

Design and Implementation of a Model-based Architecture for Cobotic Cells

Nikhil Ambardar

nikhil.ambardar@tu-dresden.de

Born on: 5th November 1989 in Mumbai, India

Course: Distributed Systems Engineering

Matriculation number: 4667008

Matriculation year: 2016

Master Thesis

to achieve the academic degree

Master of Science (M.Sc.)

Supervisors

Dr. Sebastian Götz ,

Dipl.-Inf. Johannes Mey , and

Dipl.-Inf. Sebastian Ebert

Supervising professor

Prof. Dr. rer. nat habil. Uwe Aßmann

Submitted on: 27th October 2020

Contents

1	INTRODUCTION	4
1.1	Robots and Cobots	4
1.2	Components Of a Typical Robot	6
1.3	Uses And Applied Fields	7
1.3.1	Elderly Care	7
1.3.2	Medical Uses	7
1.3.3	Warehouse Operator	7
1.3.4	Food Home Delivery	8
1.3.5	Automotive Industry	8
1.4	Importance Of Robots In Today's Time	8
1.5	Expectations From Robots	9
1.6	Evolution - Robots to Cobots	9
2	INSPIRATION AND DRIVING FORCE	10
3	BACKGROUND	12
3.1	About Franka Emika Panda Robot	12
3.2	Robotics - Features	13
3.2.1	Sensitivity	13
3.2.2	Drive a.k.a Motion	14
3.2.3	Impedance	14
3.2.4	Collision Detection and Reaction	14
3.3	Software Tools Robot Franka Uses	15
3.4	Robotic Coexistence With Humans - Meaning Of Cobots	15
3.4.1	Existence Alongside Humans	16
3.4.2	Collaboration and Co-operation	16
3.4.3	Real-time and Presence Acknowledged Collaboration	16
3.5	Accidents Due To Malfunctions and Consequences	16
3.5.1	Mechanical Failure	17
3.5.2	Electrical Anomaly In Components	17
3.5.3	Malfunctioning Software	17
3.5.4	Human Operator Errors	17
3.6	Making Robots Safer And Safe Deployment Practices	18
4	STATE OF THE ART	20
4.1	Motion Planning And Simulations	20

4.2	Modeling - Explaining Choice of Design Depictions	20
4.2.1	Unified Modeling Language (UML) Diagram For World Model Class Diagram	21
4.2.2	Business Process Modeling Notation (BPMN) for Application Model	23
4.2.3	Unified Modeling Language State Machine for Safety Model . .	25
4.3	Tools Used	26
4.3.1	Setup Environment Using ROS To Run Services, Motion Planning In MOVEIT and GAZEBO For Simulations and more	26
5	CONCEPT	32
5.1	Theme	32
5.2	The Models Designed	33
5.3	Programming Of Hardware And Software Components - Purpose And Concepts	37
5.4	Refactoring and Reclassification	38
5.5	Connecting Everything - Simulations	38
6	IMPLEMENTATION	40
6.1	Introduction	40
6.2	WORLD MODEL IMPLEMENTATION	41
6.2.1	Programming - The Implementation Of World Model	42
6.3	Future Work Implementing Application Model Programmatically	45
7	EVALUATION	46
7.0.1	World Model Implementation Results	46
7.1	World Model Test Implementation	46
	Bibliography	48

1 INTRODUCTION

Objective Of Thesis: Design and Implementation of a Model-Based Architecture for Cobotic Cells.

1.1 Robots and Cobots

A robot is a machine developed by humans to do work for them and to make their lives easy. Humans had to keep doing repetitive tasks throughout history which were essential things to do and there have been times when enough manpower was absent or unavailable to do the required work. This led to humans to think about designing a machine which can obey their commands to accomplish a series of steps to do a job. They then described these machines as a robot.

In fact, any machine designed for any specific task and purpose can be termed as a robot. But the term "robot" is generally described as a machine that is versatile and can accomplish a range of tasks. The robot is characterized by few qualities like motion in the body of robot i.e. it must be able to move itself, motion of control surfaces like arm, robot must be able to do the task it is designed for, like pickup and place down objects, or be able to work at a factory assembly line, at the same time must be able to provide safety to some degree. Robots must provide some or all of these qualities according to the purpose it is designed for. There are other aspects which can be adding and removing control surfaces to suit a specific type of job or size of area of field of the work profile.

Along the evolution of time the machines developed from designing basic tools to varying degrees of complexities like space robots. As decades and centuries of time passed by, human thinking evolved and refined, their outlook towards machines improved and so did their skills with machines to make them more self contained and all this was done to reduce human intervention for decision making. To start with these were elementary forms of machines which are principally described as robots. Nowadays humans are trying to build in artificial intelligence into robots to make them self reliant. One example is self driving cars which are being designed. In the future these cars will have a high level of artificial intelligence built into them which will make them very reliable performer.

New features were built into machines to add functionality and this was done using

physical mechanisms in early days like using valves for changing and redirecting water supply in pipes and tracks change railroad paths using levers. But as times changed and many levels of developments happened for physical machines using some form of intelligence then came the era of information technology and development of software systems and programming which was used to control hardware until the last level of action. This was done using a host of components added to hardware which linked it to software systems and they were first circuits which then turned to micro-controllers, and then to electronic components that could be programmed and integrated with physical systems which were then used to manipulate control surfaces thus forming high level of evolution in robotics [6](Chapter1).

Robotics is a relatively new and evolved technical field. It is an evolved version of technical development of machines and software field which are both part of robotics. It won't be wrong to claim that robotics is epitome of technical development in machines. Nonetheless robotics is still a very expensive field of research. Robots have a high one time cost as well as very high operational, development and research costs and if in a broken down condition can become expensive to fix. It is still evolving and studied as a modern research area and it is continuously expanding and acting as a base to develop other technologies. Study of robotics is only available to a lucky few people and research is done every day to make it affordable and accessible to more general people. There are many industrial organizations dealing with robotics field and they are doing extensive research for it.

This has nowadays led to creation of an ecosystem of machines which are versatile and follow hook and template structure to do many tasks using same kind of concept i.e technology was used for multiple purposes thus adding to versatility. This has added to functionalities of robots and made them much more versatile and cost efficient as they can be used to perform multiple jobs. This is akin to using different attachments on a modern day vacuum cleaning machine to the the same job of vacuuming by suction. Robots in today's time are meant to do easiest of tasks like moving objects around and giving company to elderly, to complex tasks like critical surgical operations on humans in medicine and working on automobile manufacturing and assembly line [41].

Robots are very essential to some of the today's modern industry because they help humans accomplish tasks which otherwise are too difficult to perform with the speed and accuracy with which robots can do them. In addition robots can perform tirelessly and do not cringe if they work in day or night. This makes them a versatile tool that is friendly and beneficial to humans. The robots thus need to live, work and perform alongside humans in most cases. This is usually critical in factories and also homes where they are used nowadays. These robots maybe very intelligent and this makes them very capable but in most cases they cannot completely replace human presence which is still required to monitor robotic operations and sometimes humans are required to alter plans and make decisions as per orders and this gives birth to a situation where robots need to work alongside humans and this involves only safe operations because in the event robots cause injury to human it can cause very serious situations and hazard to human well being and may also cause death of a person [36]. Here comes the concept for cobots. They are robots which are built to perform alongside humans and this too safely. The word cobot is derived from so called collaborative robots which are kind of robots made to perform alongside human presence in a commonly used area. Most times they are in very close proximity working alongside each other but the design decisions and multi level safety which range from soft built to

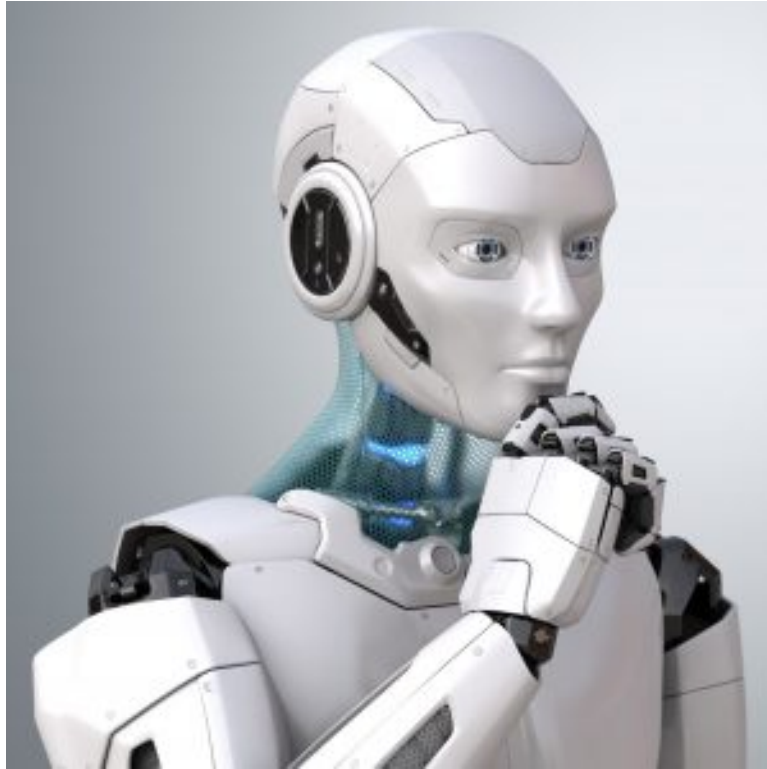


Figure 1.1 Image Source - Footnote 1

auto-emergency stop features ensure they can work very reliably alongside each other.

1

1.2 Components Of a Typical Robot

Any robot is a machine that is made up of mainly steel, plastic and more complex materials put together and typically needs electric power to run and perform some work. Robots are programmed using computer software in today's times. Software dictate the robotic parameters and manipulate control surfaces by reading sensor values to enable decision making as per the situation detected and decisive action pre-programmed for it. This can be used to fine tune its actions to perform most complex of tasks independently [22](Chapter1). Robots in today's times have embedded control built in, which guide the robot to do a job safely in real world environment setting.

Robots range from auto/semi-automatic functional form to resembling human form type to industrial grade and medically utilized robots. Robots have been made to look and behave exactly like humans in present time and this is achieved by using soft silicone materials which makes them look sophisticated and intelligent in a home setting. But they are not always designed for good looks and in industry setting they are bare bone machines most time lacking even correct covering and this is to aid in maintenance and improve operational functionality. Today robotics is working with cutting edge research enabling it to function up to the level of developing safe self

¹<https://blogs.3ds.com/northamerica/future-robots-and-ensuring-human-safety/>

driving cars [10].

