states. Instead it is operation in a in out setting. Therefore when training, both the stop and return reactions are replaced by a stateless stop reaction which slows down the robot by setting the reference pose trait to the current pose while the reaction is triggering. As this does not rely on the saved states it can just return the correct reaction for the current thread.

The JSON file containing the safety *blocks* where the rules follows a particular set-up according to table 5.1.

In the Table 5.1 the dots are replaced with rules for an imaginary case. How to create *blocks* is described in the following way. A *block* works like an “and expression” meaning that all statements inside a *block* need to be fulfilled to trigger a reaction from the safety system. The same statement/category can not be added more than once. For example, there is only one “reaction” in each *block*. The same goes for the rule statements, for example: *““velocity”:0.3”*, and for all statements. Although not all statements in a *block* are rules. Some are identifiers for debugging. Like: ”limit category” and ”combine id”. The priority number can be used as an “or” statement if the priority of multiple *blocks* are the same. If this is the case, all statements will return a reaction for example, if two *blocks* are satisfied, it will return both reactions. Two exceptions to this rule are “stop” and “decelerate”. “stop” is prioritized over the other reactions and “decelerate” is prioritized over the “return to origin” reaction. If one priority is higher than the other and both *blocks* are triggered only the reaction

```
"safety_interface":        "safety_interface":
   {                          {
   "limit category": "x",     "limit category": "Velocity exceeded",
   "Human presence":bool, --> "Human presence":False,
   "combine id": int,         "combine id": 1,
   "priority":int,            "priority":1,
   "reaction":"x"             "reaction":"decelerate"
   .                          "Velocity":double
   .                          "Box_info":"position inside 3D box volume"
   .                          "x_max": 1,
   },                         "y_max": 1,
                              "z_max": 1,
                              "x_min": 0,
                              "y_min": 0,
                              "z_min": 0,
                              },
```

Table 5.1: To the left is a JSON format representation of what a *block* looks like. The text to the left contains the mandatory parts of the *block* and the dots symbolize the different limits that can be added. To the right is a JSON format example of what a *block* can look like when it is filled in. A *block* is comparable to the modes in section 4.3 but with out the hierarchical structure of the volumes on top of the modes

of the *block* with the higher priority will run.

5.4.1 The Safety rules format

Following is a description of how to write rules to the safety program from the Json format, and an explanation of what goes into each rule. The first half describes the mandatory entries that needs to be filled in for the *block* to function. These are as follows:

- Limit category contains a string describing the situation that caused a reaction to occur so that it is easier to debug, this is done according to the example:

  ”limit category”: ”position inside 3D box volume”

- To the specific ID a number is written to help identify the specific rule that was broken in case many *blocks* have the same limit category text to help debugging. Here follows how to do that:

  ”combine id”: int

- The priority number serves the purpose of structuring the *blocks* in an organized way. This works by letting the smallest integer have the highest priority. the priority number can be used as an “or” statement if multiple rules have the same priority.

  ”priority”: int

- A boolean is used to mark if the *block* applies to a situation with or without a human presencet. If the input is not important and the same behavior should occur for both cases, two identical *blocks*, but with the human boolean set to true in one of them and false in the other can be created.

  "Human presence": bool

- The reaction string is also part of the mandatory parts. following is how to write the rule and a list of possible reaction commands.

  "Reaction":string
  For example: *"stop"* , *"decelerate"*, *"Turn off active force"*, *"return to origin"*, *“”*. See also "origin tolerance" in the next section. This “” is an empty reaction if no reaction is desirable or if a rule is to be skipped. For example, if a small hole in a restricted area is desired, which the robot is allowed to go through, the empty reaction can be useful.

This finishes the mandatory attributes needed in each rule *block*. The following are the optional rules and limits that can be added in a *block*:

- To identify if the limit is broken inside or outside a defined volume. "position inside 3d box volume" will trigger a reaction when the position is inside a given volume and thus a broken rule. The opposite is true for the "position outside 3d box volume", which will trigger a reaction when the value of the robot is outside a certain volume. Only a maximum of one 3d box volume may exist in each block.

  "Box_info": string
  Existing alternatives "position inside 3D box volume", "position outside 3D box volume"

  - The following are the limits of the 3d box volume with min and max values. It is possible to choose between one and all the limits below for the 3d volume defined. Of course, this means that the box is not always a conventional box but a box that can span to infinity in a set of directions depending on how you define the limits.
  - "x_max": double,
  - "y_max": double,
  - "z_max": double,
  - "x_min": double,
  - "y_min": double,
  - "z_min": double,

- This is how the maximum end-effector velocity limit is defined.

  "Velocity":double

- Defines the maximum allowed force limit to the end-effector position.

  "Force":double

- Origin tolerance is the tolerance that is the allowed distance for the end-effectro position to the recorded position, for it to continue to the next position for the "return to origin reaction". This must also be tied to the "return to origin reaction" otherwise it does not work.

  “origin tolerance”: double

6 Simulation Experiments

The robot arm has been tested in simulation to see if it reacts accordingly to different breaches. The “return to origin” reaction as well as the “stop”,” reduce speed” and ”reduce force” reaction has been tested respectively. Figure 6.1 shows the start configuration and the successful completion of a peg-insertion task. The next session in Figure 6.2 shows that the robot arm is stopped when reaching the defined rule limit and then returns to the origin as was expected. Then it is shown in Figure 6.3 that the stop function, which depends on the situational inputs, work on the scenario of a human entering the area. This scenario is set in the JSON file to limit the operations and not allow it to get too close to other objects. It was also compared in Figure 6.4 how well the stateless stop function is, in comparison to the return to the origin, which works the same as the normal stop function at the beginning of a breach. In Figure 6.5 it is shown that the safety system works not only for the scenario in which it was tested and programmed, but also for a completely different task showing the adaptability of the safety systems.

6.1 Testing the Correct Behavior

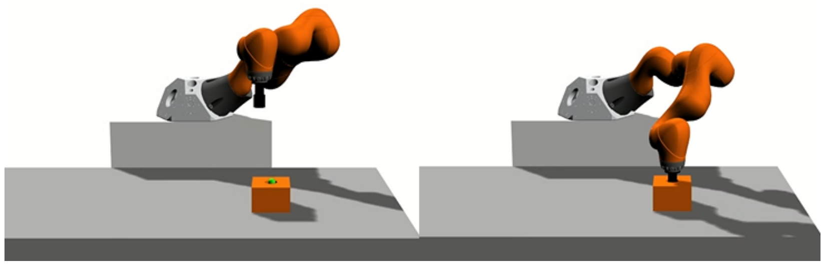


Figure 6.1: Peg insertion task. Start(left) keyframe for the robot arm where the controller is behaving normally (start to the left and ending to the right)

As the first proof of concept, the robot arm is to start 4 to 5 decimeters in a random spot over its target, in this case, a cylindrical hole. The arm itself is fitted with a peg, and this cylindrical peg carries the end-effector position. The objective of the fully trained robot arm is to insert the cylindrical peg in the hole. The robot’s task was to simulate a factory process of placing cylinders in a motor block. As can be seen in Figure 6.1 this was successfully done, showing that it is possible to set up safety rules that do not intervene during normal operations of a robot.

6.2 Testing the Recovery Behavior

The settings for the controller (not the safety) were then changed in a way that the robot arm would try to go far away from its target and do so at high velocity. The safety rules set (the same as the successful peg insertion) were that of a 3D box surrounding the area, the x and y coordinates are in the same plane as the table and the z is in the vertical direction. There was a small buffer between the table and this defined box, as well as limits set in the x and y direction to prevent the arm from going too far away from the objective. In the positive z direction, which was in the up direction, a more lenient limit was set. As Figure 6.2 shows the robot arm that started in the normal position and moved along the blue path was successfully stopped from exceeding the box limits shown in red and was then returned to the start position along the same path shown in yellow. See also Table 6.2, for the first value that breached the rule.

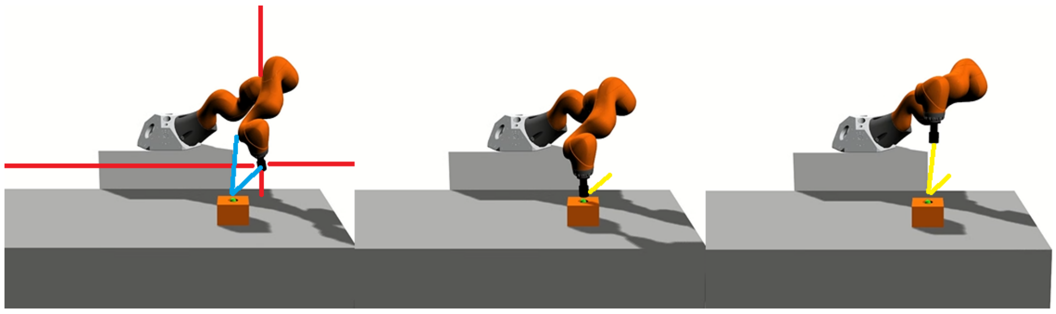


Figure 6.2: Breach, recovery and end of recovery keyframe, left to right, for the robot arm where the controller is overshooting (path marked in blue) a lot, causing a breach of rules (marked in red), the return to origin path is marked in yellow.

6.3 Testing the Stop Behavior for Human Safety

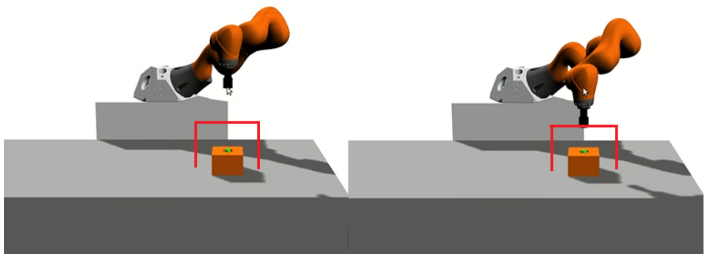


Figure 6.3: Start and end keyframe, from left to rigth, for the robot arm when a human is present causing stricter rules which in turn caused a breach which caused the “stop” reaction. The rule is marked in read in this figure.

Further, the presence of a human was set to true because the safety rules are set so that if a human is present, the ”return to origin” reaction is replaced with the ”stop” reaction and an additional box is checked around the target to prevent clamping. The stop function was shown to work as intended, shown in Figure 6.3, where the robot arm was stopped until the end of the episode.

6.4 Memory-less Stop Behavior

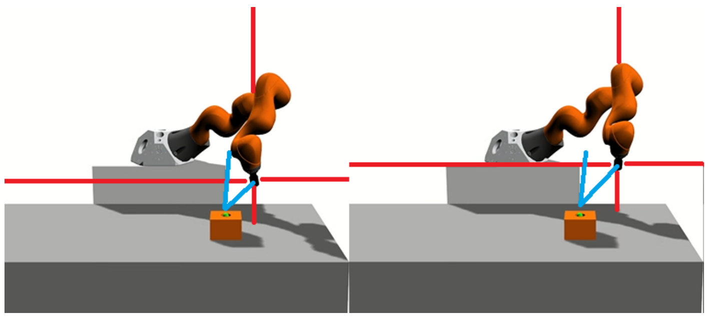


Figure 6.4: Return-to-origin reaction furthest point, to the left, in comparisons to simple stop furthest point, to the right, after the breach. The figure shows the path in blue taken by the end-effector and the breach point in red.

The safety system was then run in stateless mode, the mode for when the robot is in training. The robot arms controller was still set in the same way so the arm goes away from the target with increased speed. The arm was successfully stopped, however, the distance travelled from the box limit point to its stopping position was increased as shown in Figure 6.4. This was to be expected since the stateless stop reaction is a much simpler method with a moving stop position, whilst the stop and return-to-origin reactions respectively move to a memorized position of the first incursion of the rule, giving it a stronger braking ability. Just to clarify, the moving stop function works by setting the reference value to the current value for the controller until it gets an update of the position 0.1 s later in this case.

6.5 Testing with Obstacle

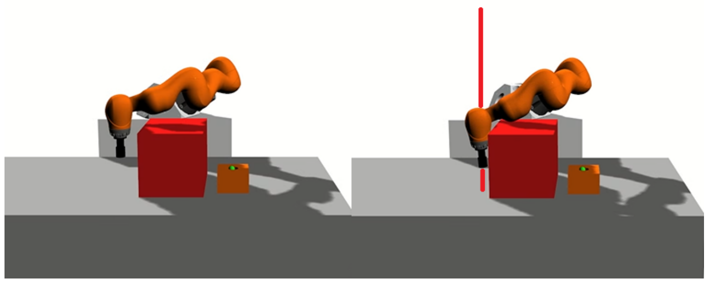


Figure 6.5: Start and end keyframe from left to rigth for another scenario, the *avoid-obstacle* scenario, where the robot stops. The new senario is that of a obstacle betwene the robot arm an the cylindrical hole. The robot arm end-effector is not allowed to come close to the read obstacke box. The breach of the limit is shown in the last keyframe with the read line showing where the limit is.

Now to see if this was a general solution or a more specific one, the same proposed safety rules and reactions were added for the *avoid-obstacle* scenario shown in Figure 6.5. This was a new scenario not encountered during the design an programming phase. Similar rules around the cylindrical hole were added, as well as a box volume limit surrounding the read cube in the workspace. The arm, as seen in Figure 6.5, was stopped before the end-effector (the center bottom of the peg's coordinates) hit the obstacle. When it comes to the rest of the arm, which is touching the obstacle, it is to be mentioned that only the end-effector position was known to the safety system at the time. The concept is clearly shown for the end-effector position, causing a reaction, which it does here clearly. As to the rest of the arm, this is left for further development and discussion.

6.6 Further Data on the Tests

Figure 6.6 Shows a snippet captured a few milliseconds before the breach that lead to the recovery behavior from Figure 6.2. This snippet shows how the speed limit is breached and how it is immediately detected, it also shows how the reaction is able to slow down the robot arm to a safe speed. The same breach of speed and slowing down can also be seen in Table 6.1

Both Table 6.2 and Table 6.3 as well as Table 6.5 show the instance in time when a volume limit is breached as well as the volume limits. The tables also shows the exact position the end-effector of the arm was in at the time of the breach as well as the reaction that followed. Table 6.2 shows the volume breach that caused the recovery behavior to start. Table 6.3 shows the volume breach that caused the stop reaction. And Table 6.5 shows the breach of the *avoid-obstacle* scenario.

In Figure 6.7 as well as Table 6.4 the breach of both force and velocity can be seen. The figure shows a sniped that clearly shows how the velocity of the arm is slowed down every time it exceeds the limit. The table shows an instance of when the velocity breach is detected with in the same time frame. The figure also shows the force breach and the Boolean reaction trigger to the breach of the limit. Table 6.4 also shows an instance were the limit for force is breached and that a reaction is triggered. Note however that in these examples the limit is set very low to achieve as frequent breaches of the limits as possible to test the safety system.

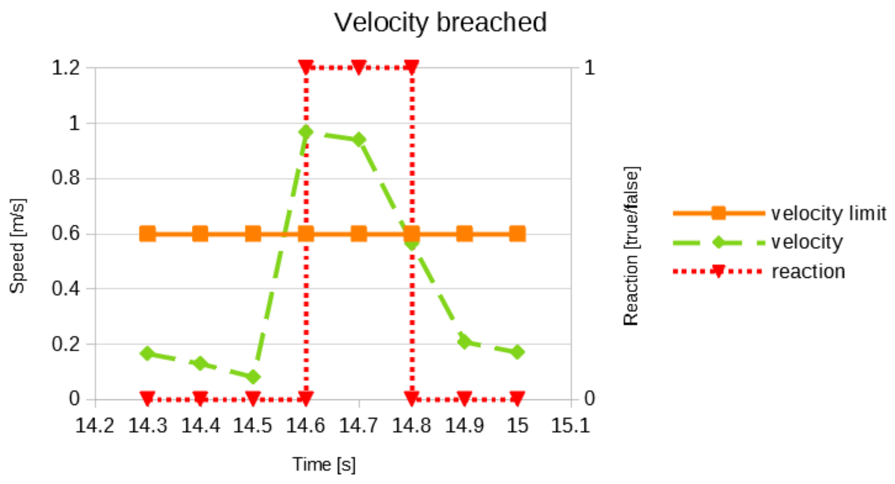


Figure 6.6: The diagram shows the recorded data around the event in Table 6.1. The solid orange line is the limit, the spaced green line is the measured value and the red dotted line shows if a reaction was triggered with its y axis to the right as a boolean.

timestamp 14.6	rule	measured value		reaction
maximum velocity	0.6	0.9673829787	decelerate	TRUE
Human presence	FALSE	FALSE	Text output:	”decelerate”
timestamp 14.7	rule	measured value		reaction
maximum velocity	0.6	0.9401015946	decelerate	TRUE
Human presence	FALSE	FALSE	Text output:	”decelerate”
timestamp 14.8	rule	measured value		reaction
maximum velocity	0.6	0.5645789733	decelerate	False
Human presence	FALSE	FALSE	Text output:	

Table 6.1: Rule breach in yellow reaction in green, showing how the decelerate reaction decreases the speed of the end-effector from 0.96 m/s to 0.94 m/s and finally when there is no more breach the speed is decreased to 0.57m/s of an interval of 0.1s between measuring points whilst breaking a new rule in Table 6.2. See also Figure 6.6.

		rule breach outside designated volume			
timestamp 14.8	rule max	measured value	rule min		reaction
x	-0.45	-0.4075623845	-0.75	Reference x	-0.4742667151
y	0.35	-0.09492050803	-0.35	Reference y	-0.0719180574
z	1.45	0.9302736751	0.649	Reference z	0.9537281945
Human presence	FALSE	FALSE		Text output:	”return to origin”

Table 6.2: Rule breach in yellow reaction in green, showing the reaction of the return to origin function giving back the previous x,y,z position as the new reference position. x, y, z are the Cartesian coordinates for both the limits and the measured end-effector position.

		rule breach inside designated volume			
timestamp 14.5	rule max	measured value	rule min		reaction
x	-0.4	-0.5688028626	-0.8	Reference x	-0.5688028626
y	0.2	-0.002292065054	-0.2	Reference y	-0.002292065054
z	0.85	0.8423955782	0.35	Reference z	0.8423955782
Human presence	True	True		Text output:	”stop”

Table 6.3: Rule breach in yellow reaction in green, breach of being inside a defined volume resulting in the reaction stop which sets the reference position to the current position. x, y, z are the Cartesian coordinates for both the limits and the measured end-effector position.

timestamp 2.4	rule	measured value		reaction
max velocity	0.1	0.1196450425	decelerate	TRUE
Human presence	FALSE	FALSE	Text output:	”decelerate”
timestamp 2.5	rule	measured value		reaction
force	0.1	1	Turn off active force	TRUE
Human presence	FALSE	FALSE	Text output:	”Turn off active force”

Table 6.4: Rule breach in yellow reaction in green, shows both the “decelerate” reaction as well as the “Turn of active force” reaction sending their respective booleans. See also Figure 6.7.

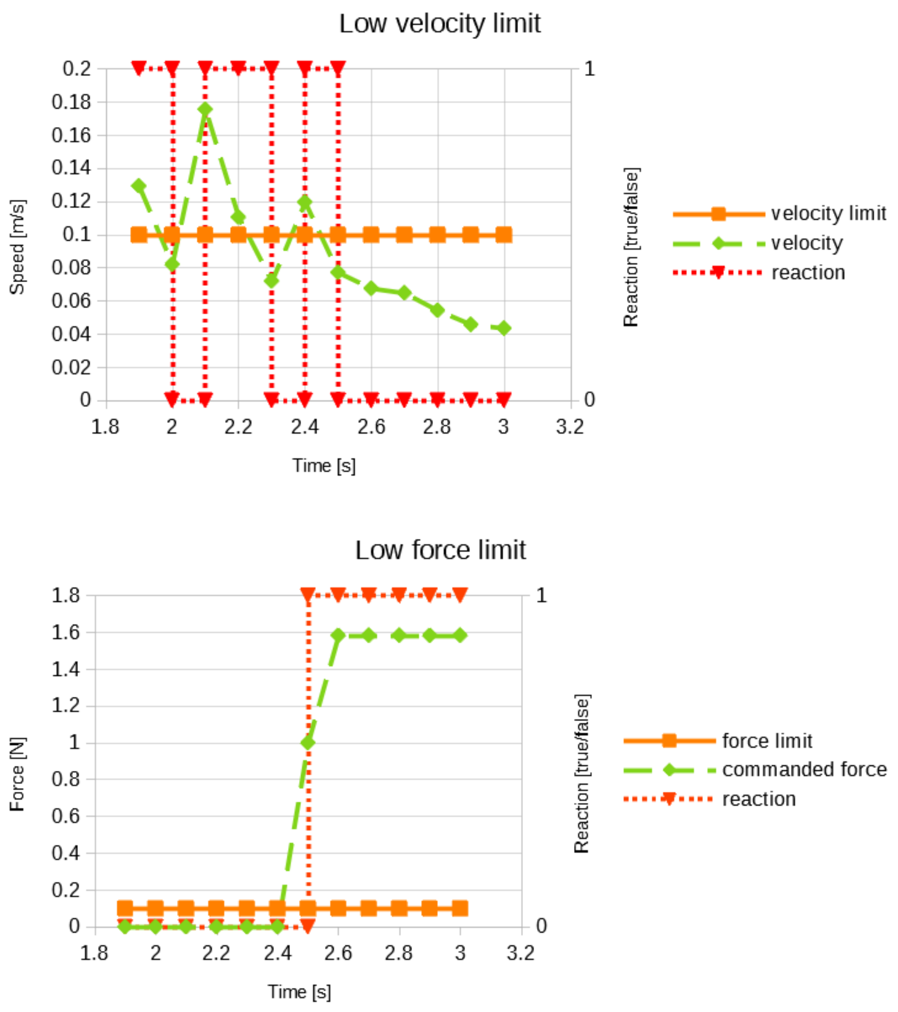


Figure 6.7: The diagram shows the recorded data around the event in Table 6.4. The solid orange line is the limit, the spaced green line is the measured value, and the red dotted line is the reaction triggers value, true or false, with its separate axis to the right.

		rule breach inside designated volume			
timestamp 0.7	rule max	measured value	rule min		reaction
x	-0.25	-0.5502321441	-0.79	Reference x	-0.5504980574
y	0.53	0.5298671124	0.07	Reference y	0.5375262483
z	0.981	0.7209949864	0.6	Reference z	0.7329785352
Human presence	True	True		Text output:	”return to origin”

Table 6.5: Rule breach in yellow reaction in green, is of the *avoid-obstacle* scenario. It is shown to react appropriately by sending a previous position in green. x, y, z are the Cartesian coordinates for both the limits and the measured end-effector position.

7 Discussion and Conclusion

The simulation results look very promising and all the reactions to the defined rules work as intended. The speed of the stop reaction and the speed of the return reaction is within acceptable limits. From the scenario of of Figure 6.3, stopping with the cartesian impedance controller from a speed of 0.15 m/s to a complete stand still, of the about 20 kg arm, took 0.3 seconds without activating any of the robot arm's brakes. In the scenario, same as Table 6.1, the decelerated reaction was decelerating from 0.97 m/s to 0.56 m/s in 0.2 seconds. The way the braking works can be a bit criticized due to the fact that the *deceleration* reaction only continues the current inertial trajectory without taking into account the desired trajectory and braking along that path. The *deceleration* could also be optimized if it would run in *memory mode* (when not training it is posible to remember previous states). Then it would decelerate in relation to one point until the speed is decreased and then let the controller take over. Instead of always updating the point to which the deceleration occurs.

A few of the recovery behaviours like the behaviour of only moving back a few steps in time when a volume breach occurs, or moving perpendicular to the breach, as well as the invisible wall behaviour which would only break in the breach direction, only allowing movements in the tangential direction, as well the perpendicular direction away from the breach, was not possible to be implemented. This was due to the time restriction for the thesis.

The field of safety in human-robot interactions is quite a broad one and finding clear and straightforward rules and guidelines to follow when setting up the safety was not as easy as first thought. This was due to the fact that there are many ways of doing things and different people have made different solutions to very similar problems. The safety method used in this thesis is a combination of many different safety solutions where the most sensible methods for the task at hand were used.

When formulating the safety studies as mentioned, the literature studies were the basis for the different states and their respective safety level. Literature studies was also important for formulation scenarios likely to occur in the robot arm setup selected for this thesis. As a helpful tool, the different scenarios were drawn up to help find all the different states in this entire simplified state space. The conclusion was that the exact situation for what are the safety parameters that will ensure safety for this specific scenario, like objects and tools and their characteristics, are to be worked out separately and not to have general rules for each situation. However some simplified states like human contact, object contact, inside a volume, and having some rough categories of rules with variable values for what is safe was seen as a sensible solution. For example, defining volumes and some state characteristics from the robot that can be given a custom maximum and minimum value.

7.1 Future Work

Some future work that can be done is adding the mentioned more sophisticated recovery behaviors currently not added. In addition to this, improving some of the already working reactions and adding some more complexity to their functions to ensure a more efficient reaction time is worth considering. The stateless mode could also be completely scratched if the safety program could be run in memory mode with the parallel threads, creating multiple running instances of the safety program connected to their respective instance.

Another thing is to add a repetition safety switch that triggers if the same safety rules are broken many times in a row which would indicate that the robot has gotten stuck and needs human attention. When writing rules in the JSON file, it is very important that all texts describing the rules, the reactions, and the limits contain no errors. This could be easily improved with a Json schema file, indicating spell errors or commands that do not exist. Another solution worth considering would be to have a graphical user interface where the different volumes and there respective rules would be seen in the 3D robot environment. This graphical user interface would then create the JSON files automatically from the scenario created. Also, a graphical user interface for the breaking of rules could also be a good idea in an industrial setting with an operator working on the safety program.

The implementation of a reward system would be of great value, as SkiREIL is a reinforcement learning platform. And tying rules and breaches of them together with some reward value would be favorable. For example, if the speed limit is broken deduct 2 points from the reward value or if a certain volume is breached deduct 4 points all depending on the desired outcome. This was also not implemented due to time constraints but it should not be too complicated as this can be done in a very similar way to the already existing architecture.

Adding more inputs for the safety system would also be of value. As one of the simulation scenarios shows, that had an arm segments which were breaching the limits due to their position being unknown for the safety system. Extra inputs could also help identify contacts with force curves for the transient contact instance, as well as helping in keeping the kinetic energy of the entire arm, including arm segments under safe limits.

Bibliography

[1] Lihui Wang, Xi Vincent Wang, József Váncza and Zsolt Kemény. *Advanced Human–Robot Collaboration in Manufacturing.* Springer, 2021. Chap. 3, pp. 74–84.

[2] Robin Jeanne Kirschner, Nico Mansfeld, Saeed Abdolshah and Sami Haddadin. "ISO/TS 15066: How Different Interpretations Affect Risk Assessment". In: (2022), pp. 7–8.

[3] Sorin Adam, Morten Larsen, Kjeld Jensen and Ulrik Pagh Schultz. "Rule-based Dynamic Safety Monitoring for Mobile Robots". In: *Journal of Software Engineering for Robotics* (2016).

[4] Paolo Barattini, Federico Vicentini, Gurvinder Singh Virk and Tamás Haidegger. *Human-Robot Interaction safety,standardization and benchmarking.* Taylor Francis Group, 2019.

[5] Milos Vasic and Aude Billard. "Safety issues in human-robot interactions". In: (2013), pp. 197–204.

[6] SIS Swedish standard institute. "Robots and robotic devices – Collaborative robots". In: (2016), p. 28.

[7] Erik Ingelstam och Stig Sjöberg. *ELFYMA TABELLER.* Sjöbergs förlag Stockholm, 1967.

[8] Lihui Wang, Xi Vincent Wang, József Váncza and Zsolt Kemény. *Advanced Human–Robot Collaboration in Manufacturing.* Springer, 2021. Chap. 3, p. 77.

[9] S. Robla-Gomez, V.M. Becerra, J.R. Llata, E. Gonzalez-Sarabia, C. Torre-Ferrero and J. Perez-Oria. "Working Together: A Review on Safe Human-Robot Collaboration in Industrial Environments". In: (2017).

[10] J.A. Marvel, J. Falco and I. Marstio. "Characterizing task-based human–robot collaboration safety in manufacturing". In: (2017), 260–275.

[11] G. Michalos, S. Makris, P. Tsarouchi, T. Guasch, D. Kontovrakis and G. Chryssolouris. "Design considerations for safe human–robot collaborative workplaces". In: (2015), 248–253.

[12] J.A. Corrales, G.J.G. Gomez, F. Torres and V. Perdereau. "Cooperative tasks between humans and robots in industrial environments". In: (2012).

[13] A. Mohammed, B. Schmidt and L. Wang. "Active collision avoidance for human–robot collaboration driven by vision sensors". In: (2017), 970–980.

[14] *What is ROS?* `https://roboticsbackend.com/what-is-ros/`. 2022.

[15] *ROS Tutorials.* `http://wiki.ros.org/ROS/Tutorials`. 2022.

[16] Francesco Rovida, Matthew Crosby, Dirk Holz, Athanasios S. Polydoros, Bjarne Großmann, Ronald P.A. Petrick and Volker Krüger. "SkiROS—A Skill-Based Robot Control Platform on Top of ROS". In: 2 (2013), pp. 121–160.

[17] Francesco Rovida. *What is SkiROS?* `https://github.com/RVMI/skiros2/wiki`. 2022.

[18] *SkiReil.* `https://github.com/matthias-mayr/SkiREIL`. 2022.

[19] Matthias Mayr, Momina Rizwan, David Sandell, Christoph Reichenbach and Volker Krueger. “Constraining and Guiding Reinforcement Learning With Rule-Based Safety Monitoring”. In: *preprint* (2022).

[20] Matthias Mayr and Julian M. Salt-Ducaju. “A C++ Implementation of a Cartesian Impedance Controller for Robotic Manipulators”. In: (2022).

Appendix A

Appendix programming part

A.1 David's Safety Program

```
#!/usr/bin/env python
# encoding: utf-8

#from turtle import delay
from formatter import NullFormatter
from lib2to3.refactor import RefactoringTool
#from lzma import CHECK_ID_MAX
#from multiprocessing.resource_sharer import stop
#from xmlrpc.client import Boolean
import copy
import rospy
from std_msgs.msg import String
from std_msgs.msg import Bool
from geometry_msgs.msg import Pose
from geometry_msgs.msg import PoseStamped
from geometry_msgs.msg import Wrench
from geometry_msgs.msg import WrenchStamped
from geometry_msgs.msg import Vector3Stamped
#from skireil import to_RosMari_msg
from skireil.msg import RosMariHumanPresent
from skireil.msg import RosMariReactionmsg
from skireil.srv import RosMari,RosMariRequest, RosMariResponse
from skireil.srv import RosMariLoad,RosMariLoadRequest, RosMariLoadResponse
from skireil.srv import RosMariRealSafety,RosMariRealSafetyRequest,
   RosMariRealSafetyResponse
import numpy as np
import threading
import time
import json
import os

class read_and_set_limits:
   debugging = False
   list_of_limits = []
   read_from_limits = []

   #debugging variable that controlls extra prints
   def set_debugging(self,debugging):
      self.debugging = debugging
```

```
    #loads in the json string with the list of rules/limits
    def load_and_set_rules(self, json_string):
        data=json.loads(json_string)
        self.list_of_limits = data
        if self.debugging:
            print(self.list_of_limits)

    #this can be good if you do not realy whant to load a json file and
        whant to chek that a newly created rule just works.
    def set_debugg_values(self):
        self.list_of_limits=self.read_from_limits

    #this returns the list of the rules/limits
    def get_list_of_limits(self):
        return self.list_of_limits

# contains funktions that cheks for safety breaches and returns a list
    relevant reactions
class check_safety_rules():
    #cheks if end efector is oustside a defined 3d_volume/2d_area/1d line
    def check_if_outside(self,limits,robot_state):
        outside=False
        name = "ee_position"
        attrs1 = ["_min","_max"]
        attrs2 = ["x", "y", "z"]
        if "Box_info" in limits:
            for attr1 in attrs1:
                for attr2 in attrs2:
                    if (attr2+attr1) in limits:
                        if(attr1=="_min"):
                            outside = outside or robot_state[name+"."+attr2]<
                                limits[attr2+attr1]
                        if(attr1=="_max"):
                            outside = outside or robot_state[name+"."+attr2]>
                                limits[attr2+attr1]

        return outside

    #returns true on gemoetric breach or absens of current rule
    def check_geometric_breach(self, limits, robot_state):
        Geometric_rule_breach=True
        outside = self.check_if_outside(limits, robot_state)
        if "Box_info" in limits:
            if limits["Box_info"] == "position outside 3D box volume":
                Geometric_rule_breach = outside
            if limits["Box_info"] =="position inside 3D box volume":
                Geometric_rule_breach = not outside

        return Geometric_rule_breach
```

```
#returns true on velocity breach or absens of current rule
def check_velocity_breach(self, limits, robot_state):
    velocity_TrueFalse = True
    if "velocity" in limits:
        velocity_TrueFalse = velocity_TrueFalse and robot_state["
            ee_velocity"]>limits["velocity"]
    return velocity_TrueFalse

#returns true on force breach or absens of current rule
def check_force_breach(self, limits, robot_state):
    force_TrueFalse = True
    if "force" in limits:
        force_TrueFalse = force_TrueFalse and robot_state["ee_force"]>
            limits["force"]
    return force_TrueFalse

#returns true if rule human precense bolean is same as the steate human
    precens or absens of current rule
def check_human(self,limits, robot_state, human_precent):
    human_TrueFalse =True
    if "Human presence" in limits:
        human_TrueFalse = human_TrueFalse and limits["Human presence"]==
            human_precent

    return human_TrueFalse

#sums upp if all breaches in rule block is breached and then return true
def check_safety_breach(self, limits, robot_state, human_precent):
    breach=True
    breach = breach and self.check_geometric_breach(limits, robot_state)
    breach = breach and self.check_velocity_breach(limits, robot_state)
    breach = breach and self.check_force_breach(limits, robot_state)
    breach = breach and self.check_human(limits, robot_state,
       human_precent)
    return breach

#finds a list of the breached rules with the highest priority and then
    retruns a dictionary with all the active reactions
def check_all_safety_rules(self, list_of_limits, robot_state,
    human_precent):
    priority = float('inf')

    reactions=[]

    for limits in list_of_limits:
        if self.check_safety_breach(limits, robot_state, human_precent):
            if limits["priority"]<=priority:
                priority = limits["priority"]
                reactions.append(limits)

    reactions_list = {}
```

```
        for R in reversed(reactions):
            if R["priority"]==priority:

                #a list of code of what to do to reactions_list if a certain
                    reaction is detected if it should be added or not in
                    sertain situtations
                # moastly it is exlusivly just add. in this case stop
                    reaction is treated differently.
                if "stop" == R["reaction"]:
                    reactions_list.pop("decelerate",None)
                    reactions_list.pop("Turn off active force",None)
                    reactions_list.pop("return to origin",None)
                    reactions_list["stop"]=R

                if "decelerate" == R["reaction"] and not ("stop" in
                    reactions_list):
                    reactions_list["decelerate"] = R
                    #print("decelerate")

                if "Turn off active force" == R["reaction"] and not ("stop"
                    in reactions_list):
                    reactions_list["Turn off active force"] = R
                    #print("Turn off active force")

                if "return to origin" == R["reaction"] and not ("stop" in
                    reactions_list):
                    reactions_list["return to origin"] = R

        return reactions_list

#has functions for adding the incomming service message to "robot_state" and
     also save them in a record for the return to origin function. also
    contains said rection and a few others on how to respond to the sender.
    and finaly sends back the service message.
class service_message_and_reaction():

    #initing al variables used
    def __init__(self):
        self.read_and_set = read_and_set_limits()
        self.check_safety = check_safety_rules()

        self.variable_i = 0

        self.robot_state = {}
        self.states_list=[]
        self.in_recovery_mode=False
        self.recovery_Limits={}
        self.recovery_counter=0

        self.stopbool = False
        self.stopPose= Pose()
        self.stopPose.position.x=0.0
        self.stopPose.position.y=0.0
        self.stopPose.position.z=0.0
        self.stopPose.orientation.x=0.0
        self.stopPose.orientation.y=0.0
        self.stopPose.orientation.z=0.0
        self.stopPose.orientation.w=0.0
        self.stopClock = 0.0
        self.stopLapTime = 2.0
        self.in_learning = False
        self.debugging = False

    #sets the in learning varialbe. in learning controlls if the program
        should be statles or not if in learning it is stateles. this is for
        the simplification of threading in the sender side.
    def set_in_learning(self, in_learning):
        self.in_learning = in_learning

    #debugging var for more prints
    def set_debugging(self, debugging):
        self.debugging=debugging # gives more active prints

    #stateless stop functions just sets the ref pose to the current pose and
         stiffnes to 200 and 20 values agreed upon
    def simple_stop(self,robot_state,ref_pose):
        self.set_coordinates(robot_state,"ee_position",ref_pose.position)
        self.set_coordinates(robot_state,"ee_orientation",ref_pose.
            orientation)
        wrench = Wrench()
        self.set_coordinage_value(0, wrench.force)
        self.set_coordinage_value(0, wrench.torque)

        stiffness = Wrench()
        self.set_coordinage_value(200, stiffness.force)
        self.set_coordinage_value(20, stiffness.torque)
        return ref_pose,wrench,stiffness

    #stopfunction with states sets to the first broken value position and
        orientation
    def stop_function(self):
        self.set_coordinates(self.robot_state, "ee_position", self.stopPose.
            position)

        self.set_coordinates(self.robot_state, "ee_orientation", self.
            stopPose.orientation,get_w=True)

        #do not set the values acording to to the edge of the defined volumes
        #the complexity of alowed and disalowed volumes make it to much of a
            hassle at this time
        #just think about it, if you have a tube of a allowed volume undoing
        an outside disalowed volume if
    #you hit the edge of this tube far away from the edge of the outside
        volume it will putt an position far down the tube
    #at its entrance isntead of close to the breach therfore a faster
        sampling might be needed in such a case
    #to ensure the closnes of the stop position to the breach point
    #also for safety adding a value for the time it spends or sample
        periodes it takes before it
    # resets so you can safely push it away posibly.

# sets a all coordinates to the same value
def set_coordinage_value(self, value, src, get_w=False):
    attrs = ["x", "y", "z"]
    if get_w:
        attrs.append("w")

    for attr in attrs:
        setattr(src, attr, value)

#stopfunction with states returnts the first broken value position and
    orientation as well as stiffnes 200 and 20 values agreed upon
def return_stop(self):
    wrench = Wrench()
    self.set_coordinage_value(0, wrench.force)
    self.set_coordinage_value(0, wrench.torque)

    stiffness = Wrench()
    self.set_coordinage_value(200, stiffness.force)
    self.set_coordinage_value(20, stiffness.torque)
    return self.stopPose,wrench,stiffness

#subracts two coordinate dictionaries to eachother
def subtract_dict_coordinates(self, dict1, dict2, name, get_w=False):
    attrs=["x", "y", "z"]
    if get_w:
        attrs.append("w")
    vector=[]
    for attr in attrs:
        vector.append(dict1[name + "." + attr]-dict2[name + "." + attr])

    return vector
#calculates the euclidian distance betwene spatial points
def euclidean_distance(self, delta_vector):
    return np.sqrt(np.sum(np.square(delta_vector)))

#returns to the origin position acording to te recorded values. when it
    is close enough acording to the preset tolerance value it will stop
    auto entering this function
def return_to_origin(self):
    referencePose= Pose()
    delta_position_vector=self.subtract_dict_coordinates(self.states_list
            [self.recovery_counter], self.robot_state, "ee_position")

        delta_orientation_vector=self.subtract_dict_coordinates(self.
            states_list[self.recovery_counter], self.robot_state, "
            ee_orientation",get_w=True)

        #Distance to origin
        delta_origin_position_vector=self.subtract_dict_coordinates(self.
            states_list[0], self.robot_state, "ee_position")

        delta_origin_orientation_vector=self.subtract_dict_coordinates(self.
            states_list[0], self.robot_state, "ee_orientation",get_w=True)

        delta_position=self.euclidean_distance(delta_position_vector)
        delta_origin_position=self.euclidean_distance(
            delta_origin_position_vector)

        if delta_origin_position <=self.recovery_Limits["origin tolerance"]:
            self.in_recovery_mode=False

        elif delta_position <=self.recovery_Limits["origin tolerance"]:
            self.recovery_counter = self.recovery_counter-1

        self.set_coordinates(self.states_list[self.recovery_counter], "
            ee_position", referencePose.position)
        self.set_coordinates(self.states_list[self.recovery_counter], "
            ee_orientation", referencePose.orientation,get_w=True)

        wrench = Wrench()
        self.set_coordinage_value(0, wrench.force)
        self.set_coordinage_value(0, wrench.torque)

        stiffness = Wrench()
        self.set_coordinage_value(200, stiffness.force)
        self.set_coordinage_value(20, stiffness.torque)

        return referencePose, wrench, stiffness

    #here all reactions prepared to be sent as a service message
    def publish_reactions(self, reaction_list):

        if self.robot_state["time"]<=0.1:
            self.stopbool = False

        wrench = Wrench()
        self.set_coordinates(self.robot_state, "ee_force", wrench.force)
        self.set_coordinates(self.robot_state, "ee_torque", wrench.torque)

        stiffness = Wrench()
        self.set_coordinates(self.robot_state, "ee_springforce", stiffness.
            force)
```

```
self.set_coordinates(self.robot_state, "ee_springtorque", stiffness.
    torque)

violated_rule_info=""
decelerate=False
turn_off_active_force = False
turn_off_stiffness=False
violated_rule_info += self.robot_state["header timestamp"]

referencePose = Pose()
self.set_coordinates(self.robot_state, "ref_ee_position",
    referencePose.position)

self.set_coordinates(self.robot_state, "ref_ee_orientation",
    referencePose.orientation,get_w=True)

if self.debugging:
    print("x: "+str(self.robot_state["ee_position.x"])+"\n"+"y: "+str
        (self.robot_state["ee_position.y"])+"\n"+"z: "+str(self.
        robot_state["ee_position.z"])+"\n")

#the reaction is decided here
for reaction in reaction_list:

    violated_rule_info += "limit category: " + (reaction_list[
        reaction]["limit category"]) + "\t"+" id: "+ str(
        reaction_list[reaction]["combine id"]) + "\t"+" reaction: "+
        (reaction_list[reaction]["reaction"]) +"\t"+" Human presenc:
        "+str(reaction_list[reaction]["Human presence"])+"\n"

    decelerate = decelerate or "decelerate" == reaction

    turn_off_active_force = turn_off_active_force or "Turn off active
         force" == reaction

    turn_off_stiffness = turn_off_stiffness or turn_off_stiffness ==
        "turn_off_stiffness"# add this to the reactions

    if not self.in_learning:
        if(not self.stopbool and "stop"==reaction):
            self.stopbool=True
            self.stopClock=self.robot_state["time"]
            self.stop_function()
            #sets up the stop sets the flag for a reached stop to true
                 so the robot stays in thet state
            #then the clock funktion clock is set this will ensure
                that the arms referances position can be manualy
                changed
            #then sets the position at wich the arm is to be locked at

        if(reaction=="return to origin" and not self.in_recovery_mode
                ):
                self.in_recovery_mode=True
                self.recovery_counter=len(self.states_list)-1
                self.recovery_Limits=reaction_list[reaction]

        elif reaction=="stop" or reaction=="return to origin": # if in
            learning no states are saved so rection must be more simple
            referencePose ,wrench, stiffness = self.simple_stop(self.
                robot_state,referencePose)

    #if not in learning it will performe recorded stop and recovery
    if not self.in_learning:
        #clock funktion that tics every laptime seconds and updates the
            stop positon
        if(self.stopPose and self.robot_state["time"]-self.stopClock>=
            self.stopLapTime):
                self.stopClock=self.robot_state["time"]
                self.stop_function()

        #checks if recovery is True and then proceds to override pose
            values and stiffens values untill it is no longer true wich
            is when it hais retrunt to the origin point
        if(self.in_recovery_mode and not decelerate and not self.stopbool
            ):
            referencePose,wrench,stiffness = self.return_to_origin()

        #also checs if stopbool ist true then overrides the pose values
            and stiffens values untill it is no longer true wich is at
            the end of the sesion when time is less then 0.1s
        if(self.stopbool):
            referencePose,wrench,stiffness = self.return_stop()
    #end of reaction
    if self.debugging:
        rospy.loginfo(violated_rule_info)
        rospy.loginfo(decelerate)
        rospy.loginfo(turn_off_active_force)
        rospy.loginfo(turn_off_stiffness)
        #rospy.loginfo(pubreaction)
        rospy.loginfo(referencePose)

    #here is returned all the values needed for the service messge
    return violated_rule_info,decelerate,turn_off_active_force,
        turn_off_stiffness,referencePose,wrench,stiffness

#loads all coordinate values from the incomming service message to the "
    robot_state" dictionary
def load_from_coordinates(self, dst,dst_name,src,get_w = False):
    attrs = ["x", "y", "z"]
    if get_w:
        attrs.append("w")
    for attr in attrs:
        dst[dst_name + "." + attr] = getattr(src, attr)

#here is done the reverse of "load_from_coordinates()" here it load from
     the "robot_state" dictionary to the service message responce
def set_coordinates(self, dst,dst_name,src,get_w = False):
    attrs = ["x", "y", "z"]
    if get_w:
        attrs.append("w")

    for attr in attrs:
        setattr(src, attr, dst[dst_name + "." + attr])

#deals with all the service messages forward and back again
def service_callback(self, req):
    refpose = Pose()
    wrench = Wrench()
    stiffness = Wrench()

    self.load_from_coordinates(self.robot_state, "ee_position", req.pose.
        position)
    self.load_from_coordinates(self.robot_state, "ee_orientation", req.
        pose.orientation, get_w = True)
    self.load_from_coordinates(self.robot_state, "ref_ee_position", req.
        refpose.position)
    self.load_from_coordinates(self.robot_state, "ref_ee_orientation",
        req.refpose.orientation, get_w = True)
    self.load_from_coordinates(self.robot_state, "ee_force", req.wrench.
        force)
    self.load_from_coordinates(self.robot_state, "ee_torque", req.wrench.
        torque)
    self.load_from_coordinates(self.robot_state, "ee_springforce", req.
        stiffness.force)
    self.load_from_coordinates(self.robot_state, "ee_springtorque", req.
        stiffness.torque)

    self.robot_state["ee_force"] = self.euclidean_distance([req.wrench.
        force.x, req.wrench.force.y, req.wrench.force.z])
    if self.debugging:
        print("ee_force")
        print(self.robot_state["ee_force"])

    self.load_from_coordinates(self.robot_state, "ee_velocity", req.
        velocity)
    self.robot_state["ee_velocity"] = self.euclidean_distance([req.
        velocity.x, req.velocity.y, req.velocity.z])
    if self.debugging:
        print("ee_velocity")
        print(self.robot_state["ee_velocity"])

    self.robot_state["human_pose"]=req.human
```

```
        if self.debugging:
            self.robot_state["human_pose"]= False # debugging
        self.robot_state["header timestamp"] = str(req.header.stamp.secs) +
            ":" + str(req.header.stamp.nsecs) +":"
        self.robot_state["time"] = req.header.stamp.secs + req.header.stamp.
            nsecs*10**-9


        #add your loading before this comment
        #when in learning, safety run in stateless mode.
        if not self.in_learning:
            temp_state=copy.deepcopy(self.robot_state)
            self.states_list.append(temp_state)

        violated_rule, decelerate, turn_off_active_force, turn_off_stiffness,
             refpose, wrench, stiffness=self.publish_reactions(self.
            check_safety.check_all_safety_rules(self.read_and_set.
            get_list_of_limits(), self.robot_state,self.robot_state["
            human_pose"]))
        return RosMariRealSafetyResponse(violated_rule, decelerate,
            turn_off_active_force, turn_off_stiffness, refpose, wrench,
            stiffness)

    def load(self, pathreq):
        self.read_and_set.set_debugging(self.debugging)
        self.read_and_set.load_and_set_rules(pathreq.path)
        #self.read_and_set.set_debugg_values()#if testing of custom rules and
             reactions is desired this can be helpfull
        acknolidgment="list arrived"
        return RosMariLoadResponse(acknolidgment)

def listener(smar):

    debugging = False
    rospy.init_node('skireil', anonymous=True)
    smar.set_debugging(debugging)
    if debugging:
        print("in service")
    s = rospy.Service("RosMariRealSafety_server",RosMariRealSafety,smar.
        service_callback)
    s2 = rospy.Service("RosMari_load",RosMariLoad,smar.load)
    in_learning = rospy.get_param("/RosMariRealSafety/learning",False)

    if debugging:
        in_learning=False # debugging
        print("in_learning might be set as a set value due to it being
            debuging")

    print("python RosMariRealSafety outpuf from rospy.get_param:
        in_learnning = "+str(in_learning))
    smar.set_in_learning(in_learning)
    rospy.spin()
    print("after rosspin")

if __name__ == '__main__':
    variable_i=0
    #read_and_set.put_limits_together()
    smar=service_message_and_reaction()
    smar.__init__()
    print("in main before listner()")
    listener(smar)
    print("in main after listner()")
```

A.2 David's Safety Interface

```
//|
//|   Copyright Inria July 2017
//|   This project has received funding from the European Research Council
    (ERC) under
//|   the European Union's Horizon 2020 research and innovation programme (
    grant
//|   agreement No 637972) - see http://www.resibots.eu
//|
//|   Contributor(s):
//|     - Matthias Mayr (matthias.mayr@cs.lth.se)
//|     - Konstantinos Chatzilygeroudis (konstantinos.chatzilygeroudis@inria
    .fr)
//|     - Rituraj Kaushik (rituraj.kaushik@inria.fr)
//|     - Roberto Rama (bertoski@gmail.com)
//|
//|
//|   This software is governed by the CeCILL-C license under French law
    and
//|   abiding by the rules of distribution of free software. You can use,
//|   modify and/ or redistribute the software under the terms of the
    CeCILL-C
//|   license as circulated by CEA, CNRS and INRIA at the following URL
//|   "http://www.cecill.info".
//|
//|   As a counterpart to the access to the source code and rights to copy,
//|   modify and redistribute granted by the license, users are provided
    only
//|   with a limited warranty and the software's author, the holder of the
//|   economic rights, and the successive licensors have only limited
//|   liability.
//|
//|   In this respect, the user's attention is drawn to the risks
    associated
//|   with loading, using, modifying and/or developing or reproducing the
//|   software by the user in light of its specific status of free software
   ,
//|   that may mean that it is complicated to manipulate, and that also
//|   therefore means that it is reserved for developers and experienced
//|   professionals having in-depth computer knowledge. Users are therefore
//|   encouraged to load and test the software's suitability as regards
   their
//|   requirements in conditions enabling the security of their systems and
   /or
//|   data to be ensured and, more generally, to use and operate it in the
//|   same conditions as regards security.
//|
//|   The fact that you are presently reading this means that you have had
//|   knowledge of the CeCILL-C license and that you accept its terms.
//|
#ifndef SKIREIL_SAFETY_INTERFACE_DAVIDS_INTERFACE_HPP
#define SKIREIL_SAFETY_INTERFACE_DAVIDS_INTERFACE_HPP

#include <skireil/safety/safety_interface.hpp>

#include <skireil/system/dart_system.hpp>
#include <skireil/parameters.hpp>
#include <cartesian_impedance_controller/cartesian_impedance_controller.h>
#include <skireil/controller/discrete_motion_generator.hpp>
#include <skireil/controller/motion_generator_configuration.hpp>
#include <skireil/experiments/cartesian_impedance_action.hpp>

#include <robot_dart/robot.hpp>
#include "geometry_msgs/Point.h"
#include <eigen_conversions/eigen_msg.h>
#include <geometry_msgs/Pose.h>
#include <geometry_msgs/Vector3.h>
#include <geometry_msgs/Vector3Stamped.h>
#include <Eigen/Core>
#include <Eigen/Geometry>
#include <ros/ros.h>
#include "std_msgs/String.h"
#include "std_msgs/Bool.h"
#include <skireil/RosMariHumanPresent.h>
#include <skireil/To_RosMari_msg.h>
#include <skireil/RosMariReactionmsg.h>
#include <skireil/RosMari.h>
#include <skireil/RosMariLoad.h>
#include <skireil/RosMariRealSafety.h>
#include <skireil/parameters.hpp>
//#include <skireil/safety/davids_safety_interface.hpp>

#include <string>
#include <vector>
#include <sstream>
```

```
using namespace std;

namespace skireil
{
   namespace safety
   {
      template <typename RolloutInfo, typename Params>
      class davids_safety_interface : public SafetyInterface<RolloutInfo,
         Params>
      {
         private:

            ros::NodeHandle _n;
            ros::ServiceClient _client2 = _n.serviceClient<skireil::
               RosMariLoad>("RosMari_load");
            ros::ServiceClient _client3 = _n.serviceClient<skireil::
               RosMariRealSafety>("RosMariRealSafety_server");

         public:
            bool configure(const nlohmann::json &conf, const std::vector<
               std::string> &robot_dof){
               skireil::RosMariLoad RML;
               std::stringstream ss;
               ss << conf;
               string data = ss.str();
               RML.request.path = data;
               if (_client2.call(RML)){
                  ROS_INFO("acknowledgement: %ld", (string)RML.response.
                     acknowledge);
                  return true;
               }
               else
               {
                  ROS_ERROR("Failed to call service RosMari");
                  return false;
               }
            }
            //returns a value that is rounded to nr amount of decimals
            double round_double(double nr,int decimals){
               int decimal=std::pow(10,decimals);
               double value = (long)(nr * decimal+0.5);
               return (double)value/decimal;
            }
            //rounds off to values to the caem nr of decimal pints before
                comparing them. and returns true if they are not equal
            bool not_equal(double var1,double var2, int decimals){
               return round_double(var1, decimals) != round_double(var2,
                  decimals);
            }

            bool operator()(Eigen::VectorXd &state, Eigen::VectorXd &
    action, double time)
{
    skireil::RosMariRealSafety RMRS;
    auto robot = this->_robot;
    if (!robot){
        LOG(FATAL)<<"empty pointer _robot
            davids_safety_interface.hpp";
    }
    //recives the different states and stores them in more
        palatable variables
    robot->set_positions(state.head(7), Params::skireil::
        robot_dof());
    robot->set_velocities(state.segment(7, 7), Params::skireil
        ::robot_dof());
    const Eigen::Matrix<double, 6, 1> deltaPos = robot->
        jacobian(Params::skireil::robot_end_effector(), Params
        ::skireil::robot_dof()) * robot->velocities(Params::
        skireil::robot_dof());

    std::pair<Eigen::Vector3d, Eigen::Quaterniond> ref_pose;
    Eigen::Matrix<double, 6, 1> var_wrench;
    Eigen::Matrix<double, 7, 1> var_stiffness;

    learning_skills::cartesian_impedance_action::
        from_eigen_vector(action, &ref_pose, &var_wrench, &
        var_stiffness);

    RMRS.request.header.stamp.sec=time;
    RMRS.request.header.stamp.nsec= 1000000000*(time-RMRS.
        request.header.stamp.sec);
    tf::poseEigenToMsg(robot->skeleton()->getBodyNode(Params::
        skireil::robot_end_effector())->getWorldTransform(),
        RMRS.request.pose);
    RMRS.request.velocity.x = deltaPos(3);
    RMRS.request.velocity.y = deltaPos(4);
    RMRS.request.velocity.z = deltaPos(5);
    RMRS.request.wrench.force.x = var_wrench(0);
    RMRS.request.wrench.force.y = var_wrench(1);
    RMRS.request.wrench.force.z = var_wrench(2);
    RMRS.request.wrench.torque.x = var_wrench(3);
    RMRS.request.wrench.torque.y = var_wrench(4);
    RMRS.request.wrench.torque.z = var_wrench(5);
    RMRS.request.refpose.position.x=ref_pose.first.x();
    RMRS.request.refpose.position.y=ref_pose.first.y();
    RMRS.request.refpose.position.z=ref_pose.first.z();
    RMRS.request.refpose.orientation.x=ref_pose.second.x();
    RMRS.request.refpose.orientation.y=ref_pose.second.y();
    RMRS.request.refpose.orientation.z=ref_pose.second.z();
    RMRS.request.refpose.orientation.w=ref_pose.second.w();
    RMRS.request.stiffness.force.x = var_stiffness(0);
    RMRS.request.stiffness.force.y = var_stiffness(1);
RMRS.request.stiffness.force.z = var_stiffness(2);
RMRS.request.stiffness.torque.x = var_stiffness(3);
RMRS.request.stiffness.torque.y = var_stiffness(4);
RMRS.request.stiffness.torque.z = var_stiffness(5);
RMRS.request.human = false;

int i = 0;

bool rule_broken = false;
bool decelerate = false;
int decimals = 5;
bool success = false;
// runs the state and action trhough the safety program
while(!success && i < 3){
    i++;
    //cheks that the call can be recived
    if (_client3.call(RMRS))
    {
        success=true;
        //setting action values if reciven a bool back for
            reaction decelerate
        if(RMRS.response.decelerate){
            std::cout << "violated_rule: " << RMRS.response
                .violated_rule << std::endl;
            ref_pose.first = robot->skeleton()->getBodyNode
                 (Params::skireil::robot_end_effector())->
                getWorldTransform().matrix().block(0, 3, 3,
                 1);
            ref_pose.second = Eigen::Matrix3d(robot->
                skeleton()->getBodyNode (Params::skireil::
                robot_end_effector())->getWorldTransform().
                matrix().block(0, 0, 3, 3));
            var_stiffness[0] = 200;
            var_stiffness[1] = 200;
            var_stiffness[2] = 200;
            var_stiffness[3] = 20;
            var_stiffness[4] = 20;
            var_stiffness[5] = 20;
            var_stiffness[6] = 0;
            rule_broken = true;
            decelerate=true;
        }
        //setting action values if reciven a bool back for
            reaction turn of active force
        if(RMRS.response.turn_off_active_force){
            std::cout << "violated_rule: " << RMRS.response
                .violated_rule << std::endl;
            var_wrench(0) = 0;
            var_wrench(1) = 0;
            var_wrench(2) = 0;
            var_wrench(3) = 0;
    var_wrench(4) = 0;
    var_wrench(5) = 0;
    rule_broken = true;
}
//setting action values if reciven a bool back for
    reaction turn of stiffness
if(RMRS.response.turn_off_stiffness){
    std::cout << "violated_rule: " << RMRS.response
        .violated_rule << std::endl;
    var_stiffness[0] = 0;
    var_stiffness[1] = 0;
    var_stiffness[2] = 0;
    var_stiffness[3] = 0;
    var_stiffness[4] = 0;
    var_stiffness[5] = 0;
    var_stiffness[6] = 0;
    rule_broken = true;
}

//setting action values directly as a respons form
    the RosMariRealSafety's values
if(!decelerate){
    if(not_equal(ref_pose.first.x(), RMRS.response.
        refpose.position.x, decimals) ||
    not_equal(ref_pose.first.y(), RMRS.response.
        refpose.position.y, decimals)||
    not_equal(ref_pose.first.z(), RMRS.response.
        refpose.position.z, decimals)){
        std::cout << "violated_rule: " << RMRS.
            response.violated_rule << std::endl;
        rule_broken = true;
    }
    ref_pose.first.x() = RMRS.response.refpose.
        position.x;
    ref_pose.first.y() = RMRS.response.refpose.
        position.y;
    ref_pose.first.z() = RMRS.response.refpose.
        position.z;
    ref_pose.second.x() = RMRS.response.refpose.
        orientation.x;
    ref_pose.second.y() = RMRS.response.refpose.
        orientation.y;
    ref_pose.second.z() = RMRS.response.refpose.
        orientation.z;
    ref_pose.second.w() = RMRS.response.refpose.
        orientation.w;
    var_stiffness(0) = RMRS.response.stiffness.
        force.x;
    var_stiffness(1) = RMRS.response.stiffness.
        force.y;
    var_stiffness(2) = RMRS.response.stiffness.
```

```
                    force.z;
                var_stiffness(3) = RMRS.response.stiffness.
                    torque.x;
                var_stiffness(4) = RMRS.response.stiffness.
                    torque.y;
                var_stiffness(5) = RMRS.response.stiffness.
                    torque.z;
                var_wrench(0) = RMRS.response.wrench.force.x;
                var_wrench(1) = RMRS.response.wrench.force.y;
                var_wrench(2) = RMRS.response.wrench.force.z;
                var_wrench(3) = RMRS.response.wrench.torque.x;
                var_wrench(4) = RMRS.response.wrench.torque.y;
                var_wrench(5) = RMRS.response.wrench.torque.z;
            }
            //always prints if a breach of rules are detected
            if(rule_broken){
                ROS_INFO("violated_rule: %ld", (string)RMRS.
                    response.violated_rule);
                LOG(WARNING) << "Detected a breach of rules,
                    violated rule: "<<RMRS.response.
                    violated_rule<<std::endl;
            }
        }
        //if the service message failed
        else
        {
            ROS_ERROR("Failed to call service RosMari");
            success = false;
        }
        }
        //if the service message failed it will frece the
            robot stopping it from moving as a safety feature.
            it will though be complient
        if(!success){
            ref_pose.first = robot->skeleton()->getBodyNode (
                Params::skireil::robot_end_effector())->
                getWorldTransform().matrix().block(0, 3, 3, 1);
            ref_pose.second = Eigen::Matrix3d(robot->skeleton()
                ->getBodyNode (Params::skireil::
                robot_end_effector())->getWorldTransform().
                matrix().block(0, 0, 3, 3));
        }
    //if the service message succeded it will translate the
        values back to a reaction and set it
    action=learning_skills::cartesian_impedance_action::
        to_eigen_vector(Params::safety::action_dim(), ref_pose
        , var_wrench, var_stiffness);
    return !rule_broken;
}

size_t state_safety_dim() const
```

```
                {
                    return 0;
                }

                size_t action_safety_dim() const
                {
                    return 0;
                }

                bool recover_from_incident(const SafetyIncident &incident,
                    const Eigen::VectorXd &state, Eigen::VectorXd &action)
                    const
                {
                    return true;
                }
        };
    }
}


#endif
```

DEPARTMENT OF COMPUTER SCIENCE | (LTH), LUND UNIVERSITY | PRESENTED 2023-04-27

MASTER'S THESIS Safe Learning and Execution of Robot Behaviors in Industrial Manipulation Tasks
STUDENT David Sandell
SUPERVISORS Matthias Mayr (LTH), Momina Rizwan (LTH)
EXAMINER Christoph Reichenbach (LTH)

Safe control of robot applications

POPULAR SCIENCE SUMMARY **David Sandell**

Safety and artificial intelligence are two important subjects to today, combining these are of high interest and significance. To combine these two was the task of this thesis with the focus lying on the safety. In robotics and industrial applications where the machines are heavy and powerful the potential for harm are also greater, therefore safety is of even more importance.

Therefore, such a safety program was created that can both protect sensitive equipment from danger and ensure safe execution when people are involved and may be injured. This was achieved through a series of different rules and responses, rules that covered areas that could not be safely reached by the robot, but also areas where higher safety was controlled and limited in the form of force and speed limits. The project was carried out with a robotic arm controlled and trained by an artificial intelligence.

The advantage of the safety system created, as mentioned earlier, is that it was applied to an artificial intelligence system, but also that unlike previous applications that often only included safety stops to maintain safety, here both proactive safety was applied in the form of maintaining speed and force limits, as well as giving the safety system the ability to correct bad behavior and return to a previous safe state if possible. Another advantage of the applied safety system is that it runs as a separate process, making it independent of the speed of the control system controlling the robot.

The work performed is a good foundation to build upon for other projects, as it is modellable for the employment in other applications, as well as laying the foundation for safety itself in the form of a set of safety rules and situations that can be applied to similar designs of robots.

In the literature studies conducted to increase knowledge about robot safety, one thing was striking. Namely, that there was no standardized taxonomy that was used as a template within these articles, but that different applications and companies defined safety differently in their own ways. Therefore, it was necessary to define a system of safety definitions based on a summary of other safety utilizations.