




Collaborative Robot Safety Project

L. Ahrens
G. Claessen
R. Hulst
B. Janssen
L.J.P Jeurgens
M.A.J. de Kort

Eindhoven, 2017

Fontys Hogeschool Engineering



Preface

This is the final document from the project; Collaborative robot safety system. The project team consists of six mechatronic engineering students. It the first team working on this project. In the following document the research of the project is described. The goal of the project is to make an industrial robot collaborative, without limiting the user to get near the robot. During the project the team was supervised by Onno Puts, Pavel Samalík and Falke Hendriks from the Fontys University of Applied Sciences. Also, Patrick Rietman from Pilz and Jeroen Snijkers from ABB assisted the team. Many thanks to all who supported this project.

Contents

Preface	I
Abstract.....	IV
1 Introduction	1
2 Risk assessment	2
2.1 Basic machine description	2
2.2 Machine specifications.....	3
2.3 Hazards and corresponding risks	4
2.4 Conclusions risk assessment	6
2.5 Recommendations risk assessment.....	6
3 System Requirements Document	7
3.1 Stakeholders	7
3.2 Use case	7
3.3 Requirements.....	10
4 Safety measure 1: padding	12
4.1 Required foam density.....	12
4.2 Impact absorption.....	14
4.3 Conclusions padding	16
4.4 Recommendations padding	16
5 Safety measure 2: covers	17
5.1 Concept evaluation	17
5.2 System development covers.....	21
5.3 Conclusions covers.....	22
5.4 Recommendations covers.....	22
6 Safety measure 3: vision system.....	23
6.1 Sensors	23
6.2 Software	39
6.3 Vision.....	40
6.4 Three dimensional positioning.....	41
6.5 Testing.....	41
6.6 Conclusions vision system.....	42
6.7 Recommendations vision system	42
7 Conclusions	44

8	Recommendations	45
	Bibliography	46
Appendix 1	Hazards, risks and risk reduction methods	47
Appendix 2	Requirements.....	55
Appendix 2.1	By intrinsic design	55
Appendix 2.2	By safeguards	61
Appendix 2.3	By information	62
Appendix 3	Test document	63
Appendix 4	Software code	67
Appendix 4.1	Vision test bluefox.....	67
Appendix 4.2	Vision test 2 (Background subtraction).....	69
Appendix 4.3	QR detection Bluefox	71
Appendix 4.4	QR detection Bluefox with backgroud subtraction.....	75
Appendix 4.5	QR webcam	79
Appendix 4.6	Move test ABB IRB120	83
Appendix 4.7	Move test ABB IRB140	86
Appendix 4.8	Place object	89
Appendix 4.9	Place walls	92
Appendix 4.10	Place 3D object.....	96

Abstract

The goal of the Cooperative Robot Safety Project is to build a safety system, making it possible to safely work in an area shared with an operating robotic arm. It is preferred to realize a modular safety system which can be applied on most robotic arms. The safety system needs to comply the Machinery Directive, which is a law regarding safe machinery.

The primary guide in developing this system is the Machinery Directive, which serves as a law for safe machinery. A risk assessment is the most important document in order to be able to comply with the Machinery Directive. From the risk assessment, all hazards, risks and risk reduction methods (using ISO standards as a guide) can be derived in order to create the safety system. Because of the state of the art of the application of the robot, not all requirements suggested by any applicable ISO standards could be met, but were implemented as much as possible.

The system that has been developed, consists of three safety measures:

1. Padding, serving as impact damping to prevent harmful collision with a person;
2. Covers to reduce the risk of body parts, clothes, hair or jewellery getting stuck, entangled or crushed in any gaps in the robot;
3. A vision system, used to adapt the speed of the robotic arm, depending on the distance to a person.

Using the above safety measures, all risks as defined by the risk assessment can be reduced to acceptable risks, which is defined as the lowest possible risk. Except for the risk corresponding to hazard number 1.4: crushing a person. This risk can only be reduced to a possible risk, which is considered to be less safe than an acceptable risk.

A demonstrator is built to serve as a prove of concept. The robot used for the demonstrator is the ABB IRB 140.

The following is recommended regarding:

1. The risk assessment;
 - a. Research the possibility to put the robot on a non-actuated movable base (e.g. a base on wheels, or by adding springs/dampers) to reduce the risk of hazard no. 1.4 (crushing a person between the robot and a foreign object) by creating more degrees of freedom. It can be advised to make a difference between crushing on a foreign (e.g. workpiece of wall) object and crushing on the robot, or the robot base.
2. Padding:
 - a. Due to lack of publicly available documentation, it is advised to test different foam samples for impact to determine materials best suited for the specifications according to this document;
 - b. It is furthermore recommended to consult human injury biomechanics to further optimize specifications for impact reducing materials.
 - c. Ideally the robot itself should be built using a less stiff (weak) exterior.
3. Covers:
 - a. Further research options to cover (fingertip size or smaller) gaps.
 - b. Ideally the shape of the robot should intrinsically be safe. Meaning not containing (small) gaps and unable to crush limbs. Current co-bots on the market, e.g. UR-5, serve as good examples.

4. The vision system.
 - a. Use multiple different sensors, for example use not only two dimensional cameras but a system consisting of a 2D camera, a 3D camera, and a volumetric radar.
 - b. Use a different setup than the ROS system. Due to non-real-time and offline planning problems. This does mean custom software should be developed.
 - c. Use a different technique to get a three-dimensional representation of the world around the robot.

1 Introduction

The high-tech industry is shifting more and more towards small batch production of high complexity. This creates the need for easily configurable robots in the robotic industry. This will solve issues where currently work must be done manually. Having easily reconfigurable robots which can safely work alongside workers will allow tasks to be done as needed.

The RAAK (G)een Moer Aan project is a cooperative project between the industry and education, overarching the Cooperative Robot Safety Project. The goal in the RAAK (G)een Moer Aan project is to realize a demonstrator, which is a robotic arm, that is able to autonomously tighten nuts and bolts in an area which is shared with workers. This raises questions regarding the safety of the system.

The goal of the Cooperative Robot Safety Project is to build a safety system, making it possible to safely work in an area shared with an operating robotic arm. It is preferred to realize a modular safety system which can be applied on most robotic arms. The safety system needs to comply the Machinery Directive, which is a law regarding safe machinery.

The remainder of this project is organized as follows. Chapter 2 provides a risk assessment. A System Requirements Document is provided by chapter 3. The design of the safety system is divided in three safety measures: padding, covers and a vision system, which are addressed respectively in chapters 4, 5 and 6. Conclusions and recommendations are treated respectively in chapters 7 and 8.

2 Risk assessment

This risk assessment is used to develop a safety system for the Collaborative Robot Safety System project as part of the Raak (G)een Moer aan Project. The goal of the project is to develop a modular safety system for robotic arm for collaborative use with humans.

A risk analysis is crucial in complying with the Machinery Directive, which is a law regarding safe machinery.

In the selection of the hazards, the additional hazards from application-specific components (e.g. end-effectors) are not considered, because these topics are out of scope for the safety system that is being developed in this project.

This document is based on ISO 12100, ISO 10218-1, ISO 10218-2, ISO 13849-1 and ISO TS 15066

Safety measures are taken using the process as displayed in Figure 1. This figure can be found in ISO 13849-1.

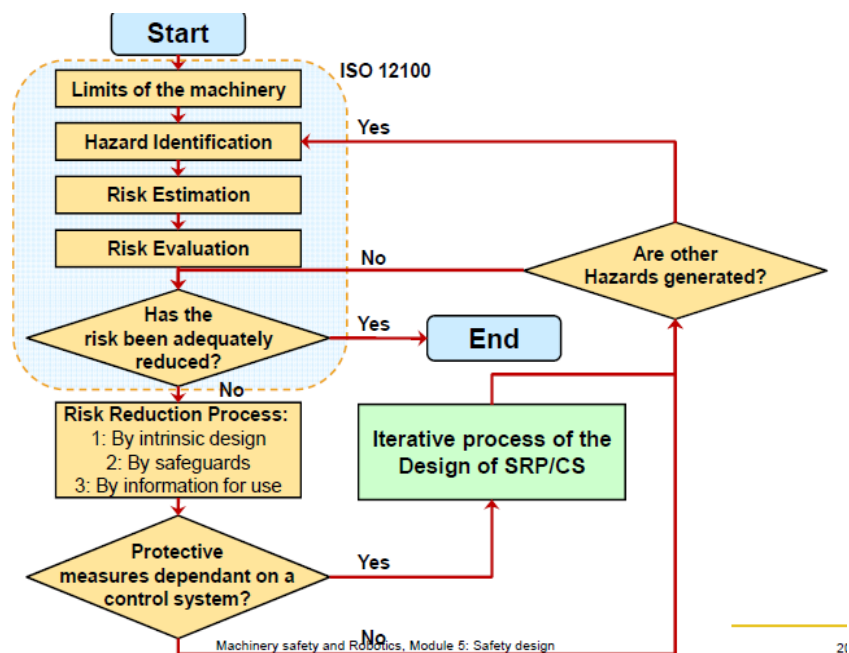


Figure 1 Procedure for taking safety measures

2.1 Basic machine description

The specifications of the robotic arm in the use case of the Collaborative Robot Safety System project are based on the ABB IRB140 model.

2.1.1 Intended use

The ABB IRB 140 robot, Figure 2, is an industrial robot. Without an additional safety system, the workspace of the ABB IRB 140 needs to be isolated from humans.

In this use-case, the main task of this robot is to tighten nuts and bolts on an auger (Figure 3) in a shared environment with humans. The ABB IRB 140 is in this case mounted on a stationary frame. This robot operates in an industrial workspace, where multiple tools and objects may enter its workspace. The length of a workpiece (on which an auger needs to be attached) may vary between 60 cm and 600 cm with a mass varying between 100 kg and 3000 kg.



Figure 2 ABB IRB 140 robot. No end-effector is attached

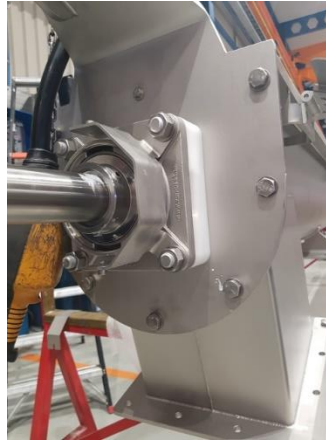


Figure 3 Auger with nuts and bolts

2.1.2 Machine components

The ABB IRB 140 consists of a base with two links and an end-effector. The end-effector can be taken off the robot and switched with another end-effector for a specific task.

2.2 Machine specifications

The machine specifications consist of the machine limits, operational and maintenance information, and power source. These specifications can be found respectively in Table 1, Table 2 and Table 3.

Table 1 Machine limits

Machine limits	
Machine Name/Type	ABB IRB 140
Intended Environment	Indoors industrial
Intended Use	Tighten nuts and bolts in a workspace which is shared with humans
Robot mass	98 [kg]
Robot payload	6 [kg]
Max speed	2.5 [m/s]
Machine Dimensions	Radius of 810 [mm]
Machine Environment	Indoors, dry, clean, non-explosive, non-flammable

Table 2 Operational and maintenance information

Operational and maintenance information	
Operational Information	
No. of Operators	1
Maintenance Operation	
Maintained by	Operator, Maintenance Technician
Maintenance Frequency	Weekly
Cleaning	Operator
Jamming repair	Operator

Table 3 Power source

Power source	
Main Feed, Elec. Supply:	200–600 V, 50/60 Hz
Pneumatic Supply	Not Applicable
Hydraulic Supply	Not Applicable

2.3 Hazards and corresponding risks

2.3.1 Hazard identification

Hazards are determined by considering the following sources:

- Annex I of ISO 10218-1
- Annex I of ISO 10218-2
- ISO TS 15066
- Reasonable foreseeable misuse
- Common sense

2.3.2 Risk classification method

Two different risk analysis methods are considered: the Fine & Kinney method and the 14-point risk graph method.

For this risk analysis, the Fine & Kinney method was best suited according to the pros and cons in Table 4. The possibility of avoidance of the hazard is not considered in this risk assessment, because all hazards will be considered unavoidable. To determine performance levels (PL) in a later stage of risk reduction, all hazards will again be considered unavoidable.

Table 4 Methods for risk analysis'

Fine & Kinney		14-point risk graph	
Pros	Cons	Pros	Cons
Prioritizing is easy	Outcome varies per assessor	Unambiguous results from different assessors	Prioritizing is difficult
Experience within the team			

The downside of the Fine & Kinney method is a varying outcome per assessor. This issue is tackled by determining risk values in two groups of two team members. Inconsistent results are then discussed with all six team members. Ambiguous results can be clarified using this method.

All hazards are ranked at different risk levels, which consist of grades for severity, exposure time and probability, according to formula 1. For each of these factors, a specific grade is assigned according to a given consequence, duration or chance. The combination of each grade and corresponding consequence is based on values taken from the Fine & Kinney method.

The classification by Fine & Kinney is considered unsatisfying by the project team, because of a lack of options. Additional consequences, time indications and chances are added as suggested and agreed upon by all six team members. Corresponding grades are determined by adding all newly suggested grades and dividing the result by the number of suggestions (i.e. participating team members). The grading process is then repeated to rule out unsatisfying results.

Grading for Severity, Exposure time and Probability can be found respectively in Table 5, Table 6 and Table 7.

$$\text{Risk} = \text{Severity} \times \text{Exposure time} \times \text{Probability} \quad (1)$$

Table 5 Grading severity

Severity (S)	
Grade	Consequence
1000	Many fatalities
300	Multiple fatalities
200	One fatality
100	Major permanent injury
37.5	Minor permanent injury/ Major fracture/ Grade 3 burns/ Deep cut
22	Minor fracture/ Grade 2 burns/ Shallow cut
6.25	Absent from work/ Grade 1 burns
1.5	First aid treatment

Table 6 Grading exposure time

Exposure time (E)	
Grade	Duration
10	Constant
8	Hourly
6	Daily
3	Monthly
2	Weekly
1	Several times a year
0.5	Very seldom

Table 7 Grading probability

Probability	
Grade	Chance
10	Almost surely
6	Quite possible
3	Unusual but possible
1	Possible under certain

	conditions
0.5	Very unlikely
0.2	Nearly impossible
0.1	Virtually impossible

Finally, the risk is classified according to Table 8. This classification table is a scaled version of the original Fine & Kinney version, in order to match the grading system that is being used in this risk assessment.

Table 8 Risk classification table

Risk classification table		
Very High Risk = Stop	$R > 5000$	
High Risk = Immediate Measures	$2000 < R < 5000$	
Important Risk = Short-term Measures	$500 < R < 2000$	
Possible Risk = Long-term Measures	$100 < R < 500$	
Acceptable Risk = No Measures	$R < 100$	

All hazards with corresponding risks and risk reduction methods can be found in Appendix 1.

Hazards are not specific for the ABB IRB140, they can apply for other brands and types of robotic arms. As stated in section 2.3.1: in the selection of the hazards, the additional hazards from application-specific components, such as end-effector, payload, fixtures, working environment, etc. are not considered.

2.4 Conclusions risk assessment

All safety hazards, except for hazard number 1.4, can be reduced to an acceptable risk by:

1. Adding impact reducing padding to the robot;
2. Covering the robot so human body parts and clothing cannot get stuck in the robot;
3. Adapting the speed of the robotic arm, depending on the distance to a person.

2.5 Recommendations risk assessment

Research the possibility to put the robot on a movable base (e.g. a base on wheels, multi-direction) to reduce the risk of hazard no. 1.4. It can be advised to make a difference between crushing on a foreign (e.g. workpiece or wall) object and crushing on the robot, or the robot base.

3 System Requirements Document

The purpose of the System Requirements document is to specify the overall system requirements that will govern the development and implementation of the system. Requirements will be tested according to the test document in Appendix 3

3.1 Stakeholders

There are different stakeholders involved in this project. Below you can find a list of project members and their role, a list of the tutors from Fontys who are involved in this project, and the remaining stakeholders.

Student number	Student name	Role	E-mail	Phone number
2163650	Luuk Jeurgens	Project member	luuk.jeurgens@student.fontys.nl	0681251776
2206481	Bas Janssen	Contact/communication	smh.janssen@student.fontys.nl	0652021237
2222879	Rens van Hulst	Project member	rens.vanhulst@student.fontys.nl	0637421279
2223006	Michiel de Kort	Project member	michiel.dekort@student.fontys.nl	0642593390
2327163	Luc Ahrens	Project leader/Safety specialist	l.ahrens@student.fontys.nl	0636464987
2345455	Guido Claessen	Project member	g.claessen@student.fontys.nl	0640538864

Name	Role	E-mail
Falke Hendriks	Safety Specialist	f.hendriks@fontys.nl
Pavel Samalík	Tutor	p.samalik@fontys.nl
Onno Puts	Project owner	o.puts@fontys.nl

Name	Interest
ABB	- Involved in project, supplier of robotic arm for design and test phase
RAAK Geen Moer Aan	- Project Use case, this project is setup on the bases of the RAAK Geen Moer Aan project.
Client (Everyone who buys the product)	- Interested in a safety system which allows workers to collaboratively work with a robotic arm. - Want lowest price possible
Operator/end-user (Everyone who use the product)	- Ease of use - Increase work efficiency
Legislators	- Legal status - Compliance with applicable directives like ISO-standards
Financial Director	- Cost of product - Cost of operation - Cost of maintenance

3.2 Use case

The use case of this project is the company 'van Beek' they are the case study for the 'RAAK (G)een moer aan' project. It is intended that a collaborative robotic arm will be implemented at the production line at this company.

3.2.1 Van Beek

Van Beek is a company consisting of 40 employees which make client specific auger systems which are used in a vast range of industries. Such as food, chemical, oil, agriculture, etc.

About van Beek:

- Employee age range: from students which are learning and working at the same time, to 50 year old workers.
- 7 engineers for designing to customer needs.
- Only men are doing the assembly of the systems, these are MBO educated.
- All workers are all-round workers, they are all able to assemble, weld, roll, and test.
- At van Beek work is done in cells, each cell is devoted for rolling, welding, assembly, etc.
- The cells can be easily made bigger and smaller, to fit the needs for the sizes of the augers. The height of the hall is 8 meters.
- The factory is divided up into two parts, one side is for stainless steel and one side is for steel
- If there is a system that is being made for the food industry, because of regulations they close the doors in between the two parts of the factory to avoid contamination.

About the product:

- The augers which they use range from 30cm to 12 meter in length.
- The augers they use range from 2cm to 50 cm in diameter.
- The cover in which the auger is placed has lids on it, the average number of lids is 1 to 2, but can vary up to 7.
- The standard 200 model uses 6xM12 bolts and nuts and one M8 bolt and nut to attach one end plate. And two washers for each bolt/nut combination.
- Assembly will only start when all traffic lights are green, this means that all the products are at the factory and are ready to be assembled.
- The assembly of one system is being done by 1 or two employees
- At one time, there could be 3 to 4 workers working in the assembly cell
- The assembly process takes on average 30 minutes, but for bigger systems it can take up to half a day.
- The assembly is about 10% to 15% of the total time spend on a system.
- The total system takes about 4 to 10 weeks to be completed.
- One or two people normally do the whole process for one system

Goals of robot collaboration:

- Reach a higher quality level.
 - o Never forget to tighten a bolt.
 - o Always with the correct torque.
- Robot can work non-stop. This means more production time.
- Make work lighter for employees.
- Take out repetitive work processes.

3.2.2 Use case diagram

Figure 3 depicts the use case diagram of the project within the given project borders. A use case diagram describes the relations between roles, systems and users on a higher level. There are two given users, the user and the service team. Next to that there are two given systems, the robot arm and the safety system. Between those, there are multiple roles (oval shapes). The lines are the relations between these entities.

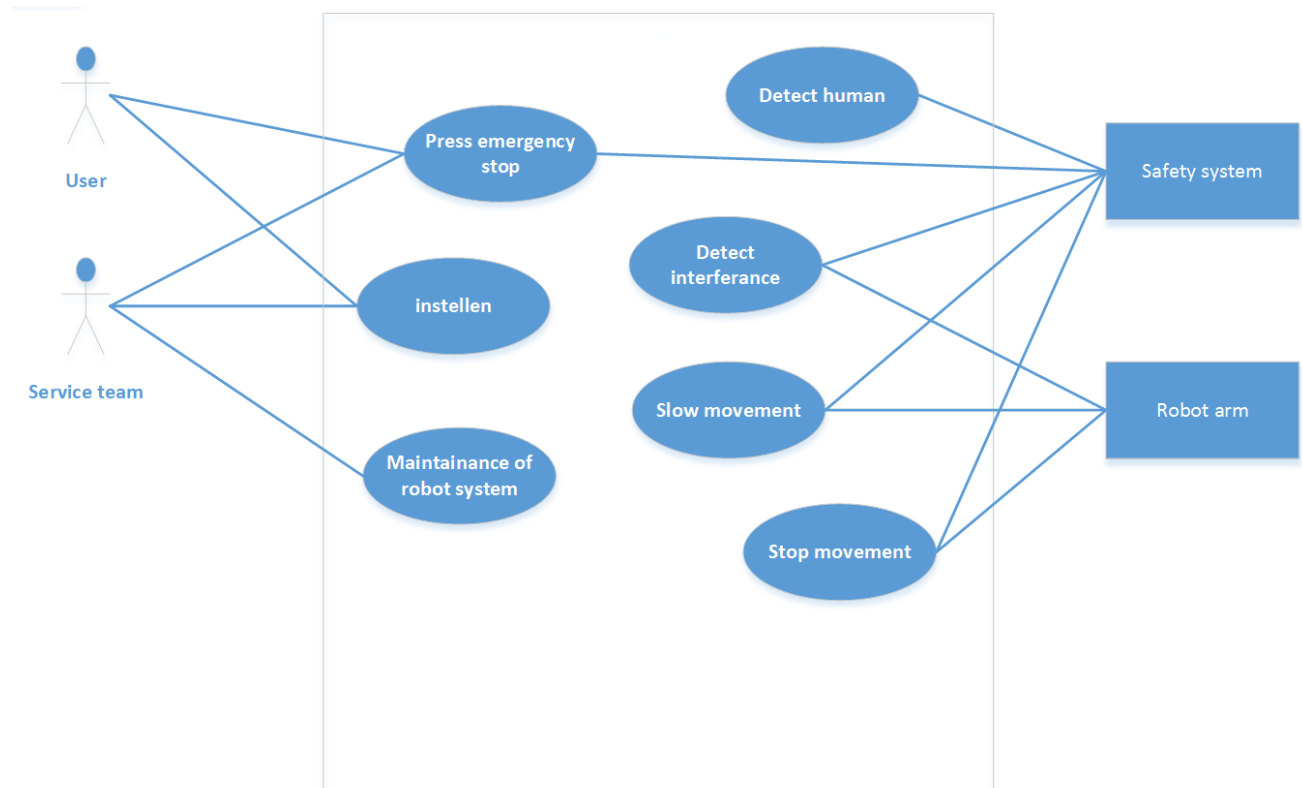


Figure 3 Use-case diagram

3.3 Requirements

In order to deliver a quality product that meets all the stakeholders' needs and wishes, a set of requirements needs to be set up. This is done by consulting the client and ISO norms.

3.3.1 Method

All requirements are important, but they are prioritized to deliver the most beneficial requirements early. Initially one will try to deliver all the 'Must have' requirements, and then 'Would like to have' or wishes of the clients will follow. The wished for, but not mandatory requirements will be the first to be removed if the delivery timescale looks threatened. The prioritization of requirements has its value in getting a focus point for the project and quality assurance. In order to categorize the requirements, different methods could be used. Some of the considered methods are listed below.

Methods:

- MoSCoW
- Shall, Will, Must
- High, Medium, Low, Nice-to-Have

3.3.1.1 MoSCoW Method

The method which will be followed during this project is the MoSCoW method. The choice has been made to use this method over the others because it makes it very clear on which requirements must be focused first. This is important to deliver a product that meets all the customer's requirements within the deadline. The MoSCoW method clearly describes how the requirement will be categorized compared to other methods like the Shall, Will, Must method. The categories of the MoSCoW method will be explained in more detail in the next section.

The name MoSCoW method comes from the abbreviations of Must, Should, Could, and Won't. Requirements will always fall under one of these four categories. The categories for the MoSCoW method are prioritized as:

Must have:

Requirements labelled as 'MUST' are critical to be delivered at the end of the project in order for the project to be a success. If even one 'MUST' requirement is not met, the project delivery should be considered a failure.

Should have

Requirements labelled as 'SHOULD' are important but not necessary for the project to succeed. While 'SHOULD' requirements can be as important as 'MUST', they are often not as time critical or there may be another way to satisfy the requirement, so that it can be held back until a future follow up.

Could have

Requirements labelled as 'COULD' are desired but not necessary. They could for example improve user experience or customer satisfaction for little development cost. These will typically be included if time and budget permit.

Won't have

Requirements labelled as 'WON'T' are considered as the least critical, lowest payback items, or not appropriate at that time. As a result, 'WON'T' requirements are not planned into the schedule for the current project. WON'T requirements are either dropped or just for reference as inclusion in a later follow up of the project.

3.3.2 Requirements listed by Risk Reduction Process

The requirements (Appendix 2) are categorized and listed based on the risk reduction processes which include reduction by intrinsic design, safeguards, and user information.

4 Safety measure 1: padding

The padding is a soft cover on the robot. This padding will serve as impact damping to prevent harmful collisions with a person. The choice for padding derives from the risk assessment, chapter 2, in which the padding reduces the severity of any impact injury. The maximum permissible pressure on a human temple (most vulnerable body part) before reaching the onset of pain will be used as a main guideline to conclude the feasibility of the determined foam characteristics. In an ideal situation all kinetic energy of the robot would be absorbed by the padding.

4.1 Required foam density

The desired energy absorption by the padding can be calculated using equation (1):

$$\underbrace{\frac{1}{2}m_r v_r^2}_{\text{Robot speed}} = \left(\underbrace{\frac{1}{2}m_h v_h^2}_{\text{Human speed}} + \underbrace{\frac{1}{2}k_h \cdot \Delta l_h^2}_{\text{Human deformation}} \right) + \underbrace{\frac{E_p}{V_p} \cdot A_p l_p}_{\text{Absorption padding}} \quad (1)$$

Where:

m_r = Mass robot [kg];

v_r = Maximum speed robot [m/s];

m_h = Mass person [kg];

v_h = Speed person after collision [m/s];

k_h = Spring constant of human body part [N/m];

l_h = Deflection distance of human body [m];

$\frac{E_p}{V_p}$ = Compression energy of the padding per volume [J/m^3];

A_p = Compressed Surface area padding [m^2];

l_p = Compression distance padding [m].

Since it is not desired to transfer energy to a person, the speed of a person after collision and the compression of the human body will be neglected.

For this calculation a robot speed of $0.4 \text{ [} m/s \text{]}$ is chosen. This is 60% faster than the speed limit given to the robot by ISO TS 15066 [1]. This provides an extra safety margin. The mass of the robot is determined by equation (2) [1]:

$$m_r = \frac{M}{2} + M_L = \frac{98}{2} + 6 = 55 \text{ kg} \quad (2)$$

Where:

m_r = mass of the robot [kg];

M = mass of the moving parts of the robot [kg];

M_L = payload of the robot [kg].

The compressed surface area of the padding is equal to the most vulnerable surface area of the human body, which is the temple [1] (located on the side of the head) with an estimated surface area of $25 \text{ [} cm^2 \text{]}$. In order to prevent excessive increase in the volume of the robot arm, the desired value for the compression distance of the padding (which is assumed to be 50% of the thickness of the padding) is

preferred to be small. An estimated 5 [mm] compression distance will be used to calculate the desired parameters for a foam.

The energy equation (1) can now be completed in equation (3), assuming no energy will be transferred to a person.

$$\frac{E_p}{V_p} = \frac{m_r \cdot v_r^2}{2A_p l_p} = \frac{55 \cdot 0.4^2}{2 \cdot 2.5 \cdot 10^{-3} \cdot 5 \cdot 10^{-3}} = 3.52 \cdot 10^5 \left[\frac{J}{m^3} \right] = 0.352 \left[\frac{J}{cm^3} \right] \quad (3)$$

The energy absorption per volume of the padding needs to be at least $0.352 \left[\frac{J}{cm^3} \right]$.

The graph in Figure 4 [2] displays the energy/volume plotted against the corresponding density of a polyurethane foam. With an energy per volume of $0.352 \left[\frac{J}{cm^3} \right]$, the corresponding density is $0.23 \left[\frac{g}{cm^3} \right]$.

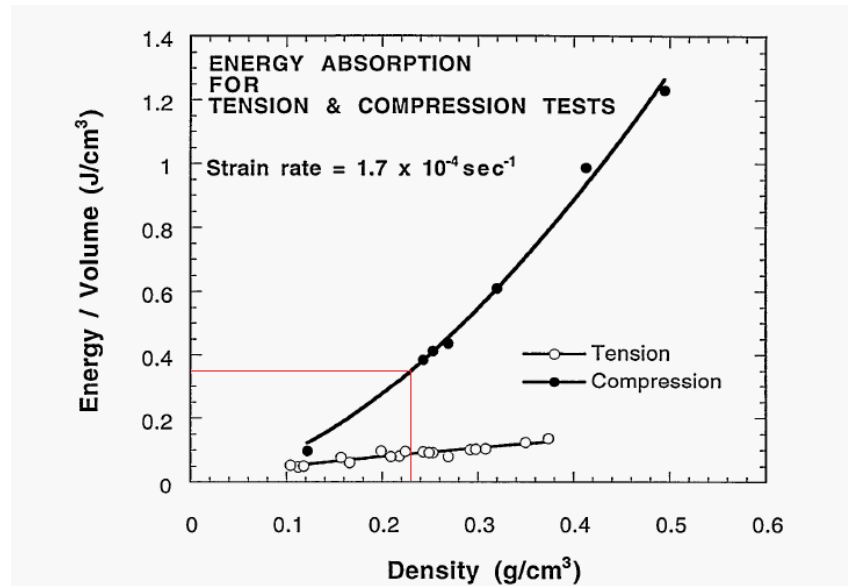


Figure 4 The energy absorption of the foam vs. density

The figures 1, 2 and 3 are the result of a test, performed at room temperature with polyurethane foam cylinders with a length of 50,8mm and a diameter of 28,7mm. The tests were performed on a conventional Instron test frame as shown in Figure 5.

Figure 6 [2] shows the amount of energy/volume absorption plotted against the density for impact testing. A foam density of $0.23 \left[\frac{g}{cm^3} \right]$, corresponds to an absorption of $0.92 \left[\frac{J}{cm^3} \right]$. This is sufficient for the intended application at $0.352 \left[\frac{J}{cm^3} \right]$.



Figure 5 Instron test frame

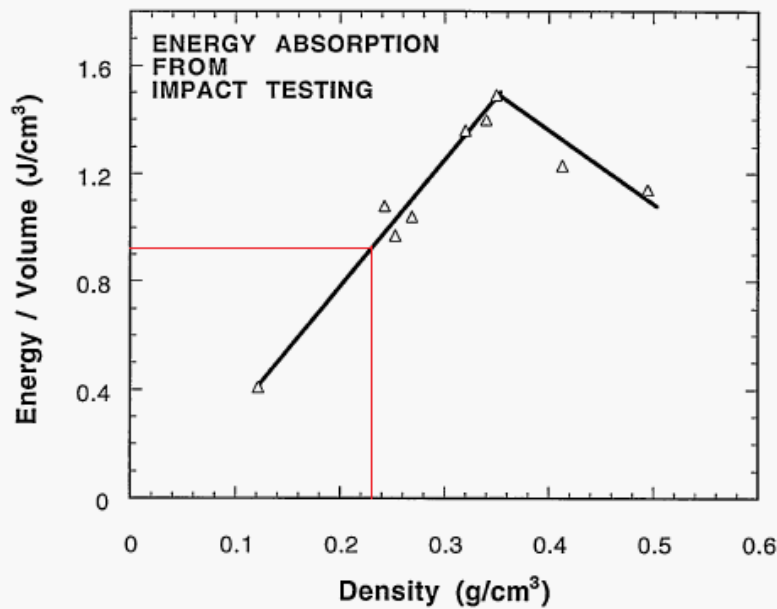


Figure 6 Energy absorption of CRETE vs. density for impact testing

4.2 Impact absorption

The maximum permissible pressure on a human temple (most vulnerable body part) before reaching the onset of pain is equal to 1.1 [MPa] [1].

According to Figure 7 [2], the foam density corresponds to a stress at 10% compression. A foam density of 0.23 $\left[\frac{g}{cm^3}\right]$ corresponds to a stress of 4 [MPa].

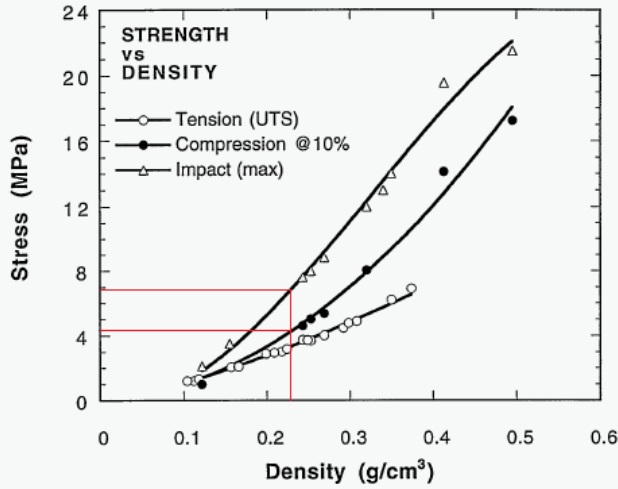


Figure 7 Maximum stresses developed in the foam

The pressure exerted by the robot on the padding with the surface of a temple (human head) is calculated using equation (4).

$$P = \frac{F_r}{A_p} \quad (4)$$

Where:

P = Exerted pressure [Pa];

F_r = Impact force robot [N];

A_p = Compressed surface area padding [m^2].

The impact force of the robot is calculated using Newtons second law, equation (5).

$$F_r = m_r \cdot a \quad (5)$$

Where:

a = Average deceleration robot after impact [m/s^2].

The average deceleration of the robot after impact is calculated in equation (6). Using the maximum speed of the robot and the desired compression distance of the padding, which are respectively $0.4 [m/s]$ and $5 \cdot 10^{-3} [m]$ (both stated before in this chapter).

$$a = \frac{l_p}{(l_p/v_r)^2} = \frac{5 \cdot 10^{-3}}{(5 \cdot 10^{-3}/0.4)^2} = 32 [m/s^2] \quad (6)$$

Equation (4) can now be completed in equation (7). Using the mass of the robot as calculated in equation (2) and the desired compressed surface area of the padding as stated previously in this chapter.

$$P = \frac{F_r}{A_p} = \frac{m_r \cdot a}{A_p} = \frac{55 \cdot 32}{2.5 \cdot 10^{-3}} = 0.704 [MPa] \quad (7)$$

4.3 Conclusions padding

A padding with a density of $0.23 \left[\frac{g}{cm^3} \right]$ and a compression distance of $5 [mm]$ is sufficient, since the pressure exerted by the robot is $0.704 [MPa]$ at robot mass of $55 [kg]$ and a speed of $0.4 [m/s]$. The pressure exerted by the robot is less than the maximum permissible pressure and the absorbed pressure by the padding combined.

4.4 Recommendations padding

Due to lack of publicly available documentation, it is advised to test different foam samples for impact in order to determine materials best suited for the specifications according to this document.

It is furthermore recommended to consult human injury biomechanics to further optimise specifications for impact reducing materials.

Ideally the robot itself should be build using a less stiff (weak) exterior.

5 Safety measure 2: covers

The second safety measure, according to the risk assessment in chapter 2, consists of any covers on the robot, which intend to reduce the risk of human body parts, clothes, hair or jewellery getting stuck, entangled or crushed in any gaps in the robot.

5.1 Concept evaluation

A series of concepts will be evaluated by a description and a sketch or impression per concept. For every concept, advantages and disadvantages will be listed in order to choose a concept in §5.1.6.

5.1.1 Cover concept 1: radius

The first concept is to fill all the gaps by adding material on the joints, so the gaps are always filled. The concept is illustrated in Figure 8, where surfaces with black stripes represent a robotic arm and the clear surfaces with a blue periphery represent the radius. This concept is easy to mount since it doesn't require any extra mounting points, it can be clamped to the robot. Because this concept can be made with a 3d printer it is easy to customize. More advantages and disadvantages of the radius concept are listed in Table 9.

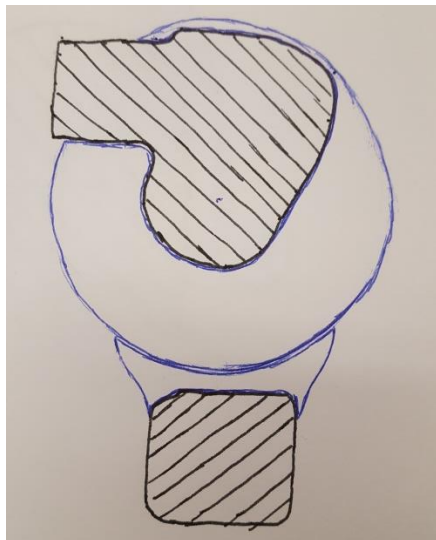


Figure 8 Radius concept

Table 9 Advantages and disadvantages of the radius concept

Advantages	Disadvantages
Cheap	Not modular
Easy to produce with 3d printing	Increases volume
Easy to mount, no extra mounting points	Small gaps might still occur
Does not obstruct the movement of the robot	

5.1.2 Cover concept 2: lamellae

The second concept is to use lamellae to cover the gaps. The lamellae are on top of each other and can slide over each other to adjust to the movement of the robot. The concept is illustrated in Figure 9, where surfaces with black periphery represent a robotic arm and surfaces with blue periphery represent the lamellae. Advantages and disadvantages of the lamellae concept are listed in Table 9.

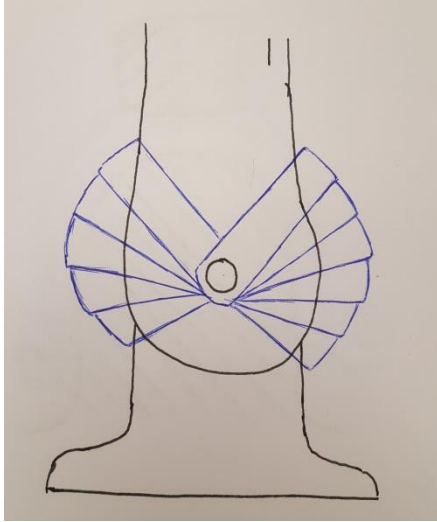


Figure 9 Lamellae concept

Table 10 Advantages and disadvantages of the lamellae concept

Advantages	Disadvantages
Cheap	Small gaps might still occur
Easy to produce	Difficult to mount
Modular	

5.1.3 Cover concept 3: accordion

The third concept is to put a cover over the robot which looks like an accordion. This concept is illustrated in Figure 10, where surfaces with black periphery represent a robotic arm and surfaces with blue periphery represent the accordion. Figure 11 provides an impression for the accordion concept. Advantages and disadvantages of the accordion concept are listed in Table 11.

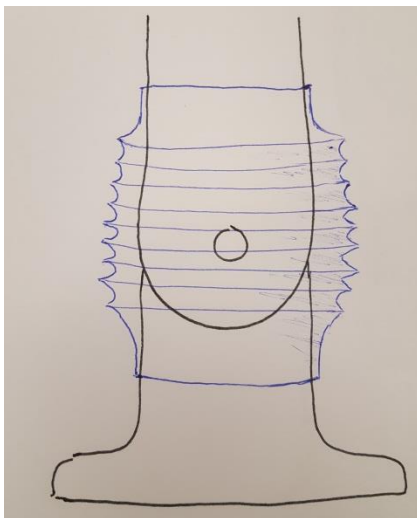


Figure 10 Accordion concept



Figure 11 Accordion concept impression

Table 11 Advantages and disadvantages of the accordion concept

Advantages	Disadvantages
Covers the whole robot	The material is flexible so you can push it between the joints
Easy to mount	Not easy to produce
Modular	Expensive

5.1.4 Cover concept 4: elastic bag

The fourth concept is an elastic bag, capable of following the curves of the robot. Figure 12 provides an impression for the accordion concept. Advantages and disadvantages of the elastic bag concept are listed in Table 12.

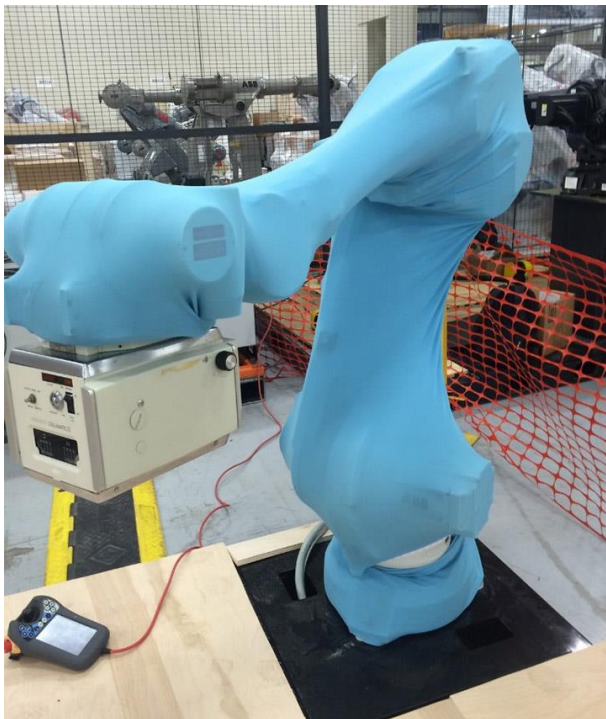


Figure 12 Elastic bag concept

Table 12 Advantages and disadvantages of the elastic bag concept

Advantages	Disadvantages
Covers the whole robot	The material is flexible so you can push it between the joints
Easy to mount	Wear sensitive
Modular	Could get stuck in the robot

5.1.5 Cover concept 5: rolling shutter

The last concept is to use a rolling shutter to cover any gaps. Both sides of the rolling shutter are mounted on the robot, when the robot moves the shutter roles in or out. This concept is shown in Figure 13, where surfaces with black stripes represent a robotic arm and surfaces with blue periphery represent the rolling shutter. Advantages and disadvantages of the rolling shutter concept are listed in Table 13.

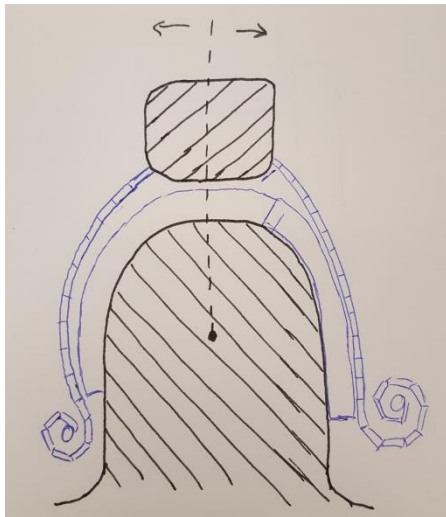


Figure 13 Rolling shutter concept

Table 13 Advantages and disadvantages of the rolling shutter

Advantages	Disadvantages
Covers the whole robot	Not modular
Easy to mount	Difficult to produce
	Expensive

5.1.6 Concept selection

Comparing the advantages and the disadvantages of all concepts, the best option is to make a cover by using a radius. This concept is easy to make as well as easy to mount. The cover does not require extra mounting points since it can be clamped on the robot arm. The cover is easy to customize and can be made with a 3d-printer. The cover does not obstruct the movement of the robot, it only makes the arm a bit bigger. Small gaps may still occur.

5.2 System development covers

After evaluating the different covers, the radius concept, in which the gaps are filled with an object with the corresponding radius, is chosen to be the best concept. The cover that is designed is a radius for the fourth joint of the ABB IRB 120 robot. This robot is not the robot that is used as a demonstrator in the current project, but the ABB IRB 120 is the robot that Fontys owns and will use in future projects.

The cover exists of 3 parts. The first part (Figure 15) is located on top of the support beam in joint 3. This is illustrated in Figure 14 with the number 1. The other two components in Figure 16 can be put together around joint 4. They are supported with nuts and bolts as shown in Figure 16. This is illustrated in Figure 14 with number 2. Together these components fill the gap so nothing/nobody can get entrapped.

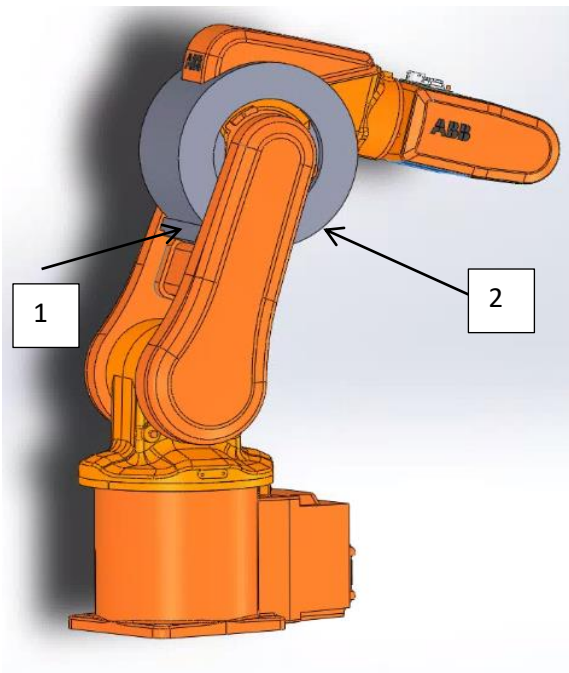


Figure 14 Total Concept

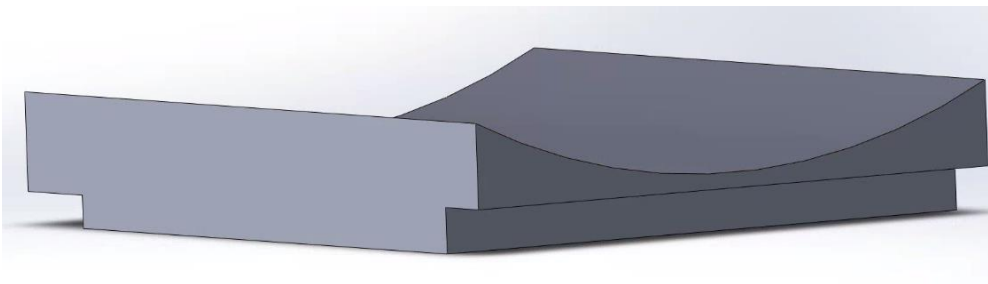


Figure 15 Part 1

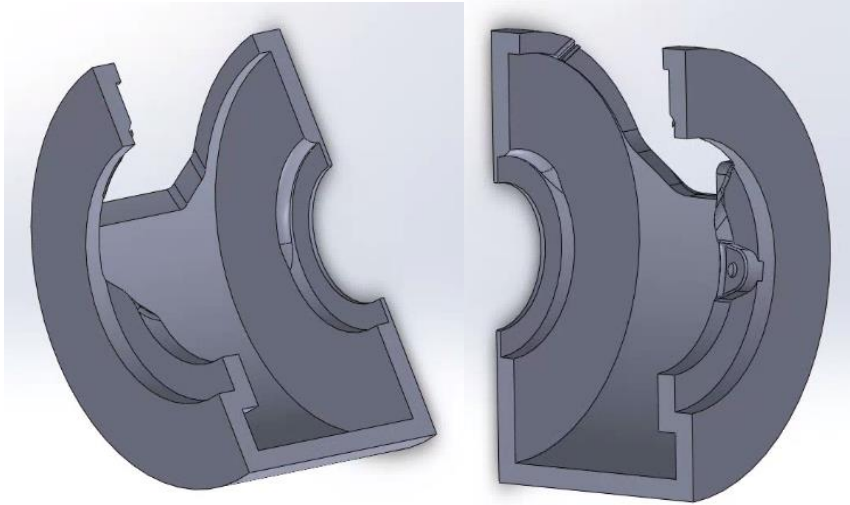


Figure 16 Parts 2 and 3

5.3 Conclusions covers

A radius is chosen as the best suitable cover concept. It is easy to make as well as easy to mount. The cover does not require extra mounting points since it can be clamped on the robot arm. The cover is easy to customize and can be made with a 3d-printer. The cover does not obstruct the movement of the robot, but it increases its volume. Small gaps may still occur.

The radius that is designed is a cover for the fourth joint of the ABB IRB 120 robot. This robot is not the robot that is used as a demonstrator in the current project, but the ABB IRB 120 is the robot that Fontys owns and will use in future projects. The radius is produced with a 3d-printer using ABS.

5.4 Recommendations covers

Further research options to cover small (fingertip size or smaller) gaps.

Ideally the shape of the robot should intrinsically be safe. Meaning not containing (small) gaps and unable to crush limbs. Current co-bots on the market, e.g. UR-5, serve as good examples.

6 Safety measure 3: vision system

This chapter describes the components of the safety system that has been built as a demonstrator. First the sensors used in the demonstrator will be defined. Next the software used for the system is detailed, including the base layer and the custom code. After the overview a detailed explanation of the employed computer vision is provided, which is followed by the attempt at three dimensional reconstruction. Ending this chapter are the conclusion and recommendations.

6.1 Sensors

As part of the collaborative robot safety project, all items located within the robot's working area need to be detected. By all items one should think about objects, people, and the robot itself. These items need to be detected in order to adaptive plan the robot's path to avoid coming in contact and potentially hurting a worker or damaging an object. To realize this research has been done on different type of sensors and a couple sensor concepts have been worked out, these are described in the sections below.

6.1.1 Sensor Research

6.1.1.1 Radar

In this chapter the different types of radar will be discussed. Questions like, is radar suitable for this project, what are its advantages and disadvantages will be answered? At last there will be conclusion formulated.

6.1.1.2 2D Radar

An air surveillance radar covers an area around the radar, and listens for echo signals. Its antenna pattern is adjusted for the specific task. Mostly the antenna forms a rotating beam. This form of volume scan is then called 2D radar. Such radar can measure only two coordinates to determine the position. It can detect two superimposed flying airplanes only as a single (but then larger) target.

For the third coordinate, the height information, the system had to work with two radars. One of the radar was working as surveillance radar and the second radar device was then specialized as a so-called height finder [3]. Both types, the search radar and the height finder, can measure only two coordinates. In both cases it is 2D radar equipment.

In used radars by the military the cost factor plays a subordinate role. However radars in air traffic control must not be too expensive. Therefore, only 2D radar are used. The altitude information is then provided by the secondary radar.

6.1.1.3 3D Radar

If the measurements of all three space coordinates are made within a radar system, this one is called as 3D Radar. A special form of 3D radars is the weather radar using a very narrow beam. After each scanning rotation, the antenna elevation is changed for the next sounding. This scenario will be repeated on many angles to scan all the volume of air around the radar within the maximum range. For a complete cycle with rotation and tilt of the antenna in all directions takes up to 15 minutes [3]. This time approach is not suitable for air surveillance radar because very fast aircrafts can travel a huge distance in a very short time.

3D radar had to exist of multiple receive channels in parallel, and the radar antenna had to deploy multiple antenna patterns during the receiving time. Such radar was the Medium Power Radar (MPR),

which is no longer in service today. Its huge parabolic antenna has 9 meter horns and configures a total of 12 different narrow antenna beams that are aligned one above another. The targets height is calculated from this elevation angle and the measured distance. An extremely high pulse power had to be transmitted simultaneously in all directions during transmission.

Older 3D radar device having a linear phased array does not transmit simultaneously in all directions under observation. This is done sequentially in time. These antennas can only scan the space within a limited angle. Here there are two possibilities: either the antenna is rotating and electronically scans the elevation angle, or there are four planar antennas statically distributed around a carrier. Both variants allow an overall coverage around the radar site. Here, the radar transmits only in a certain direction and then waits for the echo signal from this direction.

The rotating antenna has a critical disadvantage. Because each elevation angle will be scanned sequentially in time, the antenna must not rotate too fast, to avoid. However, the version using the static antennas has the advantage in time scheduling that virtually four radars scan simultaneously the space. These four radar front-ends are subject of a common radar data processing and display. Here, the radar system can operate much more flexible and it can be used as multifunction radar.

Since the use of the advantages of digital beamforming and the possible parallel digital processing of all receiving channels, this timing problem is completely overcome.

6.1.1.3.1 3D-scan millimetre-wave radar:

Small radar which fits in the palm of your hand, but it is made for long range. The Fujitsu ERBA is the module with the most precise and shortest range. This module has a range of 0.2 – 15 meters. And a resolution of 0.2 meters [4] [5]. Unfortunately, the resolution is not precise enough for our purpose since it needs to be able to detect small objects as well, like a pinkie finger.

6.1.1.3.2 Google Soli:

The google Soli is a new sensing technology that uses miniature radar to detect touchless gesture interactions. This technology can detect sub-millimetre motions with great accuracy but it does not output location or other 3D information since it uses Doppler radar [6]. This makes it not suitable for our project.

6.1.1.3.3 Advantages and Disadvantages for radar

Advantages

- Radar has no influence on persons so it is safe to work close to a radar
- The data from a radar is very accurate
- With radar it is possible to scan 360 degrees in a flat surface so the whole working space can be mapped
- Radar is less effected by environmental variables like light compared to 2D cameras
- Radar can look through objects so there will be more information on its surroundings

Disadvantages

- Radar does not fit in the project budget
- Radar can affect other devices and cause distortion
- The minimum sensing range of radar is limited so small objects like finger tips can not be detected

6.1.1.3.4 Conclusion and recommendations radar

Radar is accurate to the millimetre and the needed 3D information can be obtained by this measuring technique for this project. It solves the need for constant light environments and is very fast. The down side is that radar is very expensive, some quotes were requested and they ranged from 7,500 for a 2D scanner to 65,000 euro for a 3D scanner. On top of that radar is normally used for long distances and bigger object detection than wished for in this project. The smallest resolution which can be found at the moment is for 20 cm which is too big for this project since an object the size of a pinkie finger needs to be detected. At last since radar uses radio frequency waves, it could cause EMC interference with other devices and fail. This is not acceptable in order to guarantee safety. Ultimately the price and resolution makes it not suitable for this project.

Google is currently developing a new sensing technology which is the Soli. It uses miniature radar to detect touchless gesture interactions. This technology can detect sub-millimetre motions with great accuracy but it doesn't output location or other 3D information since it uses Doppler radar. This makes it useless for our project but depending on the price of a module, many of them could possibly be placed on the robotic arm and end effector to detect objects just before collision and stop the robotic arm. This idea is similar to the Bosch APAS but then instead of using capacitive sensing one would use the Google Soli.

6.1.1.4 Stereo Vision

Computer stereo vision is the extraction of 3D information from digital images or projected light sources. By comparing information about a scene from two capture points, 3D information can be extracted by examination of the relative positions of objects in the two images. In traditional stereo vision, two cameras, displaced horizontally from one another are used to obtain two differing views on a scene (passive stereo vision). By comparing these two images, the relative depth information can be obtained in the form of a disparity map, which encodes the difference in horizontal coordinates of corresponding image points. The values in this disparity map are inversely proportional to the scene depth at the corresponding pixel location

6.1.1.4.1 Pre-processing

In a computer vision system, several pre-processing steps are required to obtain accurate information. For the collaborative robot safety project this has several important impacts. First of all, the information obtained from the stereo vision needs to be accurate in order to assure safety and prevent collisions. The other reason is that the information obtained from the stereo vision needs to be used real-time to control the robotic arm. This means that the processing time required needs to be limited to under a second [7].

The pre-processing steps are as follows:

1. The image must first be undistorted, such that barrel distortion and tangential distortion are removed. This ensures that the observed image matches the projection of an ideal pinhole camera.
2. The image must be projected back to a common plane to allow comparison of the image pairs, known as image rectification.
3. An information measure which compares the two images is minimized. This gives the best estimate of the position of features in the two images, and creates a disparity map.

4. Optionally, the received disparity map is projected into a 3D point cloud. By utilizing the cameras' projective parameters, the point cloud can be computed such that it provides measurements at a known scale.

6.1.1.4.2 Passive vs Active stereo vision

From a single image, it is not generally possible to obtain information about the shape, location, and orientation of 3D objects in a viewed scene. Passive stereo vision approaches use two or more calibrated cameras in distinct locations to solve this problem. However, the difficult problem of determining which points in both images correspond to the same 3D scene point, or the so called “correspondence problem”, arises. One way to bypass the correspondence problem is to actively control the scene illumination by projecting known patterns onto the objects. This approach is referred to as “active stereo vision” or “structured light”. One of the cameras in a stereo vision arrangement is replaced by a projector, or “active camera” [8].

6.1.1.4.2.1 Passive Stereo Vision

One of the most commonly used passive 3D systems is based on stereo imaging in which two cameras are used to capture two separate images of a scene from two different viewpoints. In such systems, the location and optical parameters of each separate camera must be calibrated so that triangulation methods (as seen in Figure 17) can be used to determine the correspondence between pixels in each image. Relative depth of points in the scene can then be computed since the depth of each point is inversely proportional to the difference in the distance of the corresponding points and their camera centres. This can be then used to generate a disparity map that visually provides 3D information. Because the positional and optical parameters of two separate cameras must be accurately calibrated, many products such as the Bumblebee series from Point Grey use dual cameras that are pre-calibrated, relieving the systems developer of such tasks. Disparity values obtained from stereo image pairs are directly proportional to the distance the cameras are apart and inversely proportional to the distance an object is away from the two cameras. Thus, in integrated stereo systems, increased disparity accuracy cannot be obtained by moving the cameras further apart but only by moving the dual camera system closer to the object being examined. It is important to note that it might not be possible to place the cameras close to the viewed scene in the collaborative robot safety project, this could cause unwanted inaccuracies [9].

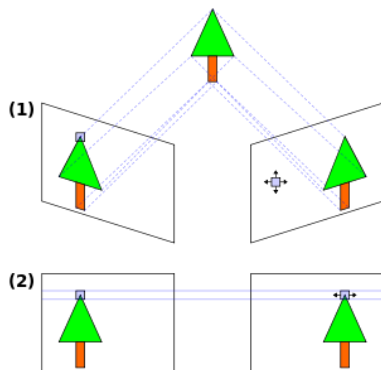


Figure 17 Stereo vision triangulation

6.1.1.4.2.2 Active Stereo Vision

The active stereo vision is a form of stereo vision which actively shines a light such as a laser or a structured light to simplify the stereo matching problem (as seen in Figure 18). In an active stereo vision system, one of the cameras is replaced with a projector or a laser unit, which projects onto the object of interest, a sheet of light at a time (or multiple sheets of light simultaneously). The idea is that once the perspective projection matrix of the camera and the equations of the planes containing the sheets of light relative to a global coordinate frame are computed from calibration, the triangulation for computing the 3D coordinates of object points simply involves finding the intersection of a ray from the camera and a plane from the sheet of light of the projector or laser [10].

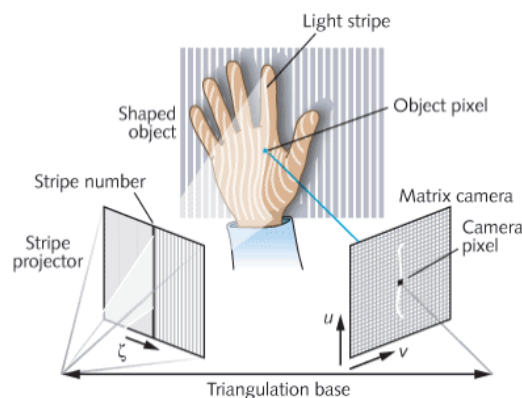


Figure 18 Structured light approach

6.1.1.4.2.3 Advantages and Disadvantages

An active stereo vision system has the following advantages:

- It deals with scenes that do not contain sufficient features, such as edges or corners, which are associated with intensity discontinuities, for the stereo matching process
- The correspondence problem is totally absent. For each pixel in the image that is illuminated by a particular sheet of light from the projector or laser unit, the plane equation of the sheet of light would have been computed from the projector calibration procedure, simple triangulation is only required to compute the 3D coordinates of the pixel.
- It gives a very dense depth or range map.
- It is applicable to many shape measurement and defect detection projects.
- For projector only systems: It is usable in many different environments/is not light depended like passive stereo vision.

Its shortcomings are:

- The system must be pre-calibrated. Unlike passive stereo vision systems with two cameras, there exists no self-calibration technique for recovering the geometry between the camera and the projector or laser unit.
- For projector and camera pairs: well-controlled lighting is required, such systems are therefore restricted to working only in indoor environments.
- The result will be in the form of a point cloud which takes a lot of processing power to create (dedicated GPU or FPGA needed). This will drastically slow down the safety system which makes it unsafe because the robot will not be able to react in time.
- Low maximum frame rate (≤ 30 Hz) and resolution ($\leq 640 \times 480$) with most systems.

6.1.1.4.3 Conclusions stereo vision

Passive stereo vision processing power requirements are similar to the 2D vision concept with three 2D cameras, and fall within this acceptable range. However, passive stereo vision makes use of two 2D cameras separated by a known distance and angle in one box, and then with the use of triangulation, the distance to a detected object is calculated. This strategy is very similar to the three 2D cameras concept, and does not add any additional benefits.

Active stereo vision can be used in a vast range of lighting environments and even in the dark. This will limit the image processing problem of constant lighting conditions. On the other hand, active stereo vision outputs a depth map, or point cloud from its obtained information. The processing power required for this is a lot higher than with passive stereo vision. The side effect of this is that one will see a drastic drop in frame rate (≤ 30 Hz) and/or a drop in resolution ($\leq 640 \times 480$) in order to be able to run this on a computer. These low frame rates and resolutions make it unsuitable for a safety system because the safety system needs to be real-time and able to detect small objects accurately.

6.1.1.4.4 Recommendations stereo vision

A solution to deal with the high amount of processing power required, is to use a dedicated GPU or FPGA for the image processing and creation of the point cloud.

6.1.1.5 Lidar

Lidar is a distance measuring method of using light in a similar fashion as using radio waves in radar. When measuring, a pulse of light is transmitted via laser. This transmitted light travels to an object and bounces back. The distance can either be calculated with the phase shift or with measuring the time transmitted light takes to bounce back to the sensor. $\text{Distance} = (\text{speed of light} \times \text{time of flight}) / 2$.

Because light travels fast, about 300,000 kilometres per second, the measuring equipment needs to operate extremely fast. So a scan cycle is extremely short and can be done at up to 150,000 pulses per second in some systems.

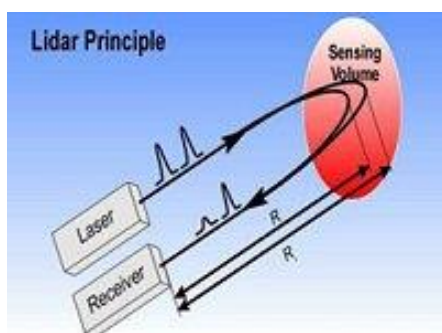


Figure 19 The lidar principle

6.1.1.5.1 Sensing area

Because the sensor itself only reads a 1 dimensional distance this isn't enough to scan an area or a plane. It is mostly placed on a rotating axis so that it can measure in a circle. With this, objects are detected in a 2D plane, position and distance.

When more distance measuring sensors are added or the rotating platform is translated or rotated over other axes than the measuring plane, it creates a 3D measuring volume. The shape of this volume depends on how the sensor is actuated.

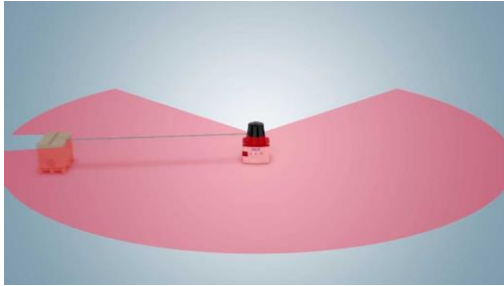


Figure 20 2D laser scanner

6.1.1.5.2 Point cloud

The resulting data from measurements is a cloud of measuring points, called a 'point cloud'. This point cloud can be used to interpret what the surroundings look like.

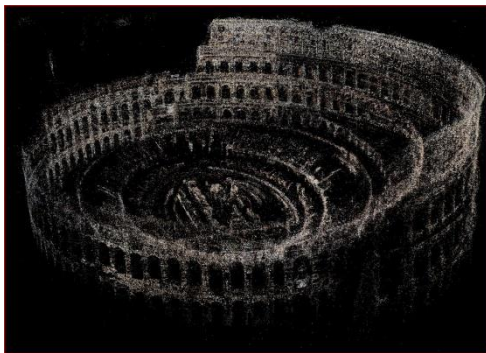


Figure 21 Point cloud

6.1.1.5.3 Advantages and Disadvantages

Advantages

- Very accurate measurement
- Fast measuring speed

Disadvantages

- Expensive technology
- Point cloud takes a lot of processing power

6.1.1.5.4 Conclusion Lidar

Lidar is a useful and millimetre accurate measuring technique for this project. It solves the need for constant light environments and is very fast. The down side is that lidar is very expensive, some quotes were requested and they ranged from 8,000 for a 2D scanner to 85,000 euro for a 3D scanner. On top of that the point cloud output by the system takes a lot of processing power to generate. Ultimately the price makes it not suitable for this project.

6.1.1.6 *Conclusion sensors research*

To solve the constant lighting conditions problem which arises from the 2D vision concept, research has been done on the different kind of sensors used in for example the self-driving car industry. This industry has been chosen because it as well as the collaborative robot safety project, have similar goals and restrictions relating to safety. The self-driving car industry uses sensors like Radar, Lidar, and stereo vision, however the newly obtained information shows that radar and lidar are currently very expensive and their sensing range is limited. Active stereo vision could potentially solve the constant lighting condition problem by using infrared structured light. Currently the resolution and frame rate must be drastically decreased when using this technology which does not make it suitable for safety, unless a dedicated GPU or FPGA is used. The processing power required to generate a depth map or point cloud, which is output by the active stereo vision, also limits the speed of the safety system and makes it not ideal for this project unless a dedicated GPU or FPGA is used.

6.1.1.7 *Recommendations sensors research*

It is highly recommended to use active stereo vision to detect obstacles and humans for this project since it simplifies the computer vision algorithms and deals with the problem of constant lighting conditions. When using active stereo vision an increase in processing power is required. This makes it unsuitable to be ran on standard computers. Therefore, it is recommended to use a dedicated GPU or FPGA to handle the resolution of the vision system at the desired framerate, to perform the computer vision calculations, and to generate the depth map or point cloud. There are special CPU's available which are designed to be connected to ROS so it can be integrated with the rest of the safety system.

6.1.2 *Sensor Concepts*

After having done research on sensors with which the group was not as familiar, to create a better understanding of their workings and possible implementations. A couple feasible concepts, which fall within the budget and time given for the project, have been set up. The active 3D vision is not listed below even though it has the most potential as a concept, because of time. There was not enough time left to order these camera's and look more into dedicated GPU's or FPGA's for the image processing.

6.1.2.1 *Different types of sensors*

The concepts described in the sections below all consist of three types of sensors. These sensors are 2D-cameras, laser scanners, or a sleeve around the robot arm. Each type of sensor has its own benefits and draw backs which are taken into consideration when coming up with a concept. The three different types of sensors are drawn in Figure 22 to provide an overview.

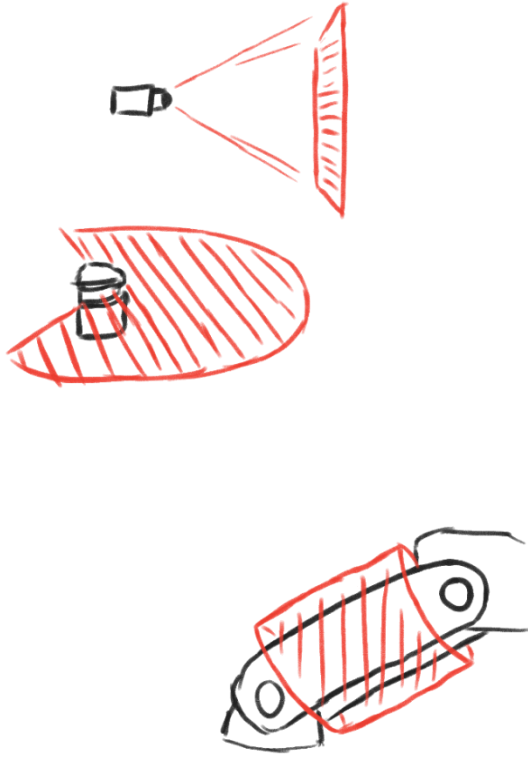


Figure 22 Three sensor concept types

6.1.2.2 2D-cameras and laser scanner

In this concept a 2D-camera is positioned above the work area of the robot for top down vision, and a laser scanner is positioned at the base of the robot to scan the horizontal axis, Figure 23. The advantage of this approach is that not much processing power is required to run the system but using only one camera also has its draw backs. By using only one camera, no height information of the objects can be obtained. This makes it impossible for the robot to adaptive plan a path which goes over or below the object to avoid it.

6.1.2.2.1 Specifications

- 2D Top Down vision
- Horizontal laser scan
- Two independent sensors of different types
- Object location and size, no height information
- Adaptive path planning possible
- Reasonable amount of processing power required

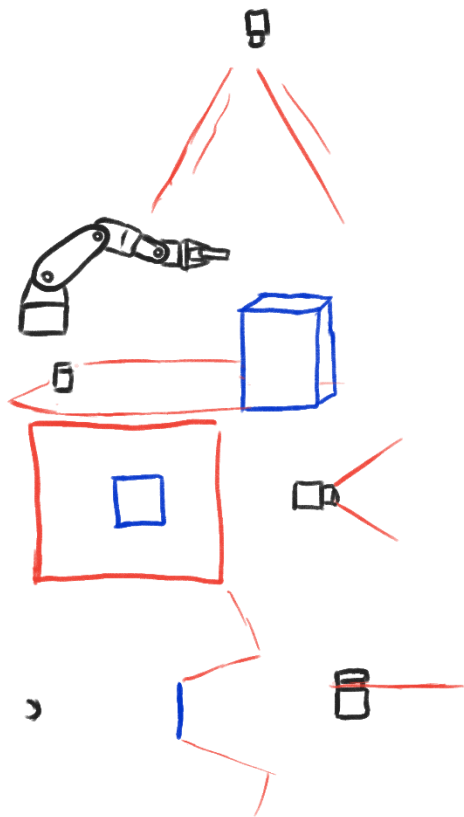


Figure 23 Two 3D-cameras and a laser scanner concept

6.1.2.3 Three 2D-cameras

This concept consists of three 2D-cameras positioned at three different axes, the X, Y, and Z-axis (Figure 24). These cameras will be linked together and will be able to output all the necessary 3D information of detected objects. This allows for precise adaptive path planning of the robot, and multiple zone control. The drawback of the concept is that it requires a lot of processing power. One could select grey scale images to reduce the required processing power.

6.1.2.3.1 Specifications

- 2D vision on all three axes
- Three independent sensors, although same sensor type
- Object location size and height
- Adaptive path planning possible
- Planning over objects possible with large enough robot
- High amount of processing power required

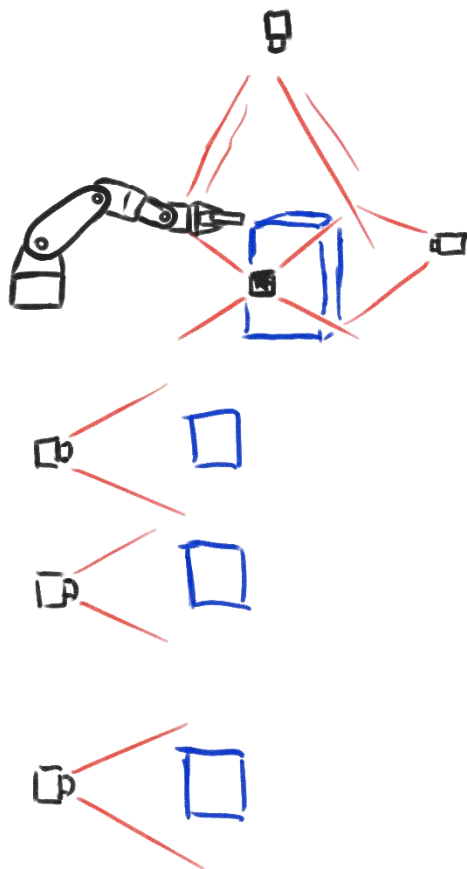


Figure 24 Three 2D-cameras

6.1.2.4 One 2D-camera

The concept seen below consists of one 2D camera which is located above the work area of the robot for a top down view, Figure 25. The advantage of this approach is that not much processing power is required to run the system but using only one camera also has its draw backs. By using only one camera, no height information of the objects can be obtained. This makes it impossible for the robot to adaptive plan a path which goes over or below the object to avoid it.

6.1.2.4.1 Specifications

- Single 2D camera in top down view
- Only location and size data, no height
- Lowest amount of processing power required
- Adaptive planning harder

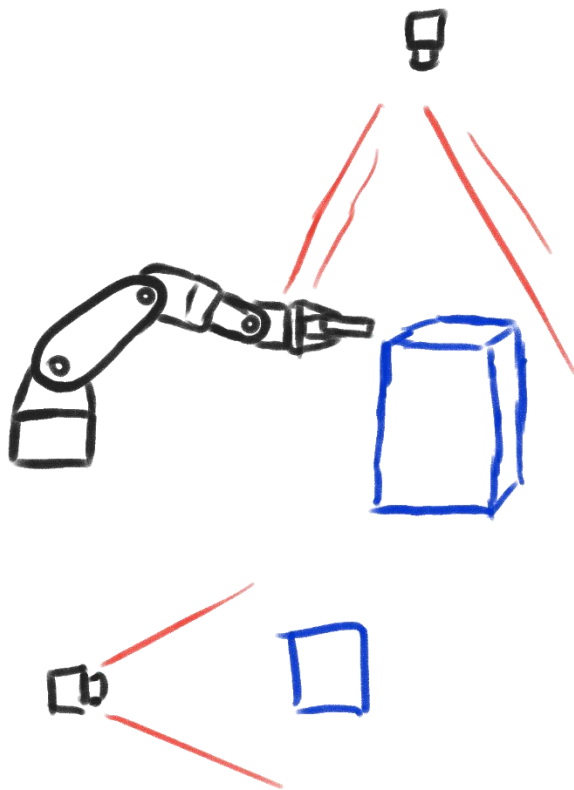


Figure 25 One 2D-camera concept

6.1.2.5 Sleeve

Bosch invented the APAS, this is a capacitive sleeve for around their robot which can detect a human within a range of 50mm. The sleeve then sends out a signal to the controller which stops the robot until the human moves further away again. Currently there are some EMC problem with this product. The product doesn't function properly around other industrial devices.

The idea is to create a similar skin which can be implemented on multiple robots, not only the Bosch APAS (Figure 26). This will add an extra layer of safety to the collaborative robot safety system of this project.

6.1.2.5.1 Specifications

- Capacitive sensor skin
- Detection range: 50mm
- Usable to detect near contact with humans

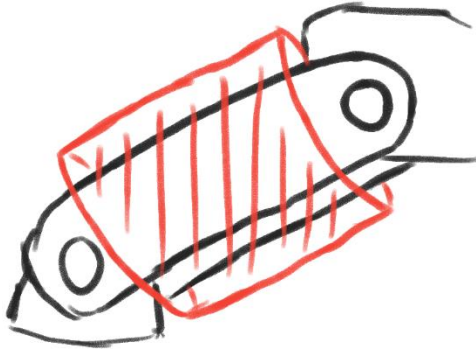


Figure 26 Sleeve concept

6.1.2.6 Conclusion sensor concepts

The final chosen concept which will be worked out in detail is the three 2D-camera concept. The biggest advantage of this concept is that it outputs 3D information of an object located in the work area of the robot. This allows for adaptive path planning. The camera from above also makes it possible to work with zones to slow the robot down if a worker is detected in the work area of the robot. The biggest disadvantage of this concept is that since it will use three 2D cameras, it requires a lot of image processing compared to alternative methods/concepts like the laser scanner or 3D vision. The choice has been made to use grayscale 2D cameras to reduce the required processing power.

6.1.2.7 Recommendations sensor concepts

The capacitive sleeve is not used in any of the concepts because research has proven that it is not realizable in the time frame for our project and is not as modular as other concepts. The sleeve is recommended as an additional safety measure for the system which can be added later if available.

Active 3D vision using projected/structured light is advised to use in a follow up on this project, because 3D vision already takes care of the 3D information and makes it easier to filter out background noise. The downside is that it requires a dedicated GPU or FPGA for image processing and the creating of the depth map/point cloud. Time limited the group to look more into dedicated GPU's or FPGA's and work this option out.

6.1.3 Sensor Specifications / Sensor Elaboration

After having chosen to work with the three 2D camera concept on a IRB120 Robotic arm, a specific camera which satisfies the requirements of the project needs to be picked. In the following chapter calculations have been made and decisions are explained in detail with respect to processing power, resolution, framerate, and focal length required by the camera.

6.1.3.1 Required resolution

The collaborative robot safety project is required to be able to detect a human pinkie finger. A human pinky finger is on average 15mm wide, therefore it should be able to be detected using a VGA resolution camera, as this resolution provides a 3mm/px resolution. This means that the minimum resolution for the cameras needs to be 640x480 pixels as can be seen in table 15.

The robot (IRB120) has a reach of 580mm. Therefore, the minimum width of the detection area should be 1160mm. In the calculations below a detection area of 1500 by 1500mm is used from a height of 3 meters.

Resolution	X pixels	Y pixels	X width area [mm]	Y width area [mm]	X resolution [mm/px]	Y resolution [mm/px]
VGA	640	480	1500	1500	2,34375	3,125
XGA	1024	768	1500	1500	1,46484375	1,953125
SXGA	1280	1024	1500	1500	1,171875	1,46484375

Table 14: Millimetre per Pixel resolution calculation for field of view

6.1.3.2 Data calculations

Based on the minimum needed resolution (640x480 pixels) and grayscale cameras which are used in order to reduce processing power, calculations have been made on the amount of data output by the system. The total amount of data which the system needs to be able to process purely based on the output of the cameras, without including the image processing, is 54 MB/s as can be seen in the calculations below.

Calculations processing data:

VGA resolution: 640x480 pixels = 307200 px

Monochrome image: 8 bit grayscale => 1 byte per pixel

1 frame: 307200 bytes => 0,3MB

60 fps => 18MB/s per camera

Three cameras are used so: 3*18MB/s => 54 MB/s

6.1.3.3 Focal length calculations

In order to see a work area of 1.5x1.5 meter from a distance of 3 meter to the camera, and a predetermined image sensor size based on availability, a focal length needs to be calculated. The focal length determines the area which we can view with the camera and its sensor size. The cameras which we looked at all could work with either an 1/4 or 1/3 inch sensor.

$$AFOV(^{\circ}) = 2 \times \tan^{-1} \left(\frac{\text{Horizontal FOV (mm)}}{2 \times WD \text{ (mm)}} \right)$$

or

$$\text{Horizontal FOV (mm)} = 2 \times WD \text{ (mm)} \times \tan \left(\frac{AFOV(^{\circ})}{2} \right)$$

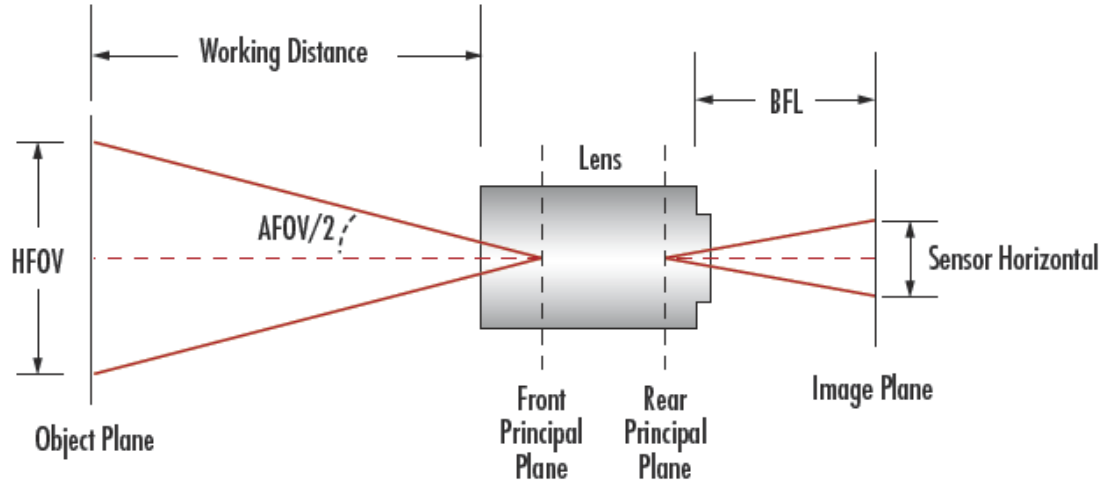


Figure 26: Field of view and focal length equations

Using the horizontal FOV equation above, and in order to see an area of 1.5x1.5 meter from a distance of 3 meter to the camera, with a sensor size of 1/4 inch, a focal length of 7.37mm is needed. The closed available/standard size is 8mm.

Using the horizontal FOV equation above, and in order to see an area of 1.5x1.5 meter from a distance of 3 meter to the camera, with a sensor size of 1/3 inch, a focal length of 9.6mm is needed. The closed available/standard size is 12mm.

6.1.3.4 Frame rate

Using Table 15 the minimum required frame rate to detect a pre-set distance that the robot and human cover per frame can be looked up. So for example, when it's needed to be able to detect at least 8mm change of distance per frame by the robot and human combined travelling at 250mm/s and 500mm/s respectively, would be 93,75 frames per second.

Robot velocity [mm/s]	Human velocity [mm/s]	Distance per frame [mm]	Frames per second
250	500	8	93,75
500	500	8	125,00
750	500	8	156,25
1000	500	8	187,50
4000	500	15	300,00
4250	500	15	316,67
4500	500	15	333,33
4750	500	20	262,50
5000	500	20	275,00
5250	500	20	287,50
5500	500	20	300,00
5750	500	20	312,50
6000	500	20	325,00
6250	500	25	270,00
6500	500	25	280,00
6750	500	25	290,00
7000	500	25	300,00

Table 15 Framerate calculation for given motion velocities and allow distance per frame.

6.1.3.5 Distance per frame

In Table 16 the distance covered per frame based on a pre-set frame rate and the speed of the robot and human can be looked up. For example when using a framerate of 25 frames per second, and the robot and human are moving at 250mm/s and 500mm/s respectively, there will be a change of 30mm per frame.

Frames per second	Robot velocity [mm/s]	Human velocity [mm/s]	Distance per frame [mm]
1	250	500	750,00
2	250	500	375,00
3	250	500	250,00
4	250	500	187,50
5	250	500	150,00
6	250	500	125,00
7	250	500	107,14
8	250	500	93,75
9	250	500	83,33
10	250	500	75,00
11	250	500	68,18
12	250	500	62,50
13	250	500	57,69
14	250	500	53,57
15	250	500	50,00
16	250	500	46,88
17	250	500	44,12
18	250	500	41,67
19	250	500	39,47
20	250	500	37,50
21	250	500	35,71
22	250	500	34,09
23	250	500	32,61
24	250	500	31,25
25	250	500	30,00
205	250	500	3,66
210	250	500	3,57
215	250	500	3,49
220	250	500	3,41
225	250	500	3,33
230	250	500	3,26
235	250	500	3,19
240	250	500	3,13
245	250	500	3,06
250	250	500	3,00
255	250	500	2,94
260	250	500	2,88
265	250	500	2,83
270	250	500	2,78
275	250	500	2,73
280	250	500	2,68
285	250	500	2,63
290	250	500	2,59
295	250	500	2,54
300	250	500	2,50

Table 16 Movement per frame calculation, at given framerate and motion velocities

6.1.3.6 Conclusions sensor elaboration

Based on the calculations made in this chapter and the safety guidelines provided by the ISO standards for collaborative robot safety, the minimum frame rate of the cameras is 30 frames per second. This follows from the fact that the ISO safety standards allow the collaborative robot to move at 250 mm/s and indicate that a human moves on average with a speed of 500mm/s. This will result in a detection possibility of 25mm per frame. This is enough to adaptively plan the path of the robot or ensure safety by stopping the movement of the robot and avoid collision.

The minimum resolution required by the camera to detect the smallest object from a distance of 3 meter is 640x480 pixels. In order to see a work area of 1.5x1.5 meter from a distance of 3 meter from the camera with a sensor size of 1/4 inch of 1/3 inch is 8mm and 12mm respectively.

When using 30 frames per second and three grayscale cameras, the system needs to be able to process at least 54MB/s of image data. On top of this the system needs to do image processing to detect objects, people, and adaptively plan the path of the robotic arm.

6.1.3.7 Recommendations sensor elaboration

It is advised to switch to colour and structured light 3D cameras instead of 2D grayscale cameras for better results. When 3D cameras are used, a lot more data will need to be processed which will be too much for a standard computer to handle. This adds the need of a dedicated GPU or FPGA to do the calculations and processing.

6.2 Software

The safety system has been built using the Robot Operating System, or ROS. This system is a software stack on top of Ubuntu Linux, and allows for rapid prototyping of robot control systems. ROS contains software to process camera data, perform path planning for robots and perform inverse kinematic calculations for robot arms. New software can be added to the ROS system in the form of nodes, which can communicate with the core ROS system. The communication with the ROS core uses topics, to which each node can subscribe or publish.

For the safety system, a number of custom nodes have been developed. For each camera a node has been designed to process the image data, and determine the closes bounding box for each detected object. The code for these nodes can be found in Appendix 4. The data output by these nodes is used to create collision object in the planning world of the robot. This node can be found in Place object.

When only a single camera is in use, the objects will be placed on a two dimensional grid and a fixed height in

three dimensional space. When a second camera is added to the system, and this camera is aligned to the reference, the system places three dimensional objects in the three dimensional space. The code for the three dimensional placement can be found in Place 3D object. This data is used by the robot planner, MoveIt, to determine a motion path for the robot, when possible.

Figure 27 shows the communication between the different nodes for the safety system and demonstrator. For each camera a ROS driver publishes the image data to a topic, to which the processing nodes subscribe. These nodes then publish the output of their processing to a new topic, to which other nodes subscribe. The final node that determines the object size and position publishes this data to the collision object topic of the planner.

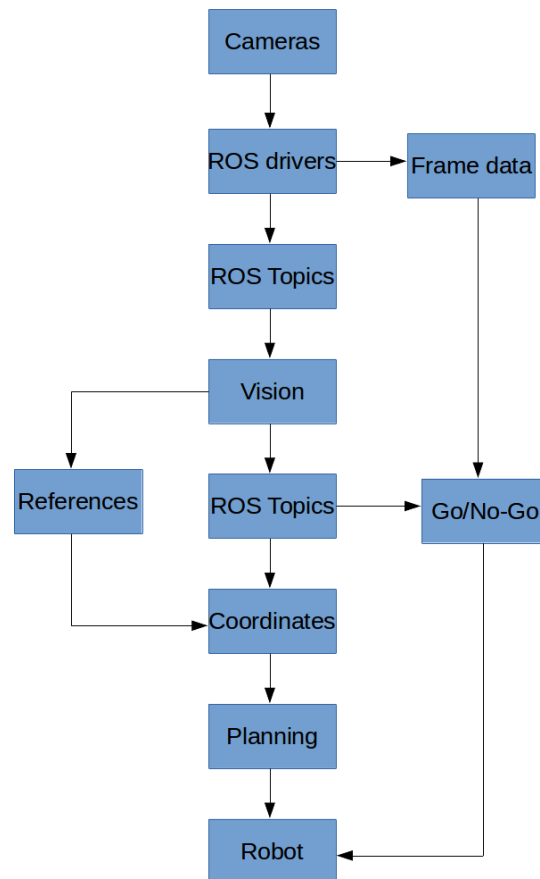


Figure 27 Safety system flowchart

For testing purposes, a node has been written that provides Cartesian coordinates for the robot end effector. The code for this node for both the ABB IRB120 and IRB140 robots can be found in Move test ABB IRB120 and Move test ABB IRB140. Using these coordinates, MoveIt calculates a motion path. When no collisions in the path have been found, the motion path is send to the robot controller and executed by the robot arm.

6.3 Vision

A proof of concept has been built using a Matrix Vision mvBlueFOX camera, equipped with a 12 mm lens as the top down camera. The nodes that process the image data use the OpenCV stack. OpenCV provides functions to implement computer vision. The safety system uses filtering functions from OpenCV to process the images, and applies an OpenCV function to detect objects.

A custom node has been developed that subscribes to the image topic of a camera, and receives the image frames from this topic. For each frame, first a Gaussian Blur is used to reduce the visibility of the shadows cast by the different objects in the frame. Next a threshold function is applied, producing a black and white image, where the white areas are the areas of interest. Using this image, a contour finding function is applied, which results in an array of contours. On each of these contours a closes bounding box function is applied, to produce a rectangle describing the best fitting representation of the detected contours. Figure 28 shows the threshold stage and the box overlaid on the original image. The centre point, rotation and size of these boxes is then published for further processing.

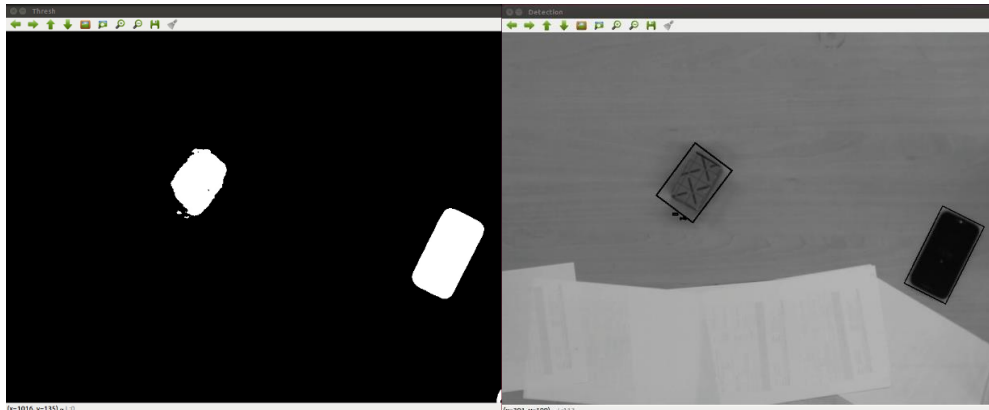


Figure 28 Left: Thresholded image showing objects of interest. Right: Original image with bounding boxes overlaid.

When the detection software starts, the first frame is used to determine the location of a QR code. This code is placed on a known distance from the robot origin in the real world, and is of a known size. Figure 29 shows an image with both the QR code reference, and detected objects enclosed by the bounding box. Using the centre coordinates from this code, the software is able to position each detected object in the image frame relative to the QR code. Since the size of the QR code is known, the resolution of the camera, in millimetre per pixel, can be automatically calculated. Using this information, the system is able to position the object in their relative position, in meters, to the robot world origin, and can reliably determine the real world object size in meters. To avoid recognizing the QR code as objects, the software takes a snapshot of this first frame, and subtracts it from all following frames to remove the background.

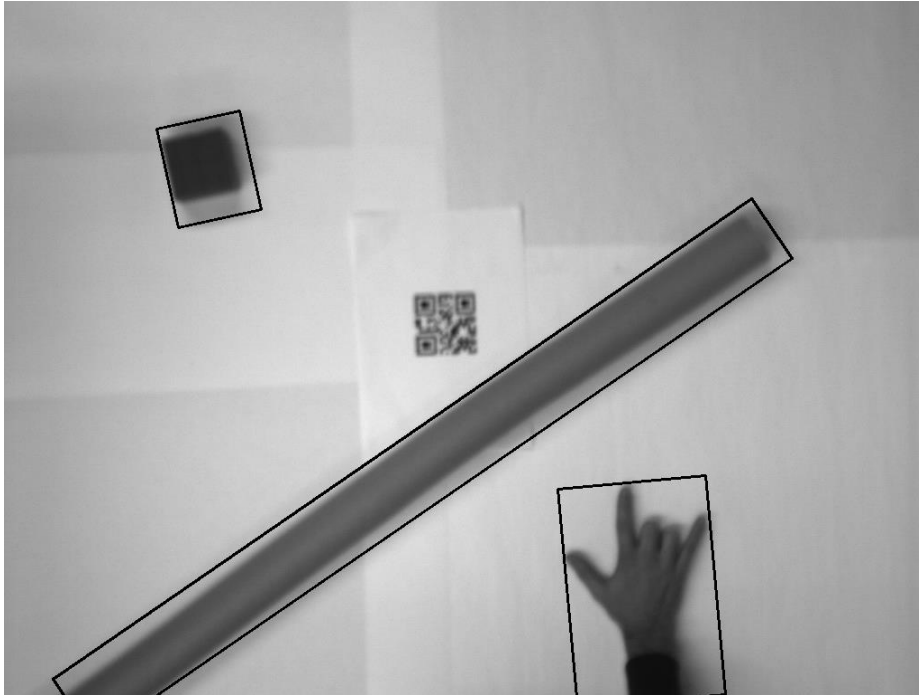


Figure 29 Box overlay on live image, QR code reference

6.4 Three dimensional positioning

As the concept for the safety system includes multiple cameras, the system should be capable of positioning the objects in three dimensional space. For each camera the vision software uses the QR code as a reference. By placing these codes on two world axis, in a known position and orientation, both cameras have a shared reference zero point. For each object detected by the cameras, position, rotation and size are published to separate topics. A node developed to combine this data into correct three dimensional objects subscribes to these topics and receives the data. By checking for the matching coordinate, as for example two cameras share the x axis, the objects xyz coordinate can be determined. This data matches the real world object and is placed in the real world location in the planning world.

6.5 Testing

Testing the safety system showed multiple problems. The following statements are based on observations during the basic functionality testing of the system. These tests did not follow the test plan as described in Appendix 3.

First using grayscale image creates difficulty in detecting objects on a non-white background. Because in grayscale multiple colours can have the same grayscale, it becomes hard to distinguish between objects. For example the floor in the mechatronics lab has a green tint. Due to this tint the floor has been covered with white paper, as is visible in Figure 29, to have all objects above it stand out from the background.

Second, ROS is not a safe system. The different parts of the ROS system crashed multiple times during the tests. For example the planner used to move the robot based on the data from the cameras crashed when the software could not determine a viable path.

Testing also showed that ROS is not real-time. This was known at the beginning of development, but proved problematic. Because ROS is not real-time, data can be processed too late and could provide an inaccurate representation of the world around the robot.

Next all the planners in ROS are offline. This means that the planner receives a representation of the world at a specific moment, plans a path using this information, and attempts execution. In context, an online planner uses a constantly updated representation of the world. This provided a problem when the world around the robot changed, but this was not used by the planner, as path execution had already started.

Furthermore, using only two dimensional cameras provided difficulty for the planning software as depth perception is not possible from a single video input. This has been attempted to resolve using three dimensional reconstruction, but it has not been completed at the moment of writing.

6.6 Conclusions vision system

For the safety system the vision system with three cameras has been selected. These cameras are placed on the three world axes and the data is used to recreate a three dimensional world. These cameras require a minimal framerate of 90 frames per second and a field of view of three by three meters. The cameras can be grayscale or colour cameras with a minimal resolution of 640x480 pixels.

The demonstrator has been built using a single camera looking down on the work area of the robot. A second camera looking from a side will be added.

The current system, using a top down camera with a resolution of 1024x768 pixels and an 12mm lens, has been built, and has had very preliminary testing to confirm the basic concept is working. From these tests no usable results have been achieved as the system has not yet been tested according to the test document in Appendix 3.

The system with a second camera has been attempted without calculating the object dimensions and location in the world. In this case a webcam running at 640x480 pixels was used. Both cameras had a shared reference in this setup. Both programs output objects with the same coordinates which showed a strong likelihood of these calculations being able to place the objects with the correct dimensions in 3D space, but need a much more complex algorithm to match the data than originally thought.

Preliminary testing showed that using only grayscale images provided extra difficulty in detecting objects as two colours can have the same grayscale colour. It also showed that using only computer vision it is not possible to eliminate all blind spots in the environment. This could be achieved by adding different sensors to the system, for example radar or depth vision.

Based on the tests with ROS, the system is notably not real time or online. This causes problems when the environment changes around the robot, as updates to the representation of the world are delayed causing the planner to miss objects in the world.

6.7 Recommendations vision system

Use multiple different sensors, for example use not only two dimensional cameras but a system consisting of a 2D camera, a 3D camera and a volumetric radar.

Use a different setup than the ROS system. Due to non-real-time and offline planning problems. This does mean custom software has to be developed.

Use a different technique to get a three dimensional representation of the world around the robot.

7 Conclusions

A safety system has been developed in order to make a robotic arm operate safely amongst human workers. The system consists of three safety measures:

1. Padding, serving as impact damping to prevent harmful collision with a person;
2. Covers to reduce the risk of body parts, clothes, hair or jewellery getting stuck, entangled or crushed in any gaps in the robot;
3. A vision system, used to adapt the speed of the robotic arm, depending on the distance to a person.

Using the above safety measures, all risks as defined by the risk assessment can be reduced to acceptable risks, which is defined as the lowest possible risk. Except for the risk corresponding to hazard number 1.4: crushing a person. This risk can only be reduced to a possible risk, which is considered to be less safe than an acceptable risk.

A System Requirements Document is part of this report. To date, some requirements have only been tested preliminary. However, a proper test document is also part of this report.

A demonstrator is built to serve as a proof of concept. The robot used for the demonstrator is the ABB IRB 140.

Padding with a density of $0.23 \left[\frac{g}{cm^3} \right]$ and a compression distance of $5 [mm]$ is sufficient to prevent harmful injury when colliding with a person, using the ABB IRB 140 at a maximum speed of $0.4 [m/s]$.

The cover that is designed for demonstration purposes is a radius which covers the fourth joint of the ABB IRB 120 robot. This robot is not the robot that is used as a demonstrator in the current project, but the ABB IRB 120 is the robot that Fontys owns and will use in future projects. The radius is produced with a 3d-printer using ABS as filament.

The radius is easy to produce and customise (e.g. using a 3d-printer) as well as easy to mount. The cover does not require extra mounting points since it can be clamped on the robotic arm. The cover does not obstruct the movement of the robot, but it increases its volume. Small gaps may still occur.

For the vision system a concept with three cameras has been selected. These cameras should be placed on a 90 degree angle with respect to each other. A minimal framerate of 90 frames per second and a field of view of three by three meters is required. The cameras can be grayscale or colour cameras with a minimal resolution of 640x480 pixels.

A top down camera with a resolution of 1024x768 pixels and a 12mm lens, has been built for the demonstrator.

Preliminary testing showed that using only grayscale images provided extra difficulty in detecting objects as two colours can have the same grayscale colour. It also showed that using only computer vision it is not possible to eliminate all blind spots in the environment. This could be achieved by adding different sensors to the system, for example radar or depth vision.

Based on the preliminary tests with ROS, the system does not meet the real-time requirements. This causes problems when the environment changes around the robot, as the changes to the robot world can be delayed causing the planner to not use the latest representation of the world.

8 Recommendations

Recommendations are based on the conclusions, regarding:

1. The risk assessment;
 - a. Research the possibility to put the robot on a non-actuated movable base (e.g. a base on wheels, or by adding springs/dampers) to reduce the risk of hazard no. 1.4 (crushing a person between the robot and a foreign object) by creating more degrees of freedom. It can be advised to make a difference between crushing on a foreign (e.g. workpiece of wall) object and crushing on the robot, or the robot base.
2. Padding:
 - a. Due to lack of publicly available documentation, it is advised to test different foam samples for impact to determine materials best suited for the specifications according to this document;
 - b. It is furthermore recommended to consult human injury biomechanics to further optimize specifications for impact reducing materials.
 - c. Ideally the robot itself should be built using a less stiff (weak) exterior.
3. Covers:
 - a. Further research options to cover (fingertip size or smaller) gaps.
 - b. Ideally the shape of the robot should intrinsically be safe. Meaning not containing (small) gaps and unable to crush limbs. Current co-bots on the market, e.g. UR-5, serve as good examples.
4. The vision system:
 - a. Use multiple different sensors, for example use not only two dimensional cameras but a system consisting of a 2D camera, a 3D camera, and a volumetric radar;
 - b. Use a different setup than the ROS system. The system should be real time and capable of handling the amount of data generated by the sensors. Due to non-real-time and offline planning problems. This does mean custom software should be developed;
 - c. Use a different technique to get a three-dimensional representation of the world around the robot, instead of attempting to rebuild the world from a set of 2 dimensional images.

Bibliography

- [1] ISO, "ISO/TS 15066 Robots and robotic devices - Collaborative robots," NEN, 2016.
- [2] S. H. Goods, C. L. Neuschwanger, C. Henderson and D. M. Skala, "Mechanical Properties and Energy Absorption Characteristics of a Polyurethane Foam," 1997.
- [3] C. Wolff, "2D or 3D radar," radiotutorial , [Online]. Available: <http://www.radartutorial.eu/02.basics/2D%20or%203D%20radar.en.html>. [Accessed 19 12 2016].
- [4] M. Kishida, K. Ohguchi and M. Shono, "70 GHz-Band High-Resolution Millimeter-Wave Radar," Fujitsu, October 2015. [Online]. Available: <https://www.fujitsu.com/global/documents/about/resources/publications/fstj/archives/vol51-4/paper09.pdf>. [Accessed 16 December 2016].
- [5] K. Shirakawa, S. Kobashi, Y. Kurono, M. Shono and O. Isaji, "3D-Scan Millimeter-Wave Radar for Automotive Application," Fujitsu, 2013. [Online]. Available: <http://www.fujitsu-ten.com/business/technicaljournal/pdf/38-1.pdf>. [Accessed 16 December 2016].
- [6] "Soli," Google, October 2016. [Online]. Available: <https://atap.google.com/soli/>. [Accessed 16 December 2016].
- [7] G. Bradski and A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, Sebastopol: O'Reilly Media, 2008.
- [8] "Computer stereo vision," Wikipedia, 22 November 2016. [Online]. Available: https://en.wikipedia.org/wiki/Computer_stereo_vision. [Accessed 16 December 2016].
- [9] G. Bianco, A. Gallo, F. Bruno and M. Muzzupappa, "A COMPARISON BETWEEN ACTIVE AND PASSIVE TECHNIQUES FOR UNDERWATER 3D APPLICATIONS," University of Calabria, 2 March 2011. [Online]. Available: <http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XXXVIII-5-W16/357/2011/isprsarchives-XXXVIII-5-W16-357-2011.pdf>. [Accessed 16 December 2016].
- [10] R. Owens, "Active stereo," CVonline, 29 October 2008. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT11/node9.html. [Accessed 16 December 2016].
- [11] ISO, "ISO 12100 Safety of machinery - General principles for design - Risk assessment and risk reduction," NEN, 2010.


Appendix 1 Hazards, risks and risk reduction methods


All hazards, according to §2.3, with corresponding risks and risk reduction methods can be found in this appendix.

Risks are classified according to Table 17.

Table 17 Risk classification table


Risk classification table		
Very High Risk = Stop	$R > 5000$	
High Risk = Immediate Measures	$2000 < R < 5000$	
Important Risk = Short-term Measures	$500 < R < 2000$	
Possible Risk = Long-term Measures	$100 < R < 500$	
Acceptable Risk = No Measures	$R < 100$	

Hazard Identification		Hazard No.	1.1
Title	Touch a worker		
Location	Shared workspace		
Target	Human eye		
Activity	Normal operation in shared workspace		
Hazard Type	Mechanical		
Consequence	Impact		
Description	The robot could accidentally touch a worker		
References:	ISO 10218-1		
Risk Estimation and Evaluation			
Severity	100	Exposure time	10
Probability	1	Risk (S x E x P)	1000
Important risk			
Risk Reduction			Reference
Cover payload			
Risk Estimation and Evaluation			
Severity	1.5	Exposure time	10
Probability	1	Risk (S x E x P)	15
Acceptable risk			


Hazard Identification			Hazard No.	1.2
Title	Run into worker			
Location	Shared workspace			
Target	Head (temple)			
Activity	Normal operation in shared workspace			
Hazard Type	Mechanical			
Consequence	Impact			
Description	The robot could accidently run into a worker			
References:	ISO 10218-1			
Risk Estimation and Evaluation				
Severity	200	Exposure time	10	
Probability	10	Risk (S x E x P)	20000	
Very high risk				
Risk Reduction			Reference	
Add a padding on the robot to reduce impact			ISO TS 15066 [5.5.5.4 b1]	
Risk Estimation and Evaluation				
Severity	37.5	Exposure time	10	
Probability	10	Risk (S x E x P)	3750	
High risk				
Risk Reduction			Reference	
Adapting speed depending on distance to human			ISO 10218-1 [5.6.2]	
Risk Estimation and Evaluation				
Severity	6.25	Exposure time	10	
Probability	3	Risk (S x E x P)	187.5	
Possible risk				


Hazard Identification			Hazard No.	1.3
Title	Limbs get stuck			
Location	Robot mechanical structure with holes			
Target	Limbs, arm			

Activity	Normal operation in shared workspace		
Hazard Type	Mechanical		
Consequence	Crushing, entrapment		
Description	Workers limbs could get stuck in the robot		
References:	ISO 10218-1		
Risk Estimation and Evaluation			
Severity	37.5	Exposure time	10
Probability	6	Risk (S x E x P)	2250
High risk			
Risk Reduction			Reference
Cover the robot so body parts can't get stuck in the robot.			ISO TS 15066 [5.5.5.4 a3]
Risk Estimation and Evaluation			
Severity	37.5	Exposure time	10
Probability	0.2	Risk (S x E x P)	75
Acceptable risk			


Hazard Identification			Hazard No.	1.4
Title	Human crushed			
Location	On a heavy obstacle (e.g. wall or workpiece)			
Target	Head (temple)			
Activity	Normal operation in shared workspace			
Hazard Type	Mechanical			
Consequence	crushing			
Description	Worker could get crushed between the robot and the workbench or workpiece			
References:	ISO 10218-1			
Risk Estimation and Evaluation				
Severity	200	Exposure time	10	
Probability	3	Risk (S x E x P)	6000	
Very high risk				
Risk Reduction			Reference	
Adapting speed depending on distance to human			ISO 10218-1 [5.6.2]	

Risk Estimation and Evaluation			
Severity	200	Exposure time	10
Probability	1	Risk (S x E x P)	2000
High risk			
Risk Reduction			Reference
Measure the force feedback			
Risk Estimation and Evaluation			
Severity	37.5	Exposure time	10
Probability	1	Risk (S x E x P)	375
Possible risk			


Hazard Identification		Hazard No.	1.5
Title	Human pinned		
Location	On a heavy obstacle (e.g. wall or workpiece)		
Target	Head (temple)		
Activity	Normal operation in shared workspace		
Hazard Type	Mechanical		
Consequence	Trapping		
Description	Worker could get pinned down between the robot and walls or other objects		
References:	ISO 10218-1		
Risk Estimation and Evaluation			
Severity	1.5	Exposure time	10
Probability	1	Risk (S x E x P)	15
Acceptable risk			

Hazard Identification			Hazard No.	1.6
Title	Human hair entangled			
Location	Shared workspace			
Target	Human hair			


Activity	Normal operation in shared workspace		
Hazard Type	Mechanical		
Consequence	Entanglement, Drawing-in		
Description	A worker’s hair could get entangled in the robot arm		
References:	ISO 10218-1		
Risk Estimation and Evaluation			
Severity	22	Exposure time	10
Probability	6	Risk (S x E x P)	1320
Important risk			
Risk Reduction			Reference
Cover the robot so body parts can't get stuck in the robot.			ISO TS 15066 [5.5.5.4 a3]
Risk Estimation and Evaluation			
Severity	22	Exposure time	10
Probability	1	Risk (S x E x P)	220
Possible risk			
Risk Reduction			Reference
Adapting speed depending on distance to human			ISO 10218-1 [5.6.2]
Risk Estimation and Evaluation			
Severity	6.25	Exposure time	10
Probability	1	Risk (S x E x P)	62.5
Acceptable risk			

Hazard Identification		Hazard No.	1.7
Title	Clothes entangled		
Location	Robot joints		
Target	Clothes (sleeve, hood)		
Activity	Normal operation in shared workspace		
Hazard Type	Mechanical		
Consequence	Entanglement, Drawing-in		
Description	A worker's loose clothing could get entangled in the robot arm		

References:	ISO 10218-1		
Risk Estimation and Evaluation			
Severity	22	Exposure time	10
Probability	3	Risk (S x E x P)	660
Important risk			
Risk Reduction			Reference
Cover the robot so clothing can't get stuck in the robot.			ISO TS 15066 [5.5.5.4 a3]
Risk Estimation and Evaluation			
Severity	22	Exposure time	10
Probability	1	Risk (S x E x P)	220
Possible Risk			
Risk Reduction			Reference
Adapting speed depending on distance to human			ISO 10218-1 [5.6.2]
Risk Estimation and Evaluation			
Severity	6.25	Exposure time	10
Probability	1	Risk (S x E x P)	62.5
Acceptable risk			

Hazard Identification		Hazard No.	1.8
Title	Jewellery entangled		
Location	Robot joints		
Target	Jewellery (fingers, neck and arms)		
Activity	Normal operation in shared workspace		
Hazard Type	Mechanical		
Consequence	Entanglement, Drawing-in		
Description	A worker's jewellery could get entangled in the robot arm		
References:	ISO 10218-1		
Risk Estimation and Evaluation			
Severity	37.5	Exposure time	10

Probability	1	Risk (S x E x P)	375
Possible risk			
Risk Reduction			Reference
Adapting speed depending on distance to human			ISO 10218-1 [5.6.2]
Risk Estimation and Evaluation			
Severity	6.25	Exposure time	10
Probability	1	Risk (S x E x P)	62.5
Acceptable risk			

Hazard Identification		Hazard No.	1.9
Title	Push workpiece		
Location	Shared workspace and surrounding area		
Target	Lower body		
Activity	Normal operation in shared workspace		
Hazard Type	Mechanical		
Consequence	crushing		
Description	The robot could push a heavy workpiece, which makes it fall on workers around the workspace of the robot.		
References:	ISO 10218-1		
Risk Estimation and Evaluation			
Severity	100	Exposure time	10
Probability	6	Risk (S x E x P)	6000
Very high risk			
Risk Reduction			Reference
Adapting speed depending on distance to workpiece			ISO 10218-1 [5.6.2]
Risk Estimation and Evaluation			
Severity	100	Exposure time	10
Probability	0.5	Risk (S x E x P)	500
Important risk			
Risk Reduction			Reference
Use force feedback			

Risk Estimation and Evaluation			
Severity	100	Exposure time	10
Probability	0.1	Risk (S x E x P)	10
Acceptable risk			

Appendix 2 Requirements

The requirements are categorized and listed based on the risk reduction processes which include reduction by intrinsic design, safeguards, and user information.

Each requirement is colour coded to indicate the priority of the requirement according to the MoSCoW method.

- **Must**
- **Should**
- **Could**
- **Won't**

Appendix 2.1 By intrinsic design

Requirement	Date	Status	Modified	Name
1.1	24-09-2016	Submitted	30-11-2016	Self-check
	Description			
	The safety system must perform a self-check on start. If the system does not pass its self-check, the robot must not power on and send an error message to the user.			
	Motivation			
	People rely on the safety system and assume it works at all times.			
	Test			
	Introduce a fault into the safety system.			

8.1.1

Requirement	Date	Status	Modified	Name
1.2	24-09-2016	Submitted	-	EMC directives
	Description			
	The safety system must be compliant with EMC directives. EMC directive 2014/30/EU			
	Motivation			
	The system cannot cause interference or be subjected to interference.			
	Test			
	Test the safety system in an EMC environment.			

Requirement	Date	Status	Modified	Name
1.3	30-11-2016	Submitted	-	Modularity, sensors
	Description			
	The safety system must be usable even if the total system is moved to a different location, but still using the same configuration as specified.			
	Motivation			
	The goal is to design a modular system. The safety system should be movable and installed in different locations such as the Mechatronics lab at Fontys, but also at van Beek.			
	Test			
	Move sensor of safety system to a different location using the same configuration.			

Requirement	Date	Status	Modified	Name
-------------	------	--------	----------	------

1.4	30-11-2016	Submitted	-	Modularity, sensors
	Description			
	The safety system must be usable even if the sensors are applied to a different robotic arm.			
	Motivation			
	The goal is to design a modular system. The safety system should be applicable on multiple robotic arms such as the UR-5 and ABB IRB-140.			
	Test			
	Move sensor of safety system to a different location using the same configuration.			

Requirement	Date	Status	Modified	Name
1.5	24-09-2016	Submitted	30-11-2016	Not limiting work area
	Description			
	The safety system should not limit the work area. There should not be the need to install masts or poles limiting the work area to accommodate sensors.			
	Motivation			
	The safety system should only improve the working environment not limit accessibility. The work area is defined as a 3-meter radius around the robotic arm.			
	Test			
	Inspect working environment empirically.			

Requirement	Date	Status	Modified	Name
1.6	24-09-2016	Submitted	30-11-2016	Payload
	Description			
	The safety system should be developed based on a robotic arm with an end effector payload of up to 5kg with a bolt sticking out.			
	Motivation			
	A safety system for an industrial welding robot versus a small collaborative robot is a different scope			
	Test			
	Check payload of robot in specifications and see if the safety system works for this robot.			

Requirement	Date	Status	Modified	Name
1.7	24-09-2016	Submitted	30-11-2016	Continued operation
	Description			
	The robot should not immediately stop when an obstacle is detected within the working area. The work area is defined as a 3-meter radius around the robotic arm.			
	Motivation			
	The robot should work in collaboration with the operator and not stop each time a hand or other obstacle is detected. Instead the robot should plan a path around the obstacle. This way the operator and robot can work at the same time on the same product. The robot should only stop if there is no available path around the obstacle and if it is within a distance of <50mm.			
	Test			
	Create an obstacle within the working area of the robot and check if the robot continues operation.			

Requirement	Date	Status	Modified	Name
1.8	24-09-2016	Submitted	-	ISO norms
	Description			
	The safety system must be designed and constructed with the purpose of approaching the health and safety requirements as stated in ISO 10218-1, ISO 10218-2 and ISO TS 15066. However, taking cooperation of human and machine in the work area, it may not be possible to meet the objectives set by ISO 10218-1 and ISO 10218-2. Also, see Machinery Directive, Annex 1, General principles 3.			
	Motivation			
	Following these ISO norms is strongly advised in order to comply with the machinery directive			
	Test			
	Verify if all solutions fit within the ISO norms.			

Requirement	Date	Status	Modified	Name
1.9	24-09-2016	Submitted	30-11-2016	Environment adaptive
	Description			
	The robot should be able to adapt to the environment. The robot should be able to adjust its path to plan a new path around objects, and adjust its speed according to the ISO TS 15066 norm. This speed is dependent on the distance between robot arm and object in the work space. This information will come from the safety system.			
	Motivation			
	The goal is to design a modular system so the system needs to adapt itself to different environments. Hazard 1.1, 1.3, 1.4, 1.5 from Risk Assessment.			
	Test			
Create an obstacle within the working area of the robot and analyse robot behaviour accordingly.				

Requirement	Date	Status	Modified	Name
1.10	24-09-2016	Submitted	-	Collaborative design
	Description			
	The robot should have rounded edges with a minimum radius of 0.3mm. And the robot should not have open spaces within the arm construction. Hazard 1.3 from Risk Assessment.			
	Motivation			
	The robot should be designed with collaborative operation as a possibility. Hazard 1.1 from Risk Assessment.			
	Test			
	Visually and by touch inspect the robot.			

Requirement	Date	Status	Modified	Name
1.11	24-09-2016	Submitted	-	Separate controllers
	Description			
	The safety system controller must be separate from the robot controller.			
	Motivation			
	One of the safety norms list that the safety system must be controlled separately.			
	Test			
	Disconnect the robot controller and check if the safety system controller still works.			

Requirement	Date	Status	Modified	Name
1.12	24-09-2016	Submitted	30-11-2016	Modularity, robots
	Description			
	The safety system must be usable with different robots (ABB IRB-140 and UR-5) which are using ROS.			
	Motivation			
	The safety system must be capable of interfacing with different robots with as minimal modifications as possible, e.g. only changing the interface with the robot.			
	Test			
	Connect the safety system to the UR5 and ABB robots available in the Mechatronics lab			

Requirement	Date	Status	Modified	Name
1.13	24-09-2016	Submitted	-	Not limiting human movement
	Description			
	The safety system should not limit human movement. The area the human operator can access should not be limited, and restrictive clothing is not preferred.			
	Motivation			
	Should improve the work flow for workers, not cause annoyance.			
	Test			
	Empirical test			

Requirement	Date	Status	Modified	Name
1.14	24-09-2016	Submitted	-	Industrial environment, IP rating
	Description			
	The system should be IP rating compliant. It should be dust resistant or sealed against dust and metal shavings, e.g. IP-5x or IP-6x rated.			
	Motivation			
	The system will be used in an industrial environment			
	Test			
	Verify if all solutions fit within the IP ratings.			

Requirement	Date	Status	Modified	Name
1.15	24-09-2016	Submitted	-	Annoyance
	Description			
	The human operator should not be “annoyed” by the safety system, e.g. there should be no need for protective eye wear. In other words, the safety system should not add any restrictions to the user.			
	Motivation			
	Should improve the work flow for workers, not cause annoyance.			
	Test			
	Empirical test			

Requirement	Date	Status	Modified	Name
1.16	24-09-2016	Submitted	-	Minimal impact on robot performance
	Description			
	The safety system should minimally or preferably not impact the efficiency of the robot.			
	Motivation			
	The goal is to improve work flow, limiting the robot will be contradictory			
	Test			
	Measure productivity of the robot with and without the safety system. Compare these results.			

Requirement	Date	Status	Modified	Name
1.17	24-09-2016	Submitted	30-11-2016	Industrial environment, temperature
	Description			
	The system must work within the temperature range for industrial environments. 10°C to 40°C.			
	Motivation			
	The system will be implemented in an industrial environment.			
	Test			
	Run the safety system in different temperature conditions and verify if it works.			

Requirement	Date	Status	Modified	Name
1.18	24-09-2016	Submitted	-	Lighting conditions
	Description			
	The safety system should work under dynamic lighting conditions. The system should not be influenced by changing light condition, e.g. when opening a door or port. The operation range shall be at least from 100 Lux to 20000 Lux.			
	Motivation			
	The system will be used in an industrial environment where lighting conditions could change a lot.			
	Test			
	Run the safety system and determine if it works when blinds are opened or a light is shined on it.			

Requirement	Date	Status	Modified	Name
1.19	31-10-2016	Submitted	30-11-2016	Do not introduce additional hazards
	Description			
	The safety system itself should not introduce additional hazards. For example, the safety system should not have sharp edges or block emergency exits.			
	Motivation			
	When creating a safety system for the robotic arm, the system itself needs to be safe as well and not come with hazards.			
	Test			
	Risk assessment of the safety system itself			

Requirement	Date	Status	Modified	Name
1.20	31-10-2016	Submitted	-	Eliminate trapping and crushing
	Description			
	Risks associated with whole body trapping or crushing between the robot system and, for example, parts of buildings, structures, utilities, other machines, and equipment, shall be eliminated or safely controlled. Clearance in accordance with ISO 10218-2:2011, 5.11.3 should be provided.			
	Motivation			
	The safety system is supposed to prevent a human from being trapped. Hazard 1.4 from Risk Assessment.			
	Test			
	Try to trap a pool noodle between the wall			

Appendix 2.2 By safeguards

Requirement	Date	Status	Modified	Name
2.1	24-09-2016	Submitted	-	Stop within unsafe distance
	Description			
	The robot must stop when it comes within an unsafe distance (<50mm) to an obstacle. Although the robot should continue working, when no other option is available the robot must stop when it comes into an unsafe distance (<50mm) to an obstacle.			
	Motivation			
	Avoid ever touching a human. Hazard 1.1 and 1.2 from Risk Assessment.			
	Test			
	Place an obstacle on the robot's path and verify if it stops			

Requirement	Date	Status	Modified	Name
2.2	24-09-2016	Submitted	-	Non-stationary robotic arm
	Description			
	It would be preferred to have a safety system that works for a robot which moves over the work floor, non-stationary robotic arms			
	Motivation			
	Not all robotic arms are stationary			
	Test			
	Test the safety system with a moving robot			

Requirement	Date	Status	Modified	Name
2.3	24-09-2016	Submitted	31-10-2016	Emergency stop
	Description			
	The safety system must include an emergency stop. The number and location of emergency stop devices shall be determined by risk assessment and shall meet the requirements of ISO 13850			
	Motivation			
	If an emergency happens. The system must completely shut down immediately.			
	Test			
	Disturb the safety system and empirically check if the light status indicator changes			

Requirement	Date	Status	Modified	Name
2.4	31-10-2016	Submitted	-	Safeguards ISO
	Description			
	Settings and adjusting collaborative safety parameters safety parameters shall be protected against unauthorized and unintentional changes by password protection or similar security measures.			
	Motivation			
	To ensure safety at all times, the safety settings may not be changed without permission. This must be protected with a password or similar protection			
	Test			
	Try changing safety setting, see if one needs a password or similar kind of			

	permission.
--	-------------

Appendix 2.3 By information

Requirement	Date	Status	Modified	Name
3.1	24-09-2016	Submitted	-	Operating state indication (Safety system)
	Description			
	The safety system must indicate its operating state using a display. The operator should be able to recognize the operating state of the safety system using feedback on a display.			
	Motivation			
	Helps with predictability and user information			
	Test			
	Run the safety system and check display to verify if it indicates the operating state.			

Requirement	Date	Status	Modified	Name
3.2	24-09-2016	Submitted	-	Operating state indication (Robot)
	Description			
	The human operator should be able to easily recognize the state of the robot by a stack of lights			
	Motivation			
	The operator must know if the safety system is activated and working properly or not.			
	Test			
	Disturb the safety system and empirically check if the light status indicator changes			

Appendix 3 Test document

This document explains how each requirement will be tested. The tests will be performed to guarantee that the requirement is met. Each test will be performed on a correctly working safety system which means that the system is running without errors. Requirements 1.5, 1.6, 1.8, 1.10, 1.14, 1.16, 1.17, 1.19, 2.2, 2.3, 2.4, 3.1 and 3.2 are not recorded in this document since these are requirements for the design of the system and not its function. All tests are yet to be performed.

Test setup 1

The test for requirement 1.1 can be found in Table 18.

Table 18 Test 1

The Requirements	1.1: The safety system must perform a self-check on start. If the system does not pass its self-check, the robot must not power on and send an error message to the user.
The Test	The safety system is off. When the system is turned on the system must perform a self-check to see if everything is working correctly. when the robot is self-checking, this can be seen on the screen of the robot. If the self-check has no errors the robot can start working.
	The safety system is off. Make sure the camera is disconnected for this test. When the system is turned on the system must perform a self-check to see if everything is working correctly. The system must generate an error since the camera is disconnected. If the self-check is working correctly the robot will not start until the problem has been solved and the system is restarted.
	The safety system is off. Make sure the cable between the robot and computer is disconnected for this test. When the system is turned on the system must perform a self-check to see if everything is working correctly. The system must generate an error since the camera is disconnected. If the self-check is working correctly the robot will not start until the problem has been solved and the system is restarted.
	The safety system is off. Make sure the cable between the camera and computer is disconnected for this test. When the system is turned on the system must perform a self-check to see if everything is working correctly. The system must generate an error since the camera is disconnected. If the self-check is working correctly the robot will not start until the problem has been solved and the system is restarted.

Test setup 2

The test for requirement 1.2 can be found in Table 19.

Table 19 Test 2

The Requirements	1.2: The safety system must be compliant with EMC directives. EMC directive 2014/30/EU
------------------	--

The Test	Put the system in a EMC neutral zone. Expose the system to different types of electric and magnetic waves to make sure there is no electromagnetic interference.
----------	--

Test setup 3

The test for requirement 1.3 can be found in Table 20.

Table 20 Test 3

The Requirements	1.3: The safety system must be usable even if the total system is moved to a different location, but still using the same configuration as specified.
The Test	When the system is running make sure it has no errors. Shut the system down and move the system to another location, restart the system. If it works correctly, the requirement is met.

Test Setup 4

The test for requirements 1.4 and 1.12 can be found in Table 21.

Table 21 Test 4

The Requirements	1.4: The safety system must be usable even if the sensors are applied to a different robotic arm.
	1.12: The safety system must be usable with different robots (ABB IRB-140 and UR-5) which are using ROS.
The Test	When the system is working correctly. Connect the safety system to a different robotic arm. These requirements are met when the system also works on a different robotic arm.

Test setup 5

The test for requirement 1.7 can be found in Table 22.

Table 22 Test 5

The Requirements	1.7: The robot should not immediately stop when an obstacle is detected within the working area. Only when there is no path around the object the robot should stop at the minimum distance of 50mm of the object, the robot shall stop. The work area is defined as a 3-meter radius around the robotic arm.
The Test	While the system is running, test the requirement by adding different objects to the field of view. The robot should continue working until it has a collision in its path and the distance till this collision is equal or below 50mm.

Test setup 6

The test for requirement 1.9 can be found in Table 23.

Table 23 Test 6

The Requirements	1.9: The robot should be able to adapt to the environment. The robot should be able to adjust its path to plan a new path around objects, and adjust its speed according to the ISO TS 15066 norm. This speed is dependent on the
------------------	---

	distance between robot arm and object in the work space. This information will come from the safety system.
The Test	While the system is running, test the requirement by adding different objects to the field of view. The robot should continue working until it has a collision in its path. When this happens, the robot should make a new path to avoid the object.

Test setup 7

The test for requirement 1.11 can be found in Table 24.

Table 24 Test 7

The Requirements	1.11: The safety system controller must be separate from the robot controller.
The Test	While the safety system is running. Disconnect the robot to check if the safety system continues running. By checking the display and the software.

Test setup 8

The test for requirement 1.13 and 1.15 can be found in Table 25.

Table 25 Test 8

The Requirements	1.13: The safety system should not limit human movement. The area the human operator can access should not be limited, and restrictive clothing is not preferred.
	1.15: The human operator should not be annoyed by the safety system, e.g. there should be no need for protective eye wear.
The Test	While the system is running, test the requirement by adding different objects to the field of view. The robot should continue working until it has a collision in its path. When the object/human is found in the path the robot must stop before it reaches the unsafe distance to human/object (<50mm). Test if the robot plans a new path to avoid the obstacle.

Test setup 9

The test for requirement 1.18 can be found in Table 26.

Table 26 Test 9

The Requirements	1.18: The safety system should work under dynamic lighting conditions. The system should not be influenced by changing light condition, e.g. when opening a door or port. The operation range shall be at least from 100 Lux to 20000 Lux.
The Test	The system is working correctly. Change the lighting in the room by putting the robot in different lights (TL, floodlight), brighter and darker. For this requirement to be met the safety system must continue working.

Test setup 10

The test for requirement 1.20 can be found in Table 27.

Table 27 Test 10

The Requirements	1.20: Risks associated with whole body trapping or crushing between the robot system and, for example, parts of buildings, structures, utilities, other machines, and equipment, shall be eliminated or safely controlled. Clearance in accordance with ISO 10218-2:2011, 5.11.3 should be provided.
The Test	The system is running. Try to trap a pool noodle between the robot and an object. If the safety system works correctly, the pool noodle cannot be trapped.

Test setup 11

The test for requirement 2.1 can be found in Table 28.

Table 28 Test 11

The Requirements	2.1: The robot must stop when it comes within an unsafe distance (<50mm) to an obstacle. Although the robot should continue working, when no other option is available the robot must stop when it comes into an unsafe distance (<50mm) to an obstacle.
The Test	Place obstacles within the range of the robot and have the robot run a course. When the robot comes near an obstacle it must stop when it comes below the distance of 50mm.

Appendix 4 Software code

Appendix 4.1 Vision test bluefox

```
#!/usr/bin/env python
from __future__ import print_function
from prj7_vision_test.msg import prj7_box
import roslib
import sys
import rospy
import cv2
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import imutils
import numpy as np

#Authors: Guido laessen, Bas Janssen
#Fontys Mechatronica
#2016

# Class that converts the image/video obtained from the bluefox2_single/image_raw topic to a
# image/video which
# can be edited in python with use of openCV.
class image_converter:
    def __init__(self):
        # Publisher for the box coordinates and size
        self.box_pub = rospy.Publisher("prj7_box_bluefox", prj7_box, queue_size=1000)

        self.bridge = CvBridge()
        # The image/video is received from the bluefox2_single/image_raw topic
        self.image_sub = rospy.Subscriber("bluefox2_single/image_raw", Image, self.callback)

    def callback(self, data):
        try:
            # cv_image = converted image/video from ROS to openCV
            cv_image = self.bridge.imgmsg_to_cv2(data, "mono8")
        except CvBridgeError as e:
            print(e)

        #-----Code to detect objects / get cordinates -----
        # blur the image slightly, and threshold it
        blurred = cv2.GaussianBlur(cv_image, (5, 5), 0)
        thresh = cv2.threshold(blurred, 84, 255, cv2.THRESH_BINARY_INV)[1]
        # find contours in the thresholded image
        cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if imutils.is_cv2() else cnts[1]
        id = 0
        # loop over the contours
```

```

for c in cnts:
    rect = cv2.minAreaRect(c)
    box = cv2.boxPoints(rect)
    box = np.int0(box) # this returns the center(x,y), width, height, and angle of rotation
    rect2 = np.array(rect)
    xy = np.array(rect2[0])
    size = np.array(rect2[1])
    # Draw smallest possible rectangle around object
    cv2.drawContours(cv_image, [box], 0, (0, 255, 0), 2)
    self.box_pub.publish(id, xy[0], xy[1], size[0], size[1], rect[2])
    id = id + 1
    # print (rect)
    #-----
# Window which displays the image
cv2.imshow("Thresh", thresh)
cv2.imshow("Detection", cv_image)
cv2.waitKey(3)

def main(args):
    rospy.init_node('prj7_vision_test_bluefox', anonymous=True)
    image_converter()
    try:
        rospy.spin()
    except KeyboardInterrupt:
        print("Shutting down")
        cv2.destroyAllWindows()

if __name__ == '__main__':
    main(sys.argv)

#-----

```

Appendix 4.2 Vision test 2 (Background subtraction)

```
#!/usr/bin/env python
from __future__ import print_function
from prj7_vision_test.msg import prj7_box
import roslib
import sys
import rospy
import cv2
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import imutils
import numpy as np

#Authors: Guido laessen, Bas Janssen
#Fontys Mechatronica
#2016

# Class that converts the image/video obtained from the bluefox2_single/image_raw topic to a
# image/video which
# can be edited in python with use of openCV.
class image_converter:
    def __init__(self):
        # The resulting image/video after editing is published to the image_topic_2 topic
        self.image_pub = rospy.Publisher("image_topic_2", Image, queue_size=1000)
        # Publisher for the box coordinates and size
        self.box_pub = rospy.Publisher("prj7_box", prj7_box, queue_size=1000)

    self.bridge = CvBridge()
    # The image/video is received from the bluefox2_single/image_raw topic
    self.image_sub = rospy.Subscriber("usb_cam/image_raw", Image, self.callback)

    def callback(self, data):
        try:
            # cv_image = converted image/video from ROS to openCV
            cv_image = self.bridge.imgmsg_to_cv2(data, "mono8")
        except CvBridgeError as e:
            print(e)
        #-----Code to detect objects / get coordinates -----
        # Subtract the background, blur the image slightly, and threshold it
        background = cv2.imread("/home/bas/catkin_ws/src/prj7_vision_test/src/background.png", 0)
        foreground = cv2.absdiff(cv_image, background)
        blurred = cv2.GaussianBlur(foreground, (5, 5), 0)
        thresh = cv2.threshold(blurred, 25, 255, cv2.THRESH_BINARY)[1]
        # find contours in the thresholded image
        cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if imutils.is_cv2() else cnts[1]
        id = 0
```

```

# loop over the contours
for c in cnts:
    rect = cv2.minAreaRect(c)
    box = cv2.boxPoints(rect)
    box = np.int0(box) # this returns the center(x,y), width, height, and angle of rotation
    rect2 = np.array(rect)
    xy = np.array(rect2[0])
    size = np.array(rect2[1])
    # Draw smallest possible rectangle around object
    # cv2.drawContours(cv_image, [box], 0, (0, 255, 0), 2)
    self.box_pub.publish(id, xy[0], xy[1], size[0], size[1], rect[2])
    id = id + 1
#-----

# Window which displays the image
cv2.imshow("Thresh", thresh)
cv2.imshow("Detection", cv_image)
cv2.imshow("Foreground", foreground)
cv2.waitKey(3)
try:
    # Convert image/video back from openCV to ROS
    self.image_pub.publish(self.bridge.cv2_to_imgmsg(cv_image, "mono8"))
except CvBridgeError as e:
    print(e)

def main(args):
    rospy.init_node('image_converter', anonymous=True)
    image_converter()
    try:
        rospy.spin()
    except KeyboardInterrupt:
        print("Shutting down")
    cv2.destroyAllWindows()

if __name__ == '__main__':
    main(sys.argv)

#-----

```

Appendix 4.3 QR detection Bluefox

```
#!/usr/bin/env python
from __future__ import print_function
from prj7_vision_test.msg import prj7_box
import roslib
import sys
import rospy
import cv2
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import imutils
import numpy as np
import zbar
import Image as Image_

#Authors: Guido Claessen, Bas Janssen
#Fontys Mechatronica
#2016

x = 0
zeroX = 0
zeroY = 0
pixel = 1

# Class that converts the image/video obtained from the bluefox2_single/image_raw topic to a
# image/video which
# can be edited in python with use of openCV.
# The class can also determine the location of its reference qr code to calculate relative position and size
# of the detected objects.
class image_converter:
    def __init__(self):
        # Publisher for the box coordinates and size
        self.box_pub = rospy.Publisher("prj7_box_bluefox", prj7_box, queue_size=1000)

        self.bridge = CvBridge()
        # The image/video is received from the bluefox2_single/image_raw topic
        self.image_sub = rospy.Subscriber("bluefox2_single/image_raw", Image, self.callback)

    def zeroreff(self, cv_image):
        global zeroX
        global zeroY
        global pixel
        # create a reader
        scanner = zbar.ImageScanner()

        # configure the reader
        scanner.parse_config('enable')
```

```

# sharpen the camera image, for better qr code detection.
kernel_sharpen_3 = np.array([[[-1,-1,-1,-1,-1],
[-1,2,2,2,-1],
[-1,2,8,2,-1],
[-1,2,2,2,-1],
[-1,-1,-1,-1,-1]]] / 8.0
output_3 = cv2.filter2D(cv_image, -1, kernel_sharpen_3)

# convert the image data to a suitable format for zbar
width, height = output_3.shape[:2]
raw = output_3.tostring()
pil = Image_.fromstring("L", (height, width), raw)
width, height = pil.size
zbarString = pil.tostring()

# wrap image data
image = zbar.Image(width, height, 'Y800', zbarString)

# scan the image for barcodes
scanner.scan(image)

# extract results
for symbol in image:
# do something useful with results
if symbol.data == "None":
return "No QR code found"
else:
loc = symbol.location
print (loc)
zeroX = (loc[0][0]+loc[2][0])/2
zeroY = (loc[0][1]+loc[2][1])/2
width = (loc[1][0] - loc[0][0])
height = (loc[2][1] - loc[1][1])
pixel_x = abs(74.0 / width)#/1000
pixel_y = abs(74.0 / height)#/1000
pixel = max(pixel_x, pixel_y)
print (symbol.data)
print ("Zero:", zeroX, zeroY)
print ("Dimensions:", width, height)
print ("mm/px:", pixel_x, pixel_y, pixel)
# cv2.rectangle(cv_image, symbol.location[0], symbol.location[2], (0, 255, 0))

def callback(self,data):
try:
# cv_image = converted image/video from ROS to openCV
cv_image = self.bridge.imgmsg_to_cv2(data, "mono8")
except CvBridgeError as e:

```

```

print(e)

global zeroX
global zeroY
global pixel
global x
# Perhaps we can do this once in a while to make sure the calibration stays accurate.
if x<1:
    image_converter.zeroreff(self, cv_image)
    x = 1

#-----Code to detect objects / get cordinates -----
# blur the image slightly, and threshold it
blurred = cv2.GaussianBlur(cv_image, (5, 5), 0)
thresh = cv2.threshold(blurred, 80, 255, cv2.THRESH_BINARY_INV)[1]
# find contours in the thresholded image
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if imutils.is_cv2() else cnts[1]
id = 0
# loop over the contours
for c in cnts:
    rect = cv2.minAreaRect(c)
    box = cv2.boxPoints(rect)
    box = np.int0(box) # this returns the center(x,y), width, heigth, and angle of rotation
    rect2 = np.array(rect)
    xy = np.array(rect2[0])
    size = np.array(rect2[1])
    # Draw smallest possible rectangle around object
    cv2.drawContours(cv_image, [box], 0, (0, 255, 0), 2)
    #Publish the center location, size and rotation of the box to the box topic in meters and degrees
    (@TODO convert degrees to radians)
    self.box_pub.publish(id, (xy[0] - zeroX) * pixel, (xy[1] - zeroY) * pixel, size[0] * pixel, size[1] * pixel,
    rect[2])
    id = id + 1
# print (rect)
#-----
# Window which displays the image
cv2.imshow("Thresh", thresh)
cv2.imshow("Detection", cv_image)
cv2.waitKey(3)

def main(args):
    rospy.init_node('prj7_vision_test_bluefox', anonymous=True)
    image_converter()
    try:
        rospy.spin()
    except KeyboardInterrupt:

```

```
print("Shutting down")  
cv2.destroyAllWindows()
```

```
if __name__ == '__main__':  
    main(sys.argv)
```

```
#-----
```


Appendix 4.4 QR detection Bluefox with background subtraction

```
#!/usr/bin/env python
from __future__ import print_function
from prj7_vision_test.msg import prj7_box
import roslib
import sys
import rospy
import cv2
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import imutils
import numpy as np
import zbar
import Image as Image_

#Authors: Guido Claessen, Bas Janssen
#Fontys Mechatronica
#2016

x = 0
zeroX = 0
zeroY = 0
pixel = 1
background = 0

# Class that converts the image/video obtained from the bluefox2_single/image_raw topic to a
# image/video which
# can be edited in python with use of openCV.
# The class can also determine the location of its reference qr code to calculate relative position and size
# of the detected objects.
class image_converter:
    def __init__(self):
        # Publisher for the box coordinates and size
        self.box_pub = rospy.Publisher("prj7_box_bluefox", prj7_box, queue_size=1000)

        self.bridge = CvBridge()
        # The image/video is received from the bluefox2_single/image_raw topic
        self.image_sub = rospy.Subscriber("bluefox2_single/image_raw", Image, self.callback)

    def zeroreff(self, cv_image):
        global zeroX
        global zeroY
        global pixel
        global background
        background = cv_image
        # create a reader
        scanner = zbar.ImageScanner()
```

```

# configure the reader
scanner.parse_config('enable')

# sharpen the camera image, for better qr code detection.
kernel_sharpen_3 = np.array([[[-1,-1,-1,-1,-1],
[-1,2,2,2,-1],
[-1,2,8,2,-1],
[-1,2,2,2,-1],
[-1,-1,-1,-1,-1]]] / 8.0
output_3 = cv2.filter2D(cv_image, -1, kernel_sharpen_3)

# convert the image data to a suitable format for zbar
width, height = output_3.shape[:2]
raw = output_3.tostring()
pil = Image_.fromstring("L", (height, width), raw)
width, height = pil.size
zbarString = pil.tostring()

# wrap image data
image = zbar.Image(width, height, 'Y800', zbarString)

# scan the image for barcodes
scanner.scan(image)

# extract results
for symbol in image:
    # do something useful with results
    if symbol.data == "None":
        return "No QR code found"
    else:
        loc = symbol.location
        print (loc)
        zeroX = (loc[0][0]+loc[2][0])/2
        zeroY = (loc[0][1]+loc[2][1])/2
        width = (loc[1][0] - loc[0][0])
        height = (loc[2][1] - loc[1][1])
        pixel_x = abs(74.0 / width)#/1000
        pixel_y = abs(74.0 / height)#/1000
        pixel = max(pixel_x, pixel_y)
        print (symbol.data)
        print ("Zero:", zeroX, zeroY)
        print ("Dimensions:", width, height)
        print ("mm/px:", pixel_x, pixel_y, pixel)
        # cv2.rectangle(cv_image, symbol.location[0], symbol.location[2], (0, 255, 0))

def callback(self,data):
    try:

```

```

# cv_image = converted image/video from ROS to openCV
cv_image = self.bridge.imgmsg_to_cv2(data, "mono8")
except CvBridgeError as e:
    print(e)

global zeroX
global zeroY
global pixel
global x
global background
# Perhaps we can do this once in a while to make sure the calibration stays accurate.
if x<1:
    image_converter.zeroreff(self, cv_image)
    x = 1

#-----Code to detect objects / get cordiates -----
#Subtract the background, blur the image slightly, and threshold it

# background = cv2.imread("/home/bas/catkin_ws/src/prj7_vision_test/src/background.png", 0)
foreground = cv2.absdiff(cv_image, background)

# blurred = cv2.GaussianBlur(cv_image, (5, 5), 0)
# thresh = cv2.threshold(blurred, 80, 255, cv2.THRESH_BINARY_INV)[1]
blurred = cv2.GaussianBlur(foreground, (5, 5), 0)
thresh = cv2.threshold(blurred, 50, 255, cv2.THRESH_BINARY)[1]
# find contours in the thresholded image
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if imutils.is_cv2() else cnts[1]
id = 0
# loop over the contours
for c in cnts:
    rect = cv2.minAreaRect(c)
    box = cv2.boxPoints(rect)
    box = np.int0(box) # this returns the center(x,y), width, height, and angle of rotation
    rect2 = np.array(rect)
    xy = np.array(rect2[0])
    size = np.array(rect2[1])
    # Draw smallest possible rectangle around object
    cv2.drawContours(cv_image, [box], 0, (0, 255, 0), 2)
    #Publish the center location, size and rotation of the box to the box topic in meters and degrees
    (@TODO convert degrees to radians)
    self.box_pub.publish(id, ((xy[1] - zeroY) * pixel) + 600, ((xy[0] - zeroX) * pixel) - 600, size[1] * pixel, size[0]
    * pixel, rect[2])
    id = id + 1
# print (rect)
#-----
# Window which displays the image

```

```
# cv2.imshow("Foreground", foreground)
cv2.imshow("Thresh", thresh)
cv2.imshow("Detection", cv_image)
cv2.waitKey(3)

def main(args):
    rospy.init_node('prj7_vision_test_bluefox', anonymous=True)
    image_converter()
    try:
        rospy.spin()
    except KeyboardInterrupt:
        print("Shutting down")
        cv2.destroyAllWindows()

if __name__ == '__main__':
    main(sys.argv)

#-----
```

Appendix 4.5 QR webcam

```
#!/usr/bin/env python
from __future__ import print_function
from prj7_vision_test.msg import prj7_box
import roslib
import sys
import rospy
import cv2
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import imutils
import numpy as np
import zbar
import Image as Image_

#Authors: Guido Claessen, Bas Janssen
#Fontys Mechatronica
#2016

x = 0
zeroX = 0
zeroY = 0
pixel = 1

# Class that converts the image/video obtained from the webcam topic to an image/video which
# can be edited in python with use of openCV.
# The class can also determine the location of its reference qr code to calculate relative position and size
# of the detected objects.
class image_converter:
    def __init__(self):
        # Publisher for the box coordinates and size
        self.box_pub = rospy.Publisher("prj7_box_usb", prj7_box, queue_size=1000)

        self.bridge = CvBridge()
        # The image/video is received from the bluefox2_single/image_raw topic
        self.image_sub = rospy.Subscriber("usb_cam/image_raw", Image, self.callback)

    def zeroreff(self, cv_image):
        global zeroX
        global zeroY
        global pixel
        # create a reader
        scanner = zbar.ImageScanner()

        # configure the reader
        scanner.parse_config('enable')
```

```

# sharpen the camera image, for better qr code detection.
kernel_sharpen_3 = np.array([[ -1,-1,-1,-1,-1],
[ -1,2,2,2,-1],
[ -1,2,8,2,-1],
[ -1,2,2,2,-1],
[ -1,-1,-1,-1,-1]]) / 8.0
output_3 = cv2.filter2D(cv_image, -1, kernel_sharpen_3)

# convert the image data to a suitable format for zbar
width, height = output_3.shape[:2]
raw = output_3.tostring()
pil = Image_.fromstring("L", (height, width), raw)
width, height = pil.size
zbarString = pil.tostring()

# wrap image data
image = zbar.Image(width, height, 'Y800', zbarString)

# scan the image for barcodes
scanner.scan(image)

# extract results
for symbol in image:
# do something useful with results
if symbol.data == "None":
return "No QR code found"
else:
loc = symbol.location
print (loc)
zeroX = (loc[0][0]+loc[2][0])/2
zeroY = (loc[0][1]+loc[2][1])/2
width = (loc[1][0] - loc[0][0])
height = (loc[2][1] - loc[1][1])
pixel_x = abs(74.0 / width) / 1000
pixel_y = abs(74.0 / height) / 1000
pixel = max(pixel_x, pixel_y)
print (symbol.data)
print ("Zero:", zeroX, zeroY)
print ("Dimensions:", width, height)
print ("mm/px:", pixel_x, pixel_y, pixel)
# cv2.rectangle(cv_image, symbol.location[0], symbol.location[2], (0, 255, 0))

def callback(self,data):
try:
# cv_image = converted image/video from ROS to openCV
cv_image = self.bridge.imgmsg_to_cv2(data, "mono8")
except CvBridgeError as e:
print(e)

```

```

global zeroX
global zeroY
global pixel
global x
# Perhaps we can do this once in a while to make sure the calibration stays accurate.
if x<1:
    image_converter.zeroreff(self, cv_image)
    x = 1

#-----Code to detect objects / get cordinates -----
# blur the image slightly, and threshold it
background = cv2.imread("/home/bas/catkin_ws/src/prj7_vision_test/src/background.png", 0)
foreground = cv2.absdiff(cv_image, background)

blurred = cv2.GaussianBlur(foreground, (5, 5), 0)
thresh = cv2.threshold(blurred, 84, 255, cv2.THRESH_BINARY)[1]
# find contours in the thresholded image
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if imutils.is_cv2() else cnts[1]
id = 0
# loop over the contours
for c in cnts:
    rect = cv2.minAreaRect(c)
    box = cv2.boxPoints(rect)
    box = np.int0(box) # this returns the center(x,y), width, height, and angle of rotation
    rect2 = np.array(rect)
    xy = np.array(rect2[0])
    size = np.array(rect2[1])
    # Draw smallest possible rectangle around object
    cv2.drawContours(cv_image, [box], 0, (0, 255, 0), 2)
    #Publish the center location, size and rotation of the box to the box topic in meters and degrees
    (@TODO convert degrees to radians)
    self.box_pub.publish(id, (xy[0] - zeroX) * pixel, (xy[1] - zeroY) * pixel, size[0] * pixel, size[1] * pixel,
    rect[2])
    id = id + 1
# print (rect)
#-----
# Window which displays the image
cv2.imshow("Thresh", thresh)
cv2.imshow("Detection", cv_image)
cv2.waitKey(3)

def main(args):
    rospy.init_node('prj7_vision_test_bluefox', anonymous=True)
    image_converter()
    try:

```

```
rospy.spin()
except KeyboardInterrupt:
    print("Shutting down")
    cv2.destroyAllWindows()
```

```
if __name__ == '__main__':
    main(sys.argv)
```

```
#-----
```


Appendix 4.6 Move test ABB IRB120

```
#include <ros/ros.h>
#include <moveit/move_group_interface/move_group.h>

//This application plans solvable positions for an ABB IRB120 arm.

int main(int argc, char **argv)
{
    ros::init(argc, argv, "prj7_move_test");
    ros::AsyncSpinner spinner(1);
    spinner.start();

    moveit::planning_interface::MoveGroup group("manipulator");
    moveit::planning_interface::MoveGroup::Plan my_plan;
    ros::Duration(2.0).sleep();

    float x,y,z;
    x = 0;
    y = 0.3;
    z = 0.8;

    int i = 0;

    while(ros::ok())
    {
        //          if(i == 2)
        //          {
        //              break;
        //          }
        x = 0;
        y = 0.3;
        z = 0.8;
        ROS_INFO("Planning for target %f, %f, %f", x, y, z);
        group.setPositionTarget(x, y, z, "");
        bool success = group.plan(my_plan);
        ROS_INFO("Planning result: %i", success);
        ros::Duration(2.0).sleep();
        if(success)
        {
            ROS_INFO("Moving to target %f, %f, %f", x, y, z);
            group.execute(my_plan);
        }

        //          ros::Duration(3.0).sleep();

        x = -0.4;
        y = -0.3;
```

```

        z = 0.7;
        ROS_INFO("Planning for target %f, %f, %f", x, y, z);
        group.setPositionTarget(x, y, z, "");
        success = group.plan(my_plan);
        ROS_INFO("Planning result: %i", success);
        ros::Duration(2.0).sleep();
        if(success)
        {
            ROS_INFO("Moving to target %f, %f, %f", x, y, z);
            group.execute(my_plan);
        }

//        ros::Duration(3.0).sleep();

        x = 0.2;
        y = -0.4;
        z = 0.7;
        ROS_INFO("Planning for target %f, %f, %f", x, y, z);
        group.setPositionTarget(x, y, z, "");
        success = group.plan(my_plan);
        ROS_INFO("Planning result: %i", success);
        ros::Duration(2.0).sleep();
        if(success)
        {
            ROS_INFO("Moving to target %f, %f, %f", x, y, z);
            group.execute(my_plan);
        }

//        ros::Duration(3.0).sleep();

        x = 0;
        y = -0.6;
        z = 0.5;
        ROS_INFO("Planning for target %f, %f, %f", x, y, z);
        group.setPositionTarget(x, y, z, "");
        success = group.plan(my_plan);
        ROS_INFO("Planning result: %i", success);
        ros::Duration(2.0).sleep();
        if(success)
        {
            ROS_INFO("Moving to target %f, %f, %f", x, y, z);
            group.execute(my_plan);
        }

//        ros::Duration(3.0).sleep();
        i++;
    }
    return 0;

```

}

Appendix 4.7 Move test ABB IRB140

```
#include <ros/ros.h>
#include <moveit/move_group_interface/move_group.h>
#include <geometry_msgs/Pose.h>

//This application planes solvable positions for an ABB IRB140 arm.

int main(int argc, char **argv)
{
    ros::init(argc, argv, "prj7_move_test");

    ros::AsyncSpinner spinner(1);
    spinner.start();

    moveit::planning_interface::MoveGroup group("manipulator");
    ros::Duration(2.0).sleep();

    geometry_msgs::Pose pose;

    float x,y,z,roll,pitch,yaw;

    int i = 0;

    group.allowLooking(true);
    group.allowReplanning(true);

    while(ros::ok())
    {
        //          if(i == 2)
        //          {
        //              break;
        //          }
        moveit::planning_interface::MoveGroup::Plan my_plan;
        x = 0.2;
        y = 0;
        z = 0.7;
        roll = 0;
        pitch = 0;
        yaw = 0;
        ROS_INFO("Planning for target %f, %f, %f", x, y, z);
        pose.position.x = x;
        pose.position.y = y;
        pose.position.z = z;
        pose.orientation.x = roll;
        pose.orientation.y = pitch;
        pose.orientation.z = yaw;
        pose.orientation.w = 1.0;
```

```

group.setPoseTarget(pose, "");
bool success = group.plan(my_plan);
while(!success)
{
    ros::Duration(1.0).sleep();
    success = group.plan(my_plan);
}
ROS_INFO("Planning result: %i", success);
ros::Duration(2.0).sleep();
if(success)
{
    ROS_INFO("Moving to target %f, %f, %f", x, y, z);
    group.execute(my_plan);
}

ros::Duration(3.0).sleep();

moveit::planning_interface::MoveGroup::Plan my_plan2;
x = 0.8;
y = 0;
z = 0.4;
ROS_INFO("Planning for target %f, %f, %f", x, y, z);
pose.position.x = x;
pose.position.y = y;
pose.position.z = z;
pose.orientation.x = roll;
pose.orientation.y = pitch;
pose.orientation.z = yaw;
pose.orientation.w = 1.0;
group.setPoseTarget(pose, "");
success = group.plan(my_plan2);
while(!success)
{
    ros::Duration(1.0).sleep();
    success = group.plan(my_plan2);
}
ROS_INFO("Planning result: %i", success);
ros::Duration(2.0).sleep();
if(success)
{
    ROS_INFO("Moving to target %f, %f, %f", x, y, z);
    group.execute(my_plan2);
}

ros::Duration(3.0).sleep();

/*
x = 0.2;
y = -0.4;

```

```

        z = 0.7;
        ROS_INFO("Planning for target %f, %f, %f", x, y, z);
        group.setPositionTarget(x, y, z, "");
        success = group.plan(my_plan);
        ROS_INFO("Planning result: %i", success);
        ros::Duration(2.0).sleep();
        if(success)
        {
            ROS_INFO("Moving to target %f, %f, %f", x, y, z);
            group.execute(my_plan);
        }

//        ros::Duration(3.0).sleep();

        x = 0;
        y = -0.6;
        z = 0.5;
        ROS_INFO("Planning for target %f, %f, %f", x, y, z);
        group.setPositionTarget(x, y, z, "");
        success = group.plan(my_plan);
        ROS_INFO("Planning result: %i", success);
        ros::Duration(2.0).sleep();
        if(success)
        {
            ROS_INFO("Moving to target %f, %f, %f", x, y, z);
            group.execute(my_plan);
        }

//        ros::Duration(3.0).sleep();
    */
    i++;
}
return 0;
}

```

Appendix 4.8 Place object

```
#include <ros/ros.h>
#include <visualization_msgs/Marker.h>
#include <moveit_msgs/CollisionObject.h>
#include <geometric_shapes/shapes.h>
#include <geometry_msgs/Pose.h>
#include <string.h>
#include <prj7_vision_test/prj7_box.h>
#include <math.h>
```

```
//Currend fov is 90x65cm at 1024x768 -> 0.9mm/pixel
```

```
ros::Publisher collis_pub;
```

```
std::string frame_id_ = "base_link";
```

```
void marker_place_block(ros::Publisher vis_pub, int id, float x_scale, float y_scale, float z_scale, float x, float y, float z)
```

```
{
    visualization_msgs::Marker marker;
    marker.header.frame_id = frame_id_;
    marker.header.stamp = ros::Time();
    marker.ns = "my_namespace";
    marker.id = id;
    marker.type = visualization_msgs::Marker::CUBE;
    marker.action = visualization_msgs::Marker::ADD;
    marker.pose.position.x = x;
    marker.pose.position.y = y;
    marker.pose.position.z = z;
    marker.pose.orientation.x = 0.0;
    marker.pose.orientation.y = 0.0;
    marker.pose.orientation.z = 0.0;
    marker.pose.orientation.w = 1.0;
    //scale = size, scale 1 = 1m
    marker.scale.x = x_scale;
    marker.scale.y = y_scale;
    marker.scale.z = z_scale;
    marker.color.a = 1.0; // Don't forget to set the alpha!
    marker.color.r = 1.0;
    marker.color.g = 0.0;
    marker.color.b = 0.0;
    vis_pub.publish( marker );
}
```

```
void collision_place_block(ros::Publisher collis_pub, int id, float x_scale, float y_scale, float z_scale, float x, float y, float z, float rotation)
```

```

{
    //add the box into the collision space
    moveit_msgs::CollisionObject col_box_msg;
    geometry_msgs::Pose pose;
    col_box_msg.id = "collision_box_" + std::to_string(id);
    col_box_msg.header.frame_id = frame_id_;
    col_box_msg.operation = moveit_msgs::CollisionObject::ADD;
    shape_msgs::SolidPrimitive box;
    box.type = shape_msgs::SolidPrimitive::BOX;
    box.dimensions.push_back(x_scale);
    box.dimensions.push_back(y_scale);
    box.dimensions.push_back(z_scale);

    pose.position.x = x;
    pose.position.y = y;
    pose.position.z = z;
    pose.orientation.x = 0;
    pose.orientation.y = 0;
    pose.orientation.z = rotation * (M_PI / 180);
    pose.orientation.w = 1.0;

    col_box_msg.primitives.push_back(box);
    col_box_msg.primitive_poses.push_back(pose);

    collis_pub.publish(col_box_msg);
}

void collision_remove_block(ros::Publisher collis_pub, int id)
{
    moveit_msgs::CollisionObject col_box_msg;
    col_box_msg.id = "collision_box_" + std::to_string(id);
    col_box_msg.header.frame_id = frame_id_;
    col_box_msg.operation = moveit_msgs::CollisionObject::REMOVE;

    collis_pub.publish(col_box_msg);
}

void collision_move_block(ros::Publisher collis_pub, int id, float x, float y, float z)
{
    //add the box into the collision space
    moveit_msgs::CollisionObject col_box_msg;
    geometry_msgs::Pose pose;
    col_box_msg.id = "collision_box_" + std::to_string(id);
    col_box_msg.header.frame_id = frame_id_;
    col_box_msg.operation = moveit_msgs::CollisionObject::MOVE;

    pose.position.x = x;

```



```

    pose.position.y = y;
    pose.position.z = z;
    pose.orientation.w = 1.0;

    col_box_msg.primitive_poses.push_back(pose);

    collis_pub.publish(col_box_msg);

}

void objectCallback(const prj7_vision_test::prj7_box::ConstPtr& msg)
{
    collision_remove_block(collis_pub, msg->id);
    collision_place_block(collis_pub, msg->id, msg->width * 0.001, msg->height * 0.001, 2, msg->x *
0.001, msg->y * 0.001, 0, msg->rotation);
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "prj7_place_object");

    ros::AsyncSpinner spinner(1);
    spinner.start();

    ros::NodeHandle n;
    ros::Publisher vis_pub = n.advertise<visualization_msgs::Marker>("visualization_marker", 0);
    collis_pub = n.advertise<moveit_msgs::CollisionObject>("collision_object", 0);
    ros::Duration(2.0).sleep();// This delay is so critical, otherwise the first published object may not
be added in the collision_space by the environment_server
    ros::Subscriber obj_sub = n.subscribe("prj7_box_bluefox", 1000, objectCallback);

    while(ros::ok())
    {
        ROS_INFO("Cleaning up objects");
        for(int i=0; i<200; i++)
        {
            collision_remove_block(collis_pub, i);
        }
        ros::Duration(1.0).sleep();
    }
    return 0;
}

```

Appendix 4.9 Place walls

```
#include <ros/ros.h>
#include <visualization_msgs/Marker.h>
#include <moveit_msgs/CollisionObject.h>
#include <geometric_shapes/shapes.h>
#include <geometry_msgs/Pose.h>
#include <string.h>
#include <prj7_vision_test/prj7_box.h>
#include <math.h>
```

```
//Currend fov is 90x65cm at 1024x768 -> 0.9mm/pixel
```

```
ros::Publisher collis_pub;
```

```
std::string frame_id_ = "base_link";
```

```
void marker_place_block(ros::Publisher vis_pub, int id, float x_scale, float y_scale, float z_scale, float x, float y, float z)
```

```
{
    visualization_msgs::Marker marker;
    marker.header.frame_id = frame_id_;
    marker.header.stamp = ros::Time();
    marker.ns = "my_namespace";
    marker.id = id;
    marker.type = visualization_msgs::Marker::CUBE;
    marker.action = visualization_msgs::Marker::ADD;
    marker.pose.position.x = x;
    marker.pose.position.y = y;
    marker.pose.position.z = z;
    marker.pose.orientation.x = 0.0;
    marker.pose.orientation.y = 0.0;
    marker.pose.orientation.z = 0.0;
    marker.pose.orientation.w = 1.0;
    //scale = size, scale 1 = 1m
    marker.scale.x = x_scale;
    marker.scale.y = y_scale;
    marker.scale.z = z_scale;
    marker.color.a = 1.0; // Don't forget to set the alpha!
    marker.color.r = 1.0;
    marker.color.g = 0.0;
    marker.color.b = 0.0;
    vis_pub.publish( marker );
}
```

```
void collision_place_block(ros::Publisher collis_pub, int id, float x_scale, float y_scale, float z_scale, float x, float y, float z, float rotation)
```

```

{
    //add the box into the collision space
    moveit_msgs::CollisionObject col_box_msg;
    geometry_msgs::Pose pose;
    col_box_msg.id = "collision_box_" + std::to_string(id);
    col_box_msg.header.frame_id = frame_id_;
    col_box_msg.operation = moveit_msgs::CollisionObject::ADD;
    shape_msgs::SolidPrimitive box;
    box.type = shape_msgs::SolidPrimitive::BOX;
    box.dimensions.push_back(x_scale);
    box.dimensions.push_back(y_scale);
    box.dimensions.push_back(z_scale);

    pose.position.x = x;
    pose.position.y = y;
    pose.position.z = z;
    pose.orientation.x = 0;
    pose.orientation.y = 0;
    pose.orientation.z = rotation * (M_PI / 180);
    pose.orientation.w = 1.0;

    col_box_msg.primitives.push_back(box);
    col_box_msg.primitive_poses.push_back(pose);

    collis_pub.publish(col_box_msg);
}

void collision_remove_block(ros::Publisher collis_pub, int id)
{
    moveit_msgs::CollisionObject col_box_msg;
    col_box_msg.id = "collision_box_" + std::to_string(id);
    col_box_msg.header.frame_id = frame_id_;
    col_box_msg.operation = moveit_msgs::CollisionObject::REMOVE;

    collis_pub.publish(col_box_msg);
}

void collision_move_block(ros::Publisher collis_pub, int id, float x, float y, float z)
{
    //add the box into the collision space
    moveit_msgs::CollisionObject col_box_msg;
    geometry_msgs::Pose pose;
    col_box_msg.id = "collision_box_" + std::to_string(id);
    col_box_msg.header.frame_id = frame_id_;
    col_box_msg.operation = moveit_msgs::CollisionObject::MOVE;

    pose.position.x = x;

```

```

    pose.position.y = y;
    pose.position.z = z;
    pose.orientation.w = 1.0;

    col_box_msg.primitive_poses.push_back(pose);

    collis_pub.publish(col_box_msg);

}

void collision_place_wall(ros::Publisher collis_pub, int id, float x_scale, float y_scale, float z_scale, float
x, float y, float z, float rotation)
{
    //add the box into the collision space
    moveit_msgs::CollisionObject col_box_msg;
    geometry_msgs::Pose pose;
    col_box_msg.id = "collision_wall_" + std::to_string(id);
    col_box_msg.header.frame_id = frame_id_;
    col_box_msg.operation = moveit_msgs::CollisionObject::ADD;
    shape_msgs::SolidPrimitive box;
    box.type = shape_msgs::SolidPrimitive::BOX;
    box.dimensions.push_back(x_scale);
    box.dimensions.push_back(y_scale);
    box.dimensions.push_back(z_scale);

    pose.position.x = x;
    pose.position.y = y;
    pose.position.z = z;
    //pose.orientation.x = x;
    //pose.orientation.y = y;
    //pose.orientation.z = z;
    pose.orientation.w = 0;

    col_box_msg.primitives.push_back(box);
    col_box_msg.primitive_poses.push_back(pose);

    collis_pub.publish(col_box_msg);

}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "prj7_place_walls");

    ros::AsyncSpinner spinner(1);
    spinner.start();

    ros::NodeHandle n;

```

```
ros::Publisher vis_pub = n.advertise<visualization_msgs::Marker>( "visualization_marker", 0 );
collis_pub = n.advertise<moveit_msgs::CollisionObject>("collision_object", 0);
ros::Duration(2.0).sleep();// This delay is so critical, otherwise the first published object may not
be added in the collision_space by the environment_server
```

```
collision_place_wall(collis_pub, 0, 2, 0.1, 2, 0.4, -0.7, 0, 0);
collision_place_wall(collis_pub, 1, 2, 0.1, 2, 0.4, 0.7, 0, 0);
collision_place_wall(collis_pub, 2, 0.1, 2, 2, -0.6, 0, 0, 0);
collision_place_wall(collis_pub, 3, 2, 2, 0.1, 0, 0, 0, 0);

while(ros::ok())
{

}
return 0;
}
```

Appendix 4.10 Place 3D object

```
#include <ros/ros.h>
#include <visualization_msgs/Marker.h>
#include <moveit_msgs/CollisionObject.h>
#include <geometric_shapes/shapes.h>
#include <geometry_msgs/Pose.h>
#include <string.h>
#include <prj7_vision_test/prj7_box.h>
#include <math.h>

ros::Publisher collis_pub;

std::string frame_id_ = "base_link";

float bluefox_box[1000][5];
float usb_box[1000][5];

void marker_place_block(ros::Publisher vis_pub, int id, float x_scale, float y_scale, float z_scale, float x,
float y, float z)
{
    visualization_msgs::Marker marker;
    marker.header.frame_id = frame_id_;
    marker.header.stamp = ros::Time();
    marker.ns = "my_namespace";
    marker.id = id;
    marker.type = visualization_msgs::Marker::CUBE;
    marker.action = visualization_msgs::Marker::ADD;
    marker.pose.position.x = x;
    marker.pose.position.y = y;
    marker.pose.position.z = z;
    marker.pose.orientation.x = 0.0;
    marker.pose.orientation.y = 0.0;
    marker.pose.orientation.z = 0.0;
    marker.pose.orientation.w = 1.0;
    //scale = size, scale 1 = 1m
    marker.scale.x = x_scale;
    marker.scale.y = y_scale;
    marker.scale.z = z_scale;
    marker.color.a = 1.0; // Don't forget to set the alpha!
    marker.color.r = 1.0;
    marker.color.g = 0.0;
    marker.color.b = 0.0;
    vis_pub.publish( marker );
}

void collision_place_block(ros::Publisher collis_pub, int id, float x_scale, float y_scale, float z_scale, float
x, float y, float z, float rotation)
```

```

{
    //add the box into the collision space
    moveit_msgs::CollisionObject col_box_msg;
    geometry_msgs::Pose pose;
    col_box_msg.id = "collision_box_" + std::to_string(id);
    col_box_msg.header.frame_id = frame_id_;
    col_box_msg.operation = moveit_msgs::CollisionObject::ADD;
    shape_msgs::SolidPrimitive box;
    box.type = shape_msgs::SolidPrimitive::BOX;
    box.dimensions.push_back(x_scale);
    box.dimensions.push_back(y_scale);
    box.dimensions.push_back(z_scale);

    pose.position.x = x;
    pose.position.y = y;
    pose.position.z = z;
    pose.orientation.x = 0;
    pose.orientation.y = 0;
    pose.orientation.z = rotation * (M_PI / 180);
    pose.orientation.w = 1.0;

    col_box_msg.primitives.push_back(box);
    col_box_msg.primitive_poses.push_back(pose);

    collis_pub.publish(col_box_msg);
}

void collision_remove_block(ros::Publisher collis_pub, int id)
{
    moveit_msgs::CollisionObject col_box_msg;
    col_box_msg.id = "collision_box_" + std::to_string(id);
    col_box_msg.header.frame_id = frame_id_;
    col_box_msg.operation = moveit_msgs::CollisionObject::REMOVE;

    collis_pub.publish(col_box_msg);
}

void collision_move_block(ros::Publisher collis_pub, int id, float x, float y, float z)
{
    //add the box into the collision space
    moveit_msgs::CollisionObject col_box_msg;
    geometry_msgs::Pose pose;
    col_box_msg.id = "collision_box_" + std::to_string(id);
    col_box_msg.header.frame_id = frame_id_;
    col_box_msg.operation = moveit_msgs::CollisionObject::MOVE;

    pose.position.x = x;

```

```

    pose.position.y = y;
    pose.position.z = z;
    pose.orientation.w = 1.0;

    col_box_msg.primitive_poses.push_back(pose);

    collis_pub.publish(col_box_msg);

}

void objectCallbackBluefox(const prj7_vision_test::prj7_box::ConstPtr& msg)
{
    /*    collision_remove_block(collis_pub, msg->id);
        collision_place_block(collis_pub, msg->id, msg->width * 0.0009, msg->height * 0.0009, 2, msg-
        >x * 0.0009, msg->y * 0.0009, 0, msg->rotation); */

    bluefox_box[msg->id][0] = msg->x;
    bluefox_box[msg->id][1] = msg->y;
    bluefox_box[msg->id][2] = msg->width;
    bluefox_box[msg->id][3] = msg->height;
    bluefox_box[msg->id][4] = msg->rotation;
}

void objectCallbackUsb(const prj7_vision_test::prj7_box::ConstPtr& msg)
{
    /*    collision_remove_block(collis_pub, msg->id);
        collision_place_block(collis_pub, msg->id, msg->width * 0.0009, 0.1, msg->height * 0.0009, msg-
        >x * 0.0009, 0, msg->y * 0.0009, msg->rotation); */

    usb_box[msg->id][0] = msg->x;
    usb_box[msg->id][1] = msg->y;
    usb_box[msg->id][2] = msg->width;
    usb_box[msg->id][3] = msg->height;
    usb_box[msg->id][4] = msg->rotation;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "prj7_place_object");

    ros::AsyncSpinner spinner(1);
    spinner.start();

    ros::NodeHandle n;
    ros::Publisher vis_pub = n.advertise<visualization_msgs::Marker>("visualization_marker", 0);
    collis_pub = n.advertise<moveit_msgs::CollisionObject>("collision_object", 0);
    ros::Duration(2.0).sleep();// This delay is so critical, otherwise the first published object may not
    be added in the collision_space by the environment_server

```



```

    ros::Subscriber obj_sub_bluefox = n.subscribe("prj7_box_bluefox", 1000,
objectCallbackBluefox);
    ros::Subscriber obj_sub_usb = n.subscribe("prj7_box_usb", 1000, objectCallbackUsb);

    while(ros::ok())
    {
/*          //ROS_INFO("Cleaning up objects");
        for(int i=5; i<200; i++)
        {
            collision_remove_block(collis_pub, i);
        }
        ros::Duration(1.0).sleep();
*/
        for(int i = 0; i<1000; i++)
        {
            for(int j = 0; j<1000; j++)
            {
                if(abs(bluefox_box[i][0] - usb_box[j][0]) < 2)
                {
                    ROS_INFO("Matching x coordinates found @ x=%f.",
bluefox_box[i][0]);
                    break;
                }
            }
        }
    }
    return 0;
}

```