So it can be concluded that robotic components are body/frame, control system, control surfaces, and drive train.

1.3 Uses And Applied Fields

Robots have been developed to suit and work in various fields in today's time. In-fact there is an appropriate type of robot for almost any work today. They range from most easy work to the most complex of tasks. In this section more light on this area is thrown by describing some use cases [29](Chapter1).

1.3.1 Elderly Care

Robots are used in home setting to provide company to lonely elderly and give care to them. There are soft bodied robots made up of silicone used in Japan, and this is a place with sparse population and few people to care for elderly. These machines there help the seniors from keeping track of their medicines to playing music to them as well as talking to them. They move objects for them, monitor their health and even help in distress by calling help for lonely seniors [34].

1.3.2 Medical Uses

Robots are used in regular and critical surgeries performed on humans in today's time. The performance of robots as compared to humans is very high in critical human diagnostic operations like C-section surgeries and much more complex ones like angioplasty.

The can perform medical procedures very fast and with clean cuts and stitches that a human hand is not capable of performing. They can as well perform knee replacement surgeries and many more like making incision to flush kidney stones with amazing accuracy. Many robots also let the surgeon doctor to take control of control surfaces and this aids in minimizing errors caused by a shaking hand a human doctor may have [24].

1.3.3 Warehouse Operator

Robots are essential workers in warehouses. They help to handle parcel items in automated way to pick up the items, stamp them with bar-code stickers and sorting them according to size and destination. They can also read information from parcels and register movement of parcel boxes in the central repository which let a user know exact status of parcel movement. Here they most times need to work alongside humans and under human monitoring. They also move items across the shop floor to a different assembly line [33].

1.3.4 Food Home Delivery

Very recently some tech giants are experimenting with idea of delivering pizza to home. For this drones are used which supposedly fly to an address and safely drop the pizza at a safe location for the customer to pickup. This is already done in practice in some cities. The drone used is also a type of robot which is governed by many technologies like GPS signals, radar proximity sensors and camera to deliver pizza without causing any harm to people around [23].

1.3.5 Automotive Industry

Robots are used in automobile manufacturing assembly lines to produce high quality automobiles. They can minimize errors when compared to using a human working style and give high quality finished products that has high error possibility when manufactured by a human. Many global automobile brands are known for their high quality cars all over the world and they use robots to manage the process of manufacturing. Robots pickup parts, move them to appropriate locations, weld or assemble them as per need, and then let a human work alongside to do some things like monitoring or operations like critical assembly [4].

1.4 Importance Of Robots In Today's Time

Robots in today's time offer some very specific and important benefits which cannot be ignored or replaced. It include safety, precision of work, quick delivery of product to market, accuracy in repetitive tasks without human like excuses and many more.

Robots are ideal for uses in high risk area like volcanic explorations, space probes, deep ocean operations and in bomb diffusion squads. In these situations they perform extraordinarily well. They are stable workers without boredom, cannot get tired, don't compromise on safety and don't make excuses for under performance. They work tirelessly and ensure accuracy, precision and quality of work in any situation. Once programmed and up and running they can work a long time without supervision and while maintaining standards of their work under all situation.

Robots are used to look for debris in ocean floor from the wreckage of a sunken ship or crashed air-crafts. There may be an deep ocean surface study exploration or a study on largely unknown aquatic life which can only be performed using robots. Robots are imperative to do such operations.

In cases of metal and mineral exploration and study on ocean floors, robots are machines that cannot be replaced and thus are credited with numerous discoveries. They can move control surfaces and transmit images of seafloor and let humans take control of situation on ocean floor from several miles distance on ocean top. The pressure present at ocean floor can only be tolerated by a submarine and sending a human in deep depths is not possible. Similarly robots are used in space probes and space missions where they do operations humans are not capable of doing like exposing themselves to work in open space where harsh UV sunlight can cause bad effects on human body and any exposure of human beings there is only limited for critical operations to minimize risks and high costs associated to cover the risks. They provide safety to humans and make work easier and faster along with giving reliability

and required precision [35] [43] [17].

Robots are used around volcanoes erupting molten lava to collect samples and study materials and these areas have very high temperatures. These are tasks that are impossible and too hazardous to be done by humans thus making robot a tool that cannot be replaced.

Employing robots is cheap and needs only power which is also highly optimized. Robots have a one time costs and low power consumption which make them cheaper to employ in most cases than humans. They are also very reliable nowadays thus offering very low maintenance and great value for money.

Robots are intelligent. They are programmed to make their own decisions and know how to tackle almost all situations. They have a learning mode where they can be taught instantaneously some tasks that they can mimic, but this is usually limited for research purposes in university setting.

1.5 Expectations From Robots

There are few basic expectation that a Robot must in all cases adhere to and they are called as Laws of Robotics which define these three expectations:described as Asimov's Laws of Robotics. They are described as follows :

- A robot must not in any case cause a minor or hazardous injury to a human being or allow the injury of a human being due to inactivity.
- A robot under any and all circumstance must obey the orders which are given by humans except of those that conflict with the First Law stated above.
- A robot must protect its existence unless in a situation in conflicts with the First or Second Law stated above [39].

1.6 Evolution - Robots to Cobots

Most use cases defined in the above sections have the need for a robot to work alongside a human. This is a necessary step in the evolution of robots and as per today's need for robotic performance. The next step in robot evolution is the introduction of the term "cobot". Cobots are nothing but robots but the ones those are made to work alongside robots. This has become a need for most robot use cases these days as the present and future of robots is to work together with humans. Cobots have many features of safety built into them which enables them to operate alongside humans without causing any injury to humans. The features range from object and obstacle detection to a complete stop when in close proximity to a human [12].

2 INSPIRATION AND DRIVING FORCE

The topic of this thesis is about design and implementation of a model based architecture for cobotic cells. With the advent of tactile internet, regularizing coexistence of robots and humans has become imperative, meaning the so called "cobots", need a new use case architecture for its unit cell to operate safely alongside humans and real world objects and obstacles. This architecture is based on multiple models each describing one aspect of use case aiding in functionality of cobots. For this the thesis described three models namely world model, application model and safety model which are described using different notations.

The world model is a global model describing the cobot and other things in its environment, giving "on the whole" information about the components in real world a cobot has, this includes one or more humans who can be moving in and out of cobotic world zone, then some obstacles and grasp object which can be a ball or cube.

The application model describes the flow of individual actions of grasping that can be performed by cobot according to a motion trajectory to accomplish the given task. This model is all about performing the task and action of the cobot. Lastly, the safety model shows how a cobot achieves the goal of not causing any harm to humans or other objects in its proximity and how to respond to them by moving around them appropriately in cases imminent collisions are detected.

The real life problem scenario can be described as follows. Robot is expected to perform some job and to make it to do that with safety i.e. detect and evade obstacles / humans, this safety and application can be achieved in two different step cases. The models designed and described ,address to this task or problem of first, to train the robot for performing actions according to a preconceived plan using inbuilt "teaching" feature of robot and then doing it safely in real world conditions.

The use case can be understood by seeing a scenario where we can train the robot in a laboratory / ideal conditions and give a working functionality to it by giving a design which shows how to perform a task which robot can use to work accordingly and this is known as application model implementation. This robot has a teaching mode where we can set a series of poses and grasp actions manually which can train the robot to perform a task according to a plan and this can be done repetitively

by the robot later in scenario 2 which is real world and has added conditions of realism.

For this ,complex condition are added to application model about how to respond when it detects a human in proximity and obstacles in trajectory paths and in addition this real world simulation adds real world conditions like adding torque to joints as is in real world to see if arm can for example really life an object.

In scenario one the architecture of robot's world model is already known and has thus been used ,its teaching capability to train it to move to a coordinate position and then start a trajectory for instance at position X to move a position close to an object that is needed to be say picked up and then it can use its gripper to pickup the object and again move arm to another desired location where it want to drop the object and there it releases the gripper to put that object down and thus completing the task at position Y. This is part of application model as described before.

This is smaller use case replication of saying a robot actually moved but here the idea is restricted to only moving arm which is the same when it comes to functionality achieved by robot moving itself vs moving its arm as previously mentioned, and this is fulfilling the same work of detecting things in proximity and achieving the tasks by completing trajectory as well as at same time to do it safely by responding appropriately as per intended use case programmed for safety.

So far above description talks about training the robot in scenario 1 and now another scenario is considered which is a real world task where the robot is made to perform the same work it was trained in Scenario 1 but in real life and this means the safety aspect should now be built into the scenario and for this a safety architecture is constructed which is used by robot, by telling it how to respond when seeing an obstacle like a cube or box for example or a human being.

This sets the tone for the work for this thesis.

3 BACKGROUND

3.1 About Franka Emika Panda Robot

Franka Emika GmbH, a technologically sound company from Munich, Germany, has come ahead to address to this issue of high costs and to provide a solution by introducing an affordable and cheap robot known as robot Panda. This is a sensitive and multifaceted machine made available to research communities in universities and for students to learn and experiment with robots. Robot Panda is a part of structural ecosystem of new age robots which are cheap to buy, program and function and are developed with the main objective as a research robotic machine made available to fiddle and learn by students in universities. Its second objective is to introduce its presence as a co-worker in a factory who work along humans in a hybrid mode model and then thirdly as an friend and helping assistant in life for lonely seniors and sick people needing basic help and assistance [14].

There are various interfaces and tutorial series made available for robot Franka which help to manipulate it. In addition there is an ecosystem of repositories used to run and manipulate the robot. Nowadays even smartphone apps are developed to give idea of robotics to students. The robot is made with many features, notable being a learning capability, where in there is a learning mode which enables robot to learn a series of poses and grasp actions which can then be run and replicated. This is a feature used as a research subject. In addition there are many tools used to run the Franka ecosystem like Moveit / Rviz i.e. used for motion planning and the robot also uses Gazebo simulator. The robot Panda relies upon the Robot Operating System as the underlying operating system which is used to run it in background. The tools which are used to manipulate robot Panda like MoveIT / Rviz only use ROS to connect to the robot and command and control it.

Robot Panda is a very sensitive machine and this give it immense capability to do tasks few other machines can do. It has torque sensing framework which can help manipulate the arm very precisely. Robot Panda is also a very safe robot machine and there are many safety features built inside it, like stop button that can be used to shut down the robot. There is as well a research community and many forums which can help educate and guide students and researchers working in the field of robotics. This as well help people to share knowledge and development with each other and develop more functionality allowing a greater experimentation with this machine.

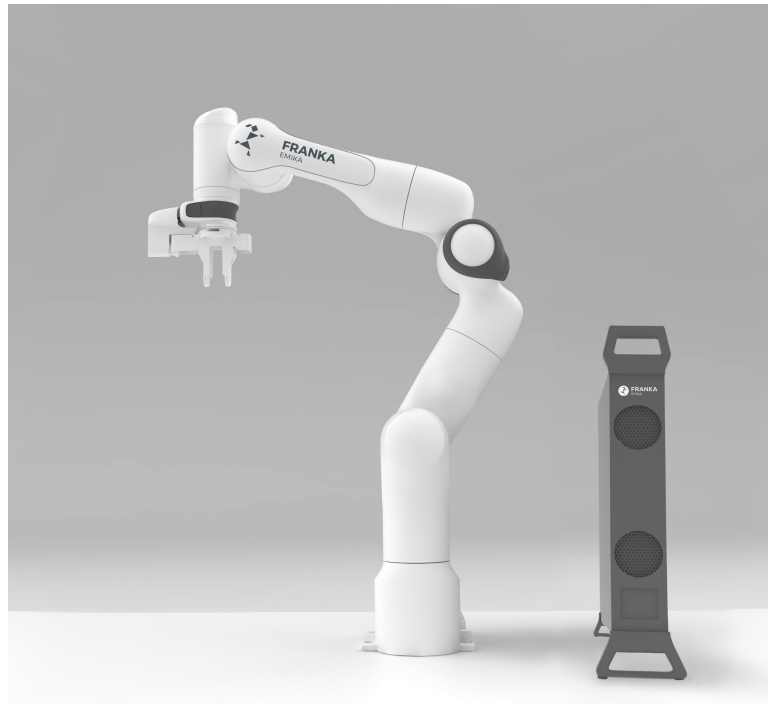


Figure 3.1 Robot Franka Emika Panda Image Source - Footnote 2

There are many more robots developed but this thesis describes only about the robot Franka Emika Panda. Here is what robot Panda looks like.

1

3.2 Robotics - Features

Robots are characterized by a common set of features. Few general features of robots are -

3.2.1 Sensitivity

Robots in today's time are characterized by an essential property of sensitivity and this makes up the idea of cobots. Robots inherently most times may not have this property, but not having this property limits their usage. In today's day and age traditional robots are obsolete as they were not sensitive enough to presence of a human which limited them as machines that were not suitable to work alongside humans. Present day belongs to cobots which have the feature of sensitivity built in. This ranges from using soft material for physical construction of robot and this is in case if cobot hits a human, the soft body and surface can limit the damage to a human being. Then there are built in features in cobots that enable them to be sensitive for its surroundings. This would mean it must sense its surrounding using host of onboard sensors to detect objects of different types and only act accordingly. A robot use a range of different sensors like light sensors in camera, pressure sensors in arms, chemical sensors to detect and measure for instance air quality, listening sensors to adjust speaking volume, radar for scanning surroundings in 3D for precision mappings, and many more as per the

¹2. <https://robots-blog.com/2016/05/10/franka-emika-everybodys-robot/>

need of the job or features to be integrated. Other features include using a slow motion mode when a human is detected in proximity to an emergency stop feature once a close obstacle is detected. Some of sensitivity features are part of materials handling [26].

Robot Franka Emika Panda is having an arm which is made up of seven joints and all have torque sensors giving robot Panda great sensitivity. This allows robot Panda to detect any kind of forces acting on its arm upto the level that it can measure them and detect exact spots where and on which joint, how much torque is applied.

3.2.2 Drive a.k.a Motion

The main characteristic of any robot is movement. A robot should be able to move itself around and this also means it should be able to move parts like an arm which enables it to perform the tasks. We can call the arm and other moving parts of cobots like a gripper as control surfaces. The motion is characterized as essential property of a cobot as it enables it to accomplish a task. Motion is almost always using remote control or by moving on pre-programmed paths.

If a robot arm cannot move it is not able to accomplish a task or if it cannot move itself in a continuous motion then it is not capable to be considered as a research interest. There are various technologies that are used to drive the robots and its control surfaces like using electronics and hydraulic systems [1]

3.2.3 Impedance

Essential property of a cobot is the built in quality of physical resistance and ability to electronically measure it. It is this property of robot which enables the robotic components to electronically alter its control surfaces inside a range so that it can have a level of tolerance on control surfaces when depressed and when relaxed physically. This uses electric resistance on surfaces to detect forces on control surface which let the robot know its surface have come in contact with an obstacle or item and also measure electronically how much force is applied to it. This property has been developed to even measure how much of a control surface is depressed as it measures the movement of spring or a foam material inside the surface which may be used as appropriate material to buffer the surface. This maybe simply explained as an operation of a mechanical spring. This capability gives a touch of gentleness and awareness when interacting with the objects in environment. This is also similar to the human arm which tenses and relaxes the muscles to adapt performing any task like pushing, depending on the load and situation [14].

3.2.4 Collision Detection and Reaction

A robot of present day usually have quality of anticipating collisions built in and this done by detecting obstacles and identifying them in its path. This is described as collision detection and the associated property is of avoidance. So once the path is detected and an obstacle identified in the path then it must be avoided. This can be achieved by stopping in a safe distance before the actual contact with the obstacle.

Another property is of reaction. This can be described as defining robotic reaction once an obstacle is detected in the path. This includes trajectory modification and alteration to find a new trajectory which maybe around the obstacle or a completely new one. Algorithms are used to achieve this property which are very complex in nature but appropriately modify the path of the robot.

There is another dimension present as well to identify with property of collision detection. The torque sensors present in Panda arm detect torque when then move against an obstacle and once past the defined safe limit, drive the robot to a complete emergency halt [11].

3.3 Software Tools Robot Franka Uses

Robot Panda uses several tools for its functioning. There are general software robot Panda uses which are also used by other machines as well as there are software specifically developed for robot Panda. Robot Panda also comes with several software library packages which let it run on user machines using Robot Operating Systems as platform already installed. The software systems robot Franka Emika Panda uses is MoveIT/Rviz for motion planning. This lets user set various parameters of robot Panda and lets them tweak some of its features to see how the motion plan executes in theory. This MoveIT framework sets tone for robot motion planning activity but the plan visualized here is only a hypothetical plan which may not be possibly executable in real world [25].

This is where there arises a need for another software to check if the motion planning is possible in real world. For this a common software used is Gazebo simulator. This simulator have additional real world parameters like torque on joint which can be altered to see how robot reacts for a motion plan in real world [32].

There maybe cases where a robot cannot really afford to bear weight to pickup an object and this maybe visually possible in MoveIT/Rviz tool as it is just a visualization tool framework but then the same motion plan is run in Gazebo simulator which runs the plan with defined torque on all joints which can as well be altered to see more reactions on robotic arm and this lets user see if the simulation is really possible in real world conditions and thus confirm if the motion plan is real world possible.

3.4 Robotic Coexistence With Humans - Meaning Of Cobots

Cobots are evolved version of traditional robots, and traditional robots are only meant to perform without safety features on both levels comprising physical safety as well as software built detection features giving intelligence to robot. Thus these are the factors which differentiate a robot from a cobot.

These features are built in cobots but traditional robots do not have them. Cobot safety relies on lightweight construction materials, curved edges on control surfaces and no sharp edges but only rounded curves, and comes with inherent limitation of speed and force when working along human presence [26].

Cobots are built for an industry setting workspace and have different hardware and software to run them along with above mentioned features of safety. Some of essential features of cobots are -

3.4.1 Existence Alongside Humans

Cobots and humans work with each other in a common area to accomplish a certain task. A traditional robot is not built to perform alongside human but a cobot is [18].

3.4.2 Collaboration and Co-operation

There are two stages of intricacies for cobots working with humans namely Collaboration and Co-operation. Humans and cobots are meant to collaborate in work. This mean they must work in a common space but not on any one task physically together. The other property is co-operation and this is a more refined stage which all cobots may not have this feature. This means human and cobot work on the same module of machine at the same time in parallel, and both the human and cobot are in motion. This is achieved not only by pre-programming some paths for cobots and humans but also in real time which is explained in next point.

3.4.3 Real-time and Presence Acknowledged Collaboration

The cobots are meant to work alongside humans safely. The capability of real time monitoring and decision making is imperative to a cobot. This mean the cobot should be able to detect and track human presence and motion in real time. A cobot must also have decision making ability upto some level to respond to human changing position that enable it to stay at a safe distance from human movement. This must also give robot ability to alter paths in real time for its intended motion.

This is a part of robotics that is not very well developed and this forms basis of self-driving cars which has not become possible in real world usage but is also evolving very fast.

3.5 Accidents Due To Malfunctions and Consequences

Traditional robots were used in industry setting in the past where humans operated them and sometimes worked alongside them. But there have been many cases where the safety of human is compromised in all areas where robots are present. And this has also cased many accidents and some of them very terrible causing grievous injuries and even deaths in many cases. Even during development of cobots there have been accidents for example many prototypes of self-driving cars have failed to provide safety to other cars and people in proximity on the road. It is safe to acknowledge that this is a big issue with cobots and there are problems present here.

There is always an element of risk in human-cobot interactions. The danger arises when a human may get hit by the robot in motion or robotic arm in motion. A human may also get trapped between robot boy or arm and an wall or iron grill in vicinity. There are many types of hazards that are identified and they maybe dangers caused mechanically i.e faults in the machine, electrical shocks to human body, overheating components causing burns and many more hazards which maybe combination of these. The hazards are studied carefully and robots continuously evolved to make them handle risks and this makes them safer. But it is also necessary to acknowledge that in real world there are some faults that can always happen which may not always be due

to the robot itself but due to wrong operation of robots or due to abuse [5](Chapter 4). Some of them are recognized to be -

3.5.1 Mechanical Failure

Robotic mechanism is made up of components like motors, actuators, connectors and sensors. These can malfunction and may directly or indirectly cause safety issues. A classic example is wrong readings sent by sensors to processor causing it to make a wrong decision. This may cause consequent failure of more components or make them behave unsafely [28].

3.5.2 Electrical Anomaly In Components

Robots are made up of electronic components like wires, circuit boards, microprocessors which age and may malfunction causing robot to behave unexpectedly and unsafely. The insulation material may wear out causing short circuits and heat damage to components [28].

3.5.3 Malfunctioning Software

Software is essential to modern day robots and this need programming. Code in any language always have bugs and more so it may fail altogether and this may cause robot to behave unexpectedly and unsafely. The problems come in all shapes and sizes ranging from robot shutting down unexpectedly ,to short circuits causing heat and even fire risks. Thus the set of instructions governing robot use should be perfect for using use cases they are built for but there are always chances of issues coming up after long time use. One of them maybe hardware is superficially compatible with software instructions [21].

3.5.4 Human Operator Errors

Robots if used in unintended use cases or experimented with beyond their capability can malfunction. A malfunctioning robot is an uncontrolled robot and thus can be a huge hazard for human life. There are some veto power humans have over robots to do things for them in their own way, but this may logically and hence technically contradictory leading to very unsafe situations where critical materials are handled for instance uranium in a nuclear reactor. A classic example is Chernobyl Nuclear Power Plant accident which was caused entirely due to operator errors [31]. Machines are built with applying functionality in mind and not to handle a combination of all use cases and this always make it possible that there may arise situations robot can go against itself or the operator. This is usually caused due to untrained engineers, operators, and users. These people may not be aware of effects of their actions causing machine to fault [15].

Robots are continuously incorporated with artificial intelligence features making them safer every day but this is a subject of research which is continuously developed and it is not possible to make a robot equally intelligent to a human being.

3.6 Making Robots Safer And Safe Deployment Practices

Collaborative robots or cobots are all about latest technology trend that is gathering pace with the advent of all new technology coming up in various fields like self driving cars and manufacturing in factory supply chains. The technology is itself developing and so are its components that can be used interchangeably across industries to develop an ecosystem of new age artificially intelligent cobots at an affordable price. This technology offer amazing advantages as they can safely work alongside humans and provide cost and time benefits that is hard to beat in industry setting where cycle time and productivity are key issues [27].

As was mentioned before that cobotic technology is still evolving and developing and assuming it to come at a level where they behave like a living being may take more decades of time and the example is self driving cars that many prominent organizations are working on but have not been able to successfully integrate that finesse and level of safety so far. There have been accidents with attempts to integrate artificial intelligence in to cobots and using machine learning to train the cobot with experiences and then implement this with assured safety or at least equal to a level of humans decision power. This means that companies are spending large amounts of money to develop such technologies and help them evolve. All because the accidents can be serious and can cause injuries and loss of human life.

There are causes of accidents using robots at workplace or industry setting. Robots were made to be fast workers and also powerful to do tedious tasks which means in most cases the control surfaces have substantial amounts of torque. This can cause injury to a human with just one strike and there may be various situations that can develop leading to an accident and thus hazard to human life and also may cause financial costs and medical attention. There are situations when even non functioning robots may cause incidents and hazards. Example is when a robot may malfunction when it is being overhauled for maintenance. There may be a worker doing overhauling when it may react irregularly and cause serious hazard to life of worker. Or there can arise a situation when a robot may be faulty suppose due to a motor issue and the worker may have to stand in the path of robot movement and thus if he fiddles with the motor and it may start running then worker can be hit with robotic arm and cause injury [40].

The accidents caused by robots can be classified based on the type of dangerous situations a human may end up in with the robot. They are described in the following cases.

A human worker maybe crushed with arm motion of the robot or a human may get trapped in a situation where the robot may move to a point and this is where there is no outlet for human to escape. This case may arise when a human gets trapped in between a wall and the robot arm for instance. Other times a robot may directly hit a human thus causing collision. There maybe other random safety situations arising when a robot and a human are present together in a common space like heat burns or electric shocks.

The above listed hazards are minimized by: [20]

- enforcing strictly pre-mapped environment and space for the cell of cobot
- strictly followed operational routine

- authorization of machine operators, maintenance workers and programmers
- speed limitation on movement of control surfaces in presence of human
- emergency stop function.

4 STATE OF THE ART

This chapter introduces and describes the technology and tools used for modeling and programming in this thesis work and this forms the state of the art for this thesis. In this section, firstly a basic introduction to the technology of task is given and this is related to what is being tried to be accomplished in real world and then following section describing first, the state of the art for theoretical models designed and then technology for the programming and related tools are described in the second section.

4.1 Motion Planning And Simulations

This robotics project revolves around the idea of motion planning. It is about design and implementation of a model based architecture for cobotic cells. The basic idea is that there is a robot which has to cohabit with humans and it is an effort to make it real world intelligent and this means it has to work in real world where obstacles, objects and humans are present. Accidents are imminent and hence the case study is about building intelligence in robot to deal with obstacles, thus enabling robot to provide safety for cohabiting humans.

Using motion planning, tasks are accomplished by the robot and safety provided and this is starting step of building theoretical models for motion planning known as application model. Similarly world model is designed as a complete visual model for world of robot as well as safety model is designed as a theoretical model for safety of robotic system. All these are also programmed and described more objectively in later chapters. But for now only tools and technologies used to design and program them are described in the following two sections respectively. Firstly, models and their need is introduced and then techniques used for visual depiction of models is described. Here a few of the optional depictions are talked and described and then the one selected for this thesis is described for each of the models. Secondly, technology used for running the program made, i.e. to implement these models is described [16].

4.2 Modeling - Explaining Choice of Design Depictions

This thesis uses the concept of designing architectural framework for human and robot cohabitation. For this model based cobotic cell design is used. Thesis work tries to find correct diagram depictions for all three intended diagrams which are world

model, application model and lastly safety model. For this the concept of model need to be explained and also then why we need models and the various types of models along with the chosen models for all the three concepts of application model, world model and safety model.

A model can be described as an abstraction of real world or a system, focusing on some specific structural and behavioral properties, which are then expressed in a syntactical and semantically defined language representation [8]. There are various types of models which use different levels of abstraction. Using right level of abstraction is critical and it usually defines the granularity of the model and the complexity it expresses. There are usually always some details that are abstracted and some highlighted. There are various techniques of model construction types and some of them are object oriented models, process based models and hybrid models. These models thus help to visualize the structure of a system and hence make them an essential tool. A brief description about the types of models follows.

Object oriented models have developed as a modeling paradigm, after being acting as programming paradigm. Most basic example is UML diagrams. The second type is process based models, they are more formal diagrams which are used to express domain specific and custom properties. Example is petri-nets and bigraphs. The third type of model is hybrid models. Some properties of safety cannot be sufficiently described as discrete models but as continuous dynamic models. Thus this model combines dynamic, discrete and continuous modeling languages. Example is hybrid-petri nets. Various design depictions were considered and after contemplation following depictions were selected.

4.2.1 Unified Modeling Language (UML) Diagram For World Model Class Diagram

This thesis work includes extensive research on the UML Class Diagram and its features and how they can be used. Then a few of tools used to construct UML Class diagrams are described.

Unified Model is a language used to appropriately formalize the static structure of a project using classes as building blocks. This modeling is most basic depiction of object oriented modeling and data modeling. UML class diagram describes classes, attributes they have and functions used to implement them, and show how the objects of these classes are related. This model cohesively describes the data contained in a project and how it all is related together to produce desired functionality [13] [19].

The class in this diagram is the basic unit. The class is an almost exact depiction of real object oriented class and contains three sections, first being the class name, second is the section for attributes i.e. data variables and their type held by the class and third section describe the operation, thus holding the methods. The UML class diagram can be extended using state machines which is described a few sections later to describe another model known as safety model and i.e. it is based on extension of UML class diagram. Here is depiction of a typical class in a class diagram in Fig.4.1.

There are various types of relation that connect classes together and their usage depends on complexity of the project. Each type of relation is described by a unique

Figure 1: The Class Icon

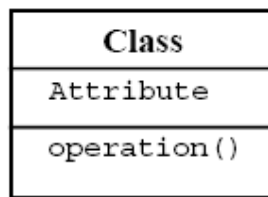


Figure 4.1 A typical class structure in UML class diagram Image

Source:<https://sites.google.com/site/revasolution/techhome/uml/clsdiag>

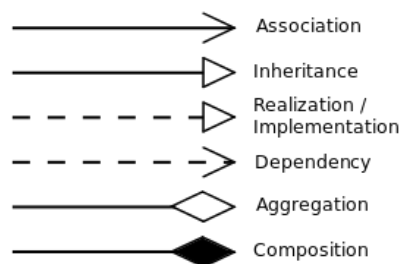


Figure 4.2 UML Relations Image Source :

https://en.wikipedia.org/wiki/File:Uml_classes_en.svg

symbol of connection. Few of the relations are dependency, association, aggregation, inheritance and composition as shown in the Fig 4.2 and described in sections later.

(A) Dependency

The dependency relation is marked by a dotted arrow. This implies a one-way relationship between two variable. It means that one variable is dependent on the other. If value of one variable in server changes then the value of variable dependent on it also changes on client.

(B) Association

An Association means a family of links. This is marked by a bold line arrow. It means a static relationship shared between objects of two classes and this relates the classes belonging to a family. Any number of classes can be related by association relation.

(C) Inheritance

Inheritance is another relationship. This is a property where in all characteristics i.e. variables and functions, of a class are adopted by another class as well and they are known to be related by inheritance. The class adopting the properties is called as the child class and the class from which it adopts them is known as parent class. This is very commonly used relationship in class diagrams. This is marked by a bold line arrow with hollow tip.

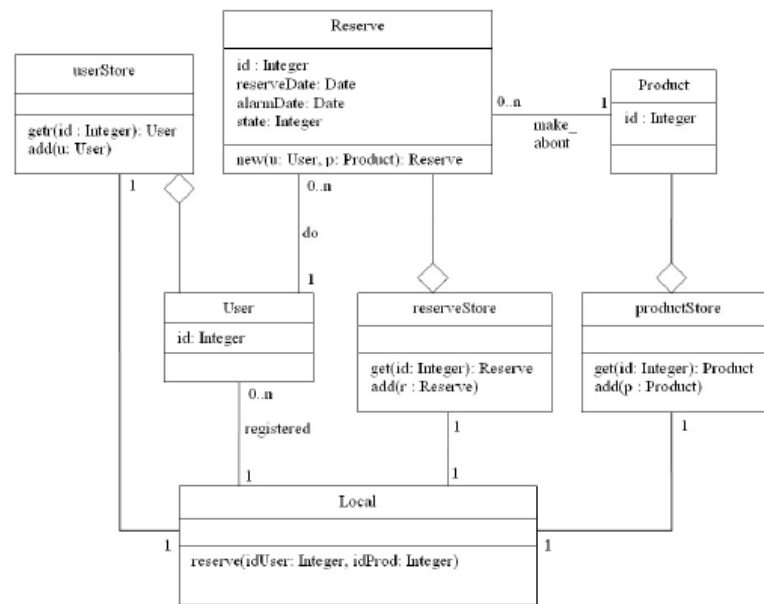


Figure 4.3 Example of UML Class Diagram Image Source :

https://www.researchgate.net/figure/Example-of-a-UML-Class-Diagram_fig1_22120

(D) Aggregation

Aggregation is one of more refined version of association relationship. Here functionality of one class is dependent on another one, i.e. if one variable do not exist then another variable having aggregation relationship is also not able to exist or be used. It is depicted by a line and tip having a quadrilateral.

Here is an example of a typical UML class diagram displayed in Fig. 4.3.

Tools used for UML class diagrams is Online Visual Paradigm. This is an online editor to construct UML class diagrams.

4.2.2 Business Process Modeling Notation (BPMN) for Application Model

Business Process Modeling Notation is used for Application Model [38]. This is a design depiction used to construct application workflows and processes. BPMN provides a standardized bridge for the gap between the business process design and implementation. A Business Process Model is a network of graphical objects, which are activities(i.e. work) and the flow controls that define their order of performance and the flow. This diagram is made up of elements and there are many category of elements but the thesis work uses two major group of elements known as flow objects and connecting objects. Here in Fig 4.4 is displayed an example of BPMN.

The flow objects contain Events, Activities and Gateways and the connecting objects have Sequence Flows, Message Flows and Association. They all come together to give desired form to the process workflows. Here in Fig.4.5 is image of all components of BPMN.

A short description of all components is as follows.

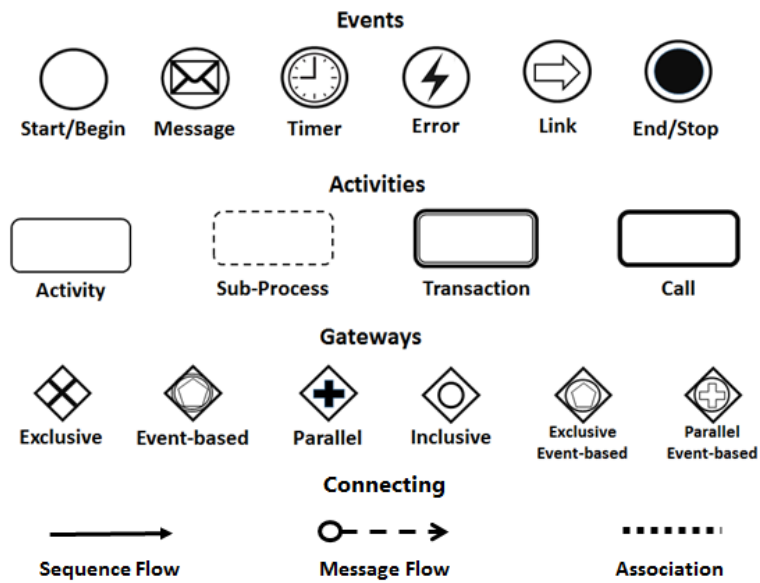


Figure 4.4 Components of BPMN Image Source :
<https://study.com/academy/lesson/business-process-model-and-notation-process-examples.html>

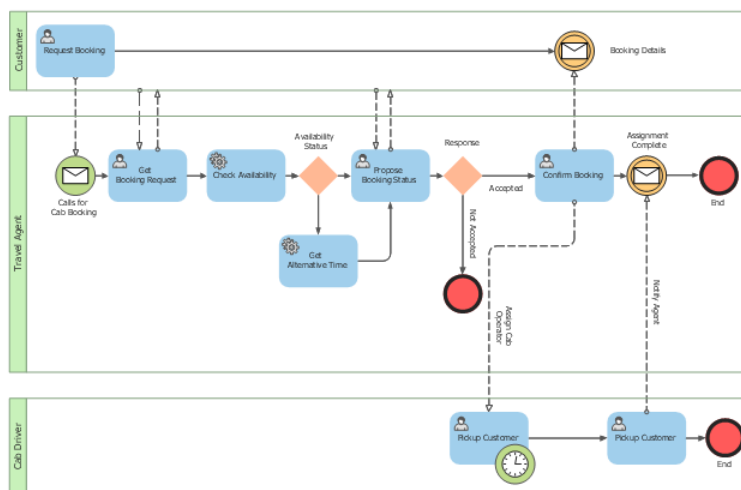


Figure 4.5 BPMN Example Image Source :
<https://www.conceptdraw.com/examples/taxi-booking-process>

(A) Events

An event is depicted as a circle. This is something that marks something happening during the course of workflow. They affect flow of the process and have a trigger point or an result state. There are three types of events, and each of them show stage of the event happening and also effect the flow. These are start, intermediate and end events as is marked in the Fig 4.5.

(B) Activity

Activity is depicted by a rounded-corner rectangle. It marks work that is performed. There are many types of activities but the main are Sub-processes and tasks as shown in Fig 4.5.

(C) Gateways

A Gateway is shown by diamond shape and is used to direct the divergence and convergence of the work flow sequence. It governs traditional decisions, as well as the decision when to split, merge, and join paths.

Here is description of Connecting objects also known as connectors. They are connected together to create structure and demonstrate flow. Three connectors are described below.

(D) Sequence Flow

A Sequence Flow is shown by a solid line with a solid arrowhead and this shows the order of the activities that will be performed in the workflow process.

(E) Message Flow

A Message Flow is depicted by a dashed line with an open arrowhead and it is used to mark message transfer between two different processes. Depicted in the image.

(F) Association

An Association is used as a dotted line with a line arrowhead and it is used to associate data, text, or miscellaneous artifacts with flow objects. They are used to show the inputs and outputs of activities.

All these components come together to let users create a precise, detailed and desired form of expression using BPMN.

4.2.3 Unified Modeling Language State Machine for Safety Model

UML state machines are finite state machines expressed in the form of unified modeling language notation. The concept associated is about organizing the way a process works, such that an entity or each of its sub-entities is always in exactly one of a number of possible states and there is well-defined conditional transitions between these states.

Today almost all software system are event driven. These systems can be external or internal events like a mouse click event. After the event is handled, the system goes back to waiting for another event. This describes the concept of finite state machines. The system can be in only one state at any given time instant. The system when changing state is known as state transition. When finite state machines are expressed in the form of UML diagrams then they are known as UML state machines. They use same components as are used in UML class diagrams.

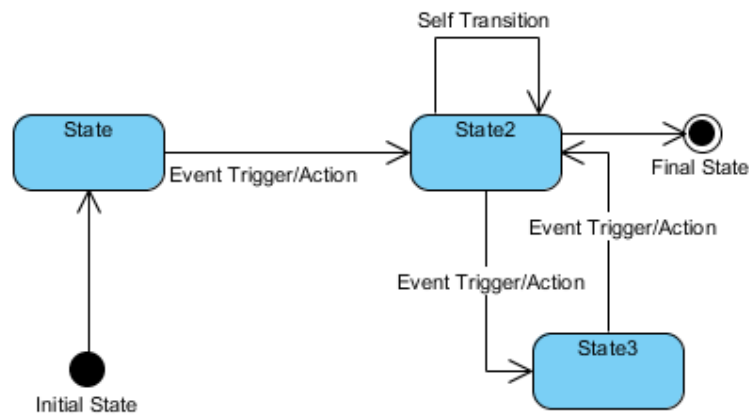


Figure 4.6 UML State Machine Image Source : <https://www.visual-paradigm.com/tutorials/how-to-draw-state-machine-diagram-in-uml/>

4.3 Tools Used

4.3.1 Setup Environment Using ROS To Run Services, Motion Planning In MOVEIT and GAZEBO For Simulations and more

(A) ROS ¹

ROS is an opensource robot operating system. ROS is not a regular OS, in the sense it does not provide regular functions of OS like process management and scheduling but it provides a different set of services and acts like structured communications layer above the host operating systems. ROS is associated with existing frameworks of robots, with brief look on available application software which uses ROS. As robotics is a wide field and continuously a topic of research and a growing one, generating code for ROS is not easy. There are different category of robots available with high degree of variation in hardware, thus not enabling programmers to reuse code or develop on modules. In addition the total amount of code needed is too much for regular programmers, as it needs a deep stack starting from driver-level software and continuing up, and also needs abstract reasoning, and more. The required breadth and width of expertise needed is far more than the skills of any single researcher and robotics software architectures must also be able to be integrated with large-scale software. To address to these problems and make life easier for a regular programmer, many robotics researchers, have constructed huge number of frameworks to handle complexity and address to rapid prototyping of software for experiments, thus resulting enabling research in industry and academia. Each of the frameworks were made keeping in mind a purpose, maybe for a response to perceived weakness of other available frameworks, or to place importance on dimensions which were seen as most important in the design process. ROS, the framework is designed not without trade-offs and prioritizations made during its design cycle which were essential to do in interest of practical uses. It is still thought the trade-offs will serve well to purposes of large-scale integrative robotics research in a wide variety of uses and cases as robotic systems grow ever more complex [30].

¹<https://moveit.ros.org>

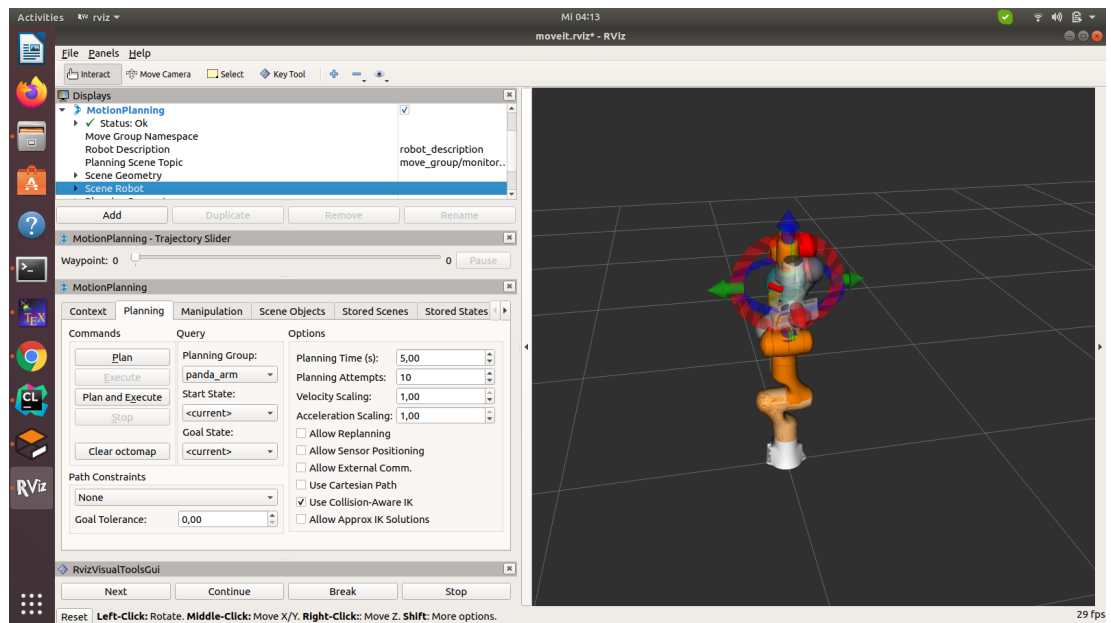


Figure 4.7 MoveIT / RVIZ Screenshot

(B) MoveIT ²

MoveIT motion planner was used for motion planning as this software lets users alter many different parameters of the robotic components and helps to create case studies in a world environment for robot Panda. Here a series of joints and poses is set and then a trajectory for motion planning, which is then used to run in a real world simulator. Its main purpose is to motion plan and introduce an obstacle which is a cube box or which can be a human being and the motion plan is about moving the robot arm around the obstacle to reach a position which was decided earlier in the motion plan [25]. There can be as well other motion plans that are simpler and more complex

This is motion planner used to plan the motion of robot Franka Panda in this case. This connects to simulator for running real world simulations using the Robot Operating System (ROS) and launch file. Here is screenshot for Rviz.

When MoveIT tool is started for the first time there comes up an empty world. On the top there is option Panel which can be clicked to see a drop down menu and from there some of the panels can be selected which need to be used. The motion planning enables user to set various field parameters and their values, like some of them are Fixed Frame, Robot Description, Planning Scene Topic, Planning Group and Planning Request and more like Planning Trajectory and they can be explored further using online tutorials provided by ROS, thus adding to a lot of functionality and flexibility³.

Fig.4.7 This is what Rviz motion planner looks like. The robot is stationary in this case and can be moved using arrows displayed in different colors to attempt motion in different directions. The 7 joint of robot Panda give it immense flexibility but there maybe some motions which are not possible and in this case it is made

²<https://moveit.ros.org/>

³https://ros-planning.github.io/moveit_tutorials/doc/quickstart_in_rviz/quickstart_in_rviz_tutorial.html

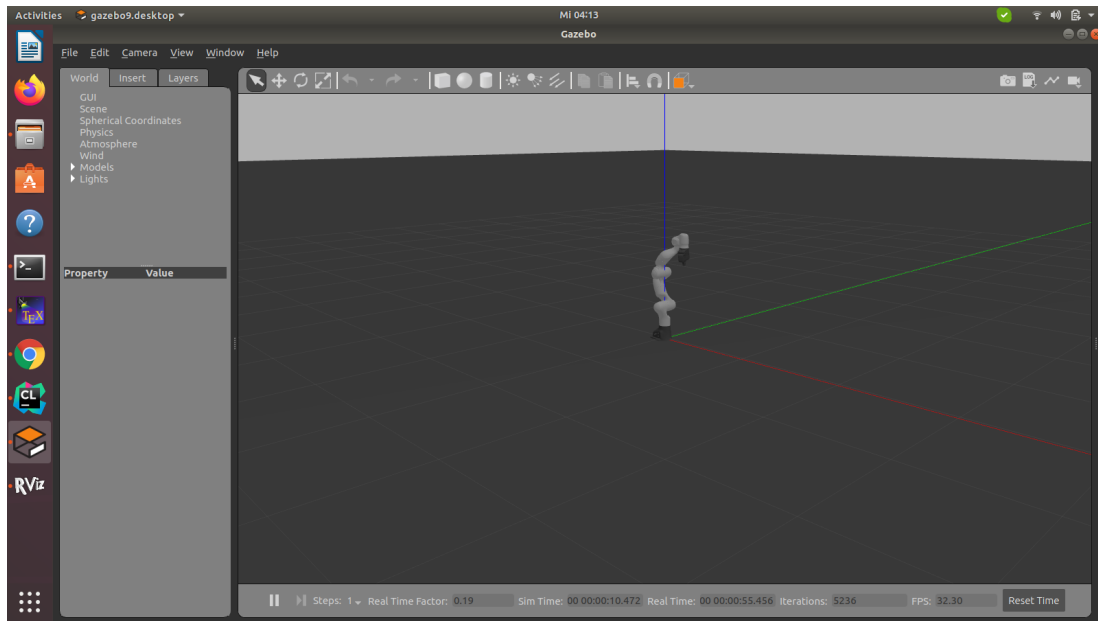


Figure 4.8 Gazebo Screenshot

obvious while using MoveIT. The Displays on the top left shown are showing various different parameters which Rviz can help tune and change. Some of them are Robot Description and Planning Scene Topic and there is Plan and Execute button which can be used to run the plan once the arm is moved to a new position termed as final position. The plan can be executed in Gazebo simulator by clicking Plan and Execute button and the trajectory is visible in the Rviz and then almost immediately the Gazebo simulator executes the motion.

(C) Gazebo ⁴

Gazebo simulator was used which is a simulator to run the motion plan from MoveIT. This lets users see if the real life simulation is possible for the conceived motion plan and trajectory visualized in a visualization tool like MoveIT. Gazebo also has additional features which can add and alter real time parameters to its simulation like altering torque of joints to see how robot reacts in varying real world conditions [32].

This is a simulator which simulates robot motion with real world parameters tuned to check if the motion plan is executable and feasible in real world. Here is how the Gazebo simulator looks on screen.

In this thesis work there was no need to explore functionality or fiddle any of features in Gazebo. It was used to visualize motion plan of MoveIT and to check if this is real world feasible in Gazebo. In this tool it is possible to tune a lot of real world parameters and the ones related to robot Panda are torques on joints but this was not required.

(D) Clion IDE ⁵

⁴<http://gazebosim.org/>

⁵<https://www.jetbrains.com/clion/>

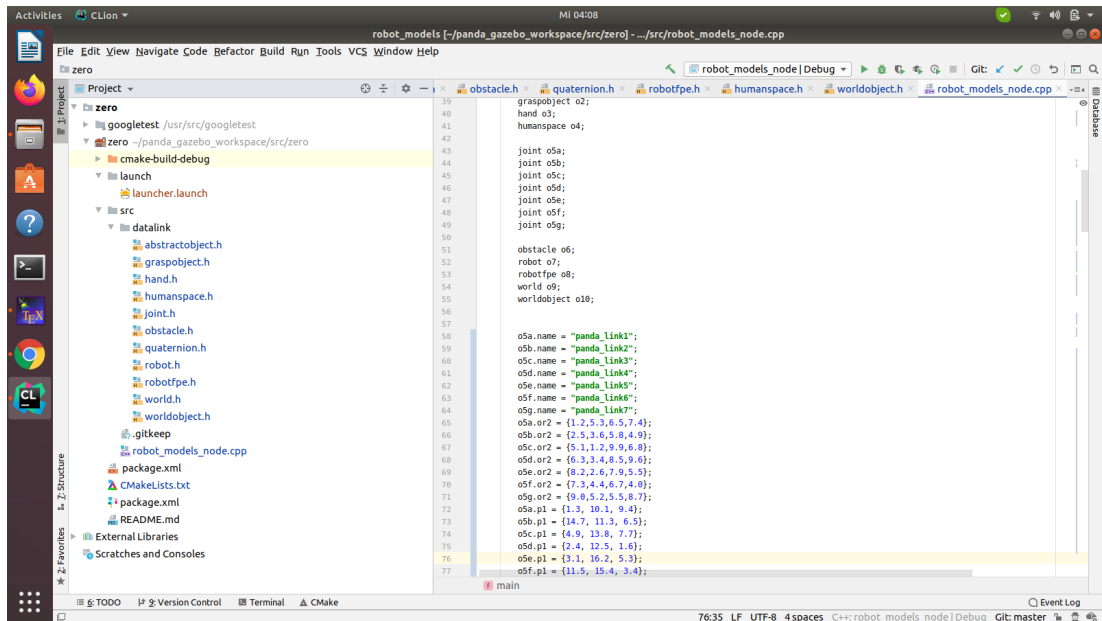


Figure 4.9 CLion Screenshot

This is the editor used to write C++ code for the tasks in this project and this is IDE used to compile and run it. CLion is used to provide many aides to help reduce the programmers workload and automate the process of adding code by providing many suggestions using inbuilt libraries and showing connections among variables. In addition it points out logic errors in advance and helps mitigate errors and warnings that may come at a later stage by improving the quality of code while still in construction process. CLion IDE can be started by opening a terminal in linux and typing command "clion". The Fig 4.9 shows how CLion IDE looks like.

CLion window shows the complete project structure on one side and each opened program on right side under tabs, making it easier to understand the code and make things easy to lookup. The window in second half of screen is the console output window. It shows the errors and command line output. These are some of useful general features of Clion IDE in addition to many more used.

(E) Gitlab ⁶

To keep the constructed code safe from loss and to keep it accessible and visible as a package to administrators and a host of users, the code is uploaded to a central repository for which Gitlab was used. This is a widely used repository for students in a university setting.

It shows the projects available under the namespace of account owner and has a range of features and commands used, which are described in the following paragraphs. The Fig. 4.10 shows how a typical Gitlab account appears in web.

After logging in to Gitlab account with credentials of username and password, the users are welcomed to a page which shows repositories available. The groups and repositories subscribed by the owner user or given access to owner user by other users are demonstrated here at the page. The top section has tabs "All"

⁶<https://gitlab.com/explore>

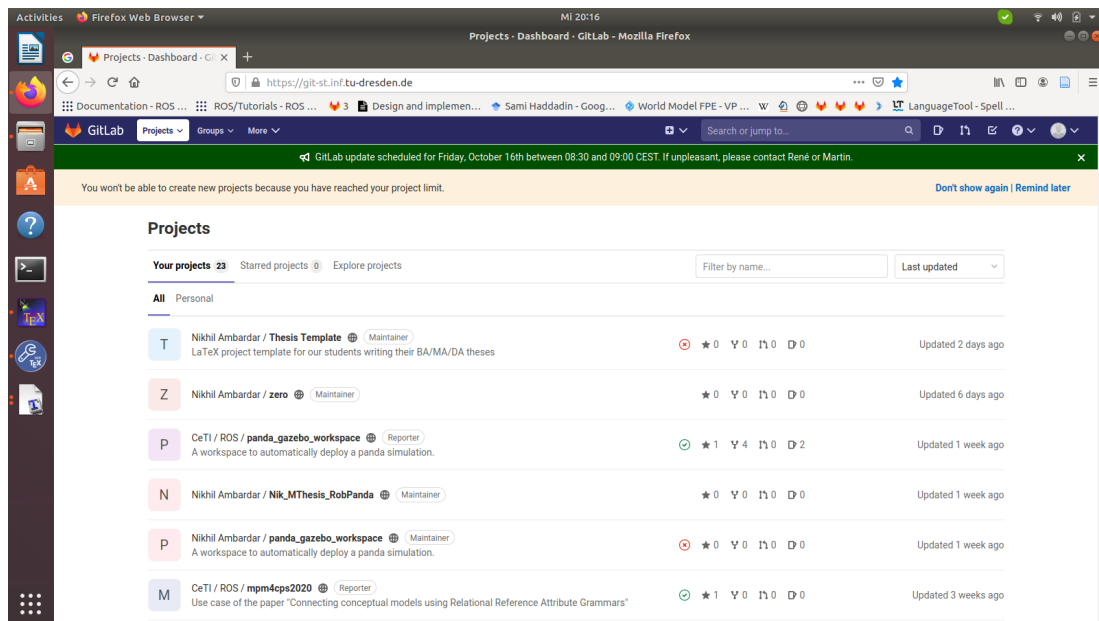


Figure 4.10 Gitlab Screenshot

and "Personal". "All" signify all projects which owner user has access to and the "Personal" tab has repositories forked by the owner user only. The owner user can give access to other users for their repositories too.

Gitlab is a tool that allows user to create a fork from a parent repository and be able to use it as a personal repository. In cases when the personal fork has become corrupt or cannot be used anymore, it can be re-forked from parent repository. Gitlab also allows construction of more than one branch of the forked repository to solve this problem, so in cases when the master branch of repository is not usable anymore, a new branch can be created and updated with. This is also useful to maintain different versions of the code and this is one of many features of Gitlab this thesis used. Forking can be done by clicking on the parent repository and then clicking on Blue tab marked as "Clone". Then appears two options "Clone with HTTPS" and "Clone with SSH", one of link is which needs to be copied. The command "git clone" can be then used on terminal, appended with the copied link from Clone tab to fork the repository to local machine.

After forking from a parent repository, work is done and added to the folders forked from repository and used on local machine. But they need to be uploaded to Gitlab at regular intervals to log the progress and keep it safe from loss on local machine. For this, the folder is to be uploaded or pushed from local machine to the Gitlab account for which commands have to be used from terminal of local machine. A short description of commands used to achieve this are stated below.

Many commands used are general git commands which mostly include "git status", "git add .", "git commit", "git fetch", "git pull" and "git push".

First of all to use git upload in the project, user must go to the project's parent directory in linux and right click to click on "Open In Terminal" option to open a terminal which is already in parent directory of project workspace. Or else, a terminal can be opened and "cd" command can be used, suffixed by the path to parent directory of workspace after a space. This "cd" command can also be run in steps and this changes current directory to stated, suffixed directory at

each step. Using this in steps, users can switch to the directory that need to be pushed in project.

Second step after switching to the directory to be pushed, is to see its status on git and this can be done using command "git status". This command shows what are the changes since last commit or what is changed and new.

Then command must be run to add all changes to git and this is done using command "git add .". This adds all changes and new files to git to be used as desired. Then ideally once again status should be checked to see if the files added are now shown in green color after adding them to git. This can be done using "git status" command once again. All files must now appear in green and this shows they were added to git since last add command. This step alternatively can be skipped.

Then changes added must be committed and this can be done using command "git commit -m "xx"". The xx is a commit message and this text is used as a markup. Any text can be used here instead of xx. This command commits changes to git. Then "git fetch" command can be used to see which is the branch fetched. This is to confirm if the commit is on correct branch.

After this command, command "git pull" can be used to see if there are still changes between local git version and what is in online repositories version. After this once again status must be checked. If there is nothing to add, then all is well but we can once again use git "add ." command to see if anything more can be added. At this point "meld" tool can be used to match the changes in file i.e. if there are any, between online git and commit but this may not be needed for regular uses and only in cases when there are any discrepancy differences due to irregularities between pushes. Then "git push" must be used to push all changes to Gitlab. This prompts for user to enter username and password and after this stage all changes are pushed to Gitlab account folder. Thus this shows a completed git command line push process.

(F) Erdal's Repositories ⁷

There are few startup repositories provided by Erdal to use by including as package in workspace and they are essential to run robot Franka Emika Panda using ROS on local machine. The robotic framework package has libfranka and this is a C++ program library, frankaros which is a ROS interface with ROS Control and MoveIt integration. The links above contain all repositories that can be used for robot Panda including the ones that has been used by work of this theses.

⁷<https://github.com/frankaemika/> , <https://erdalpekel.de/?p=55>

5 CONCEPT

5.1 Theme

This section defines all the concepts associated, discovered and learned with the work of this thesis. The main idea begins as described as a real world with a robot present. The real world consist of robot Franka Emika Panda present. The robot has an arm having seven joints. There can be one or more humans present in proximity, along with one or more obstacles which are also present in the proximity in real world. The base of robot is fixed and the arm is capable of motion and there is a gripper at the tip of arm, which need to pickup and then release an object to complete a job. The robot has to be programmed to move and not just move the arm but do this safely i.e. by detecting obstacles / humans around the robot continuously and responding appropriately, by evading the obstacles to reach final position and thus complete the pickup and release job. There can be other jobs like piercing a balloon ball as depicted in the Fig 5.1.

This section starts with a hypothetical situation to explain the need for models and the concepts associated with implementation. The robot uses a motion planner software to follow a preconceived plan which is visualized and that is to move arm and pickup an object and then move arm again according to already planned motion trajectory and release the object at a desired location thus completing the task. Image describes this as shown in Fig 5.1. So far this plan is only about doing the task, but sans the idea of any kind of obstacle or human which can cause a hindrance to already planned motion.

The activity until this point can be used to train the robot and this motion trajectory job can be re-executed by the robot using the inbuilt teaching capability feature of robot which lets the robot learn a series of pose and actions and replicate the same trajectory and grasp actions, but then a real world feature is added to robot and this is about adding essential safety feature. This safety is achieved by altering the planned trajectory right at that time instant when sensors detect obstacle in the path of robot. At this time, robot uses a new trajectory to move its arm around the obstacle, to reach a coordinate position around the obstacle to a point in pre-decided and followed motion trajectory and then continue regular motion from there on-wards to complete the motion and task. After the motion planning part a simulation software is used to see if motion trajectory correction is feasible and working in real life or not and to see how successful it can be [7] (Chapter1,Chapter4).



Figure 5.1 Robot Panda Picking Up Object Image Source :
<https://blog.generationrobots.com/en/list-of-criteria-to-look-at-before-buying-a-robot-arm/panda-franka-emika-care-robot-arm-2/>

This is a model where human / object - robot interaction operation is depicted, as shown in Fig 5.2.

5.2 The Models Designed

Tasks in thesis work are to design three models namely world model, application model and safety model using modeling tools and implement them in a C++ program. This sections begins with describing each of these three models.

(A) WORLD MODEL USING UML

The world model describes the world of robot Panda in general i.e. about what is inside the surroundings of the robot apart from the robot itself and then features and attributes of all components in this world model. In technical terms this is the world of the robot that exists in real world and components in this real world are the robot FEP itself with its arm, the obstacle like a human or an object and a cube or ball which can act as an object, that can be picked up by the robot arm. Fig. 5.3 and 5.4 shows the designed World Model class and object diagram respectively. These were designed using UML diagrams.

Unified Modeling Language (UML) class and object diagrams were constructed for world model using the web tool named Online Visual Paradigm. The UML class model is used to derive the objects and depict in the UML object model diagram [2].

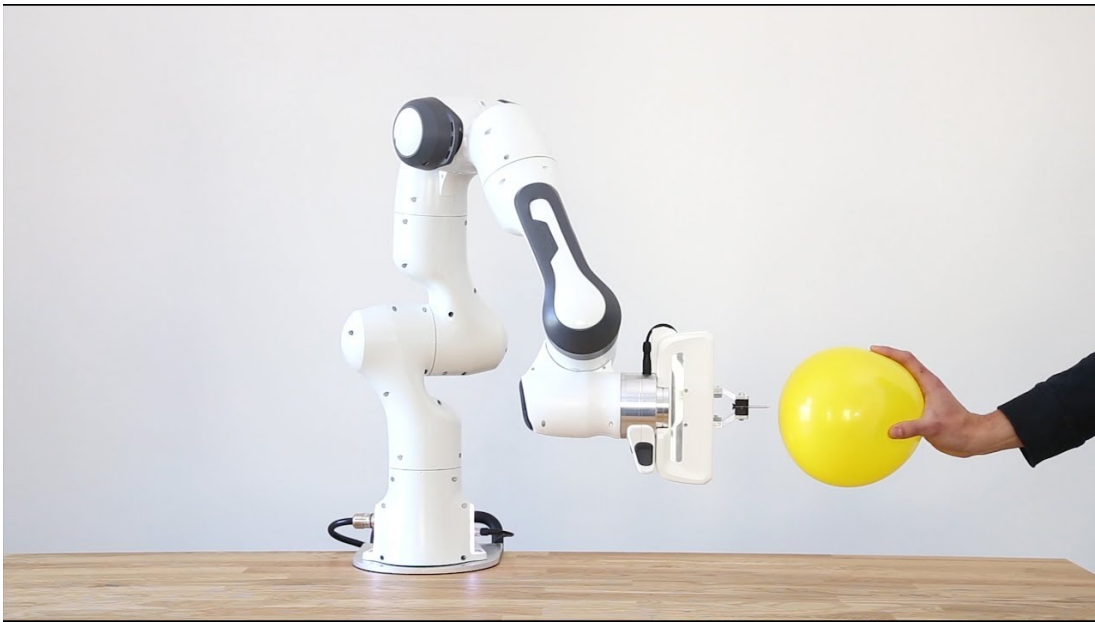


Figure 5.2 Another Image of Robot Franka Emika Panda Image Source : Panda Skills Sensitivity Video Screenshot

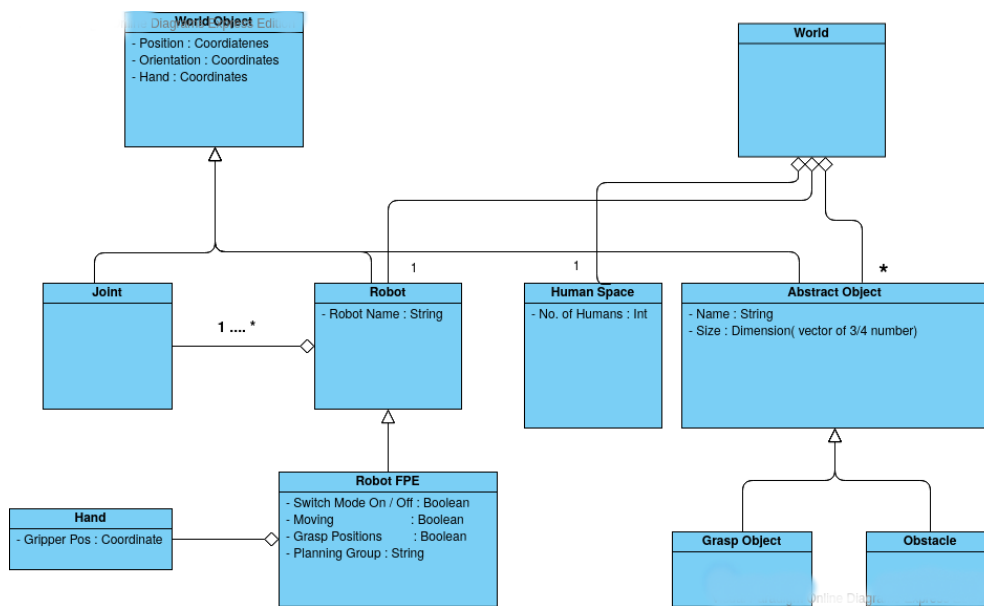


Figure 5.3 World Model UML Class Diagram

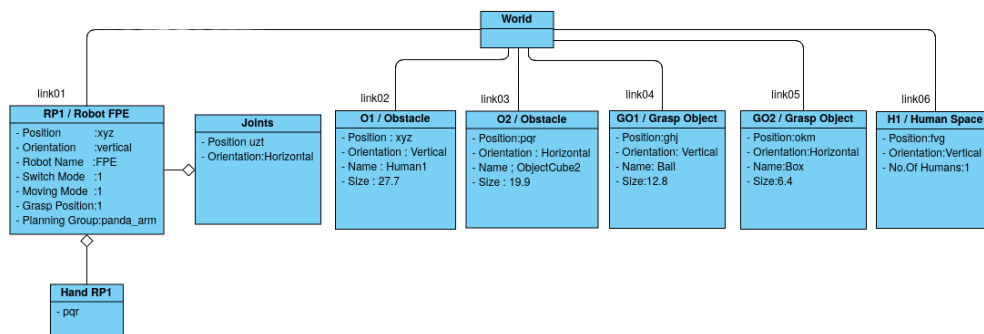


Figure 5.4 World Model UML Object Diagram

This world model class diagram has been designed to contain ten classes. The main parent classes are World Object and World, present on top most level. The classes Robot and Abstract Object inherit from one class and at same time are aggregated with another class. Relation of association and aggregation, is used where classes need to use data variable from other classes and in cases if class variable must depend on other class variables for it to exist, respectively.

Specifically this means, aggregation relation is used in a case where if the class is not involved or used, its child classes and their variables cannot be instantiated or used. Association relation is used where one class is related to another just to be able to use the other's variables. Both relations use pointers in program implementation in C++.

(B) APPLICATION MODEL – BUSINESS PROCESS MODELING NOTATION USING MODELIO

Business Process Modeling Notation(BPMN) for application model using Modelio tool¹, was chosen for application model. The designed application model is depicted in Figure 5.5 [38].

BPM notation was deemed to be a correct choice to show application process workflows as this shows segregated cells, called frames for all the entities present and this depicts each component of the world model diagram. The notation then allows to depict the relationship and connections between the logical components of the frames and shows their flow which have a comprehensive and logical consistency among cells. This is achieved using features of BPMN like activity, gateways and events which form as logical components. It uses start and end event states to mark positions for start and end and then "if" conditions are used as flow lines with process events and intermediate events to construct application model.

(C) SAFETY MODEL DEPICTION USING UML STATE MACHINE

UML State Machines are used to construct safety model. They are an extended version of UML finite state machine. The Fig 5.6 shows the safety model constructed using the tool Modelio.

The start and end events denote, the process starting and end states. The state machine can, at any point in time be only present in one state. The robot motion begins at point of start event. The transition T1 is about human presence and activity of human presence is marked, assuming there is already a human present detected in the proximity. The robot motion had started but now MoveIT is requested to provide a new motion plan around the human obstacle, this is marked by transition T3 and once a new trajectory is obtained the motion starts continuation around the human and once again robot reaches a point in original motion plan and then it keeps checking for human presence and this is marked using if condition "Detect Human", and if detected by sensors during the continuing motion plan. If the human is detected then once again new MoveIT trajectory is requested around the human and the loops keeps running

¹<https://www.modelio.org/>

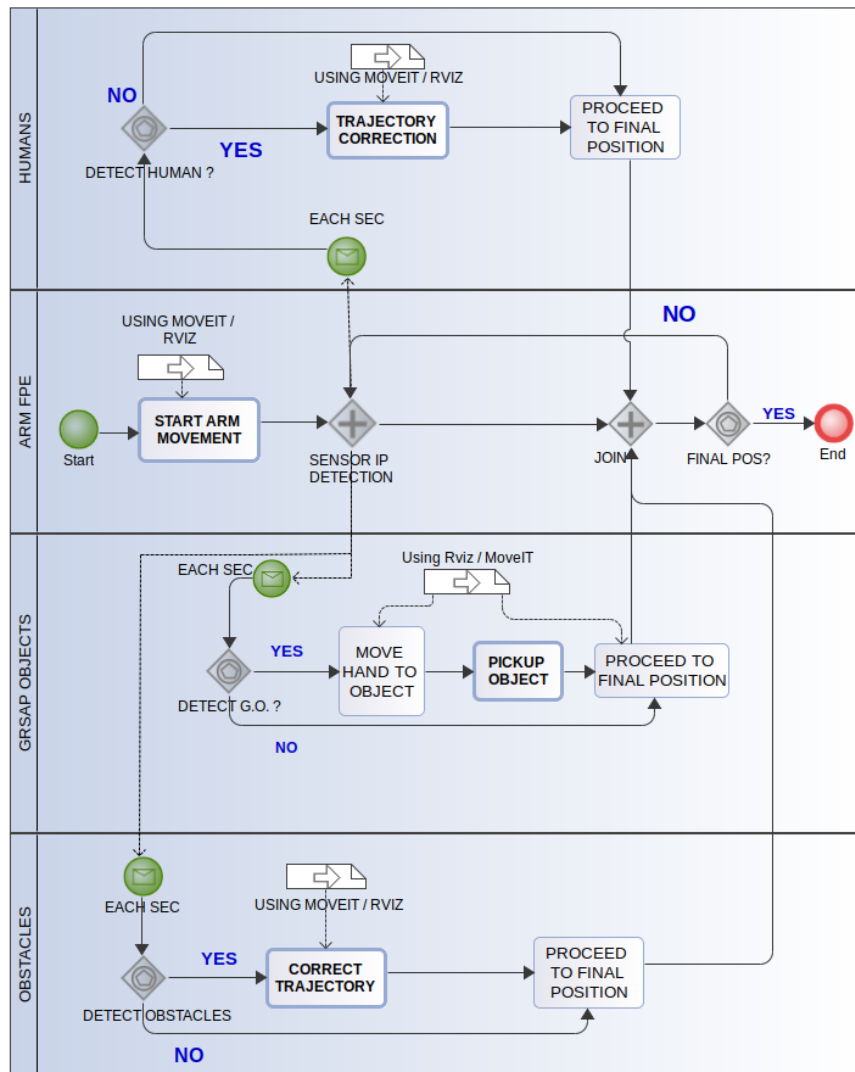


Figure 5.5 BPMN Application Model Diagram

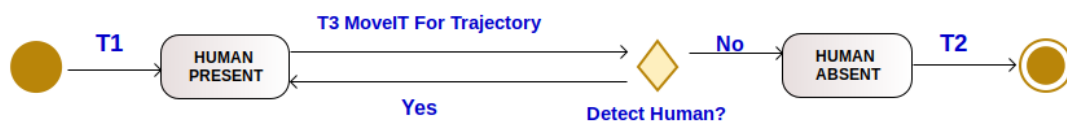


Figure 5.6 Safety Model - UML State Diagram

to check for human presence until a state is reached where human is absent and arm moved to a position around the obstacle at a point where it can begin original motion plan. Once at this point, human absent state is achieved the original motion is proceeded to be completed and marked by final transition T2 proceeding to the end event finally.

This model can be extended in more detail in the future using this concept along with a different notation that can make the cases more detailed and thus more extensive. MAPE-K loops can also be used to denote the safety of a system and in addition computer generated graphics can as well be used to depict the safety model of a system, this enables the research to not get restricted to just BPMN notation for safety models of a system [38].

There has been extensive work done to incorporate safety into real world robots particularly from Sami Haddadin. The work has built the robot with improved technology and making the robot understanding safety i.e by making them softer in operation when operating, to prevent any physical collisions by embedding injury knowledge into controls like emergency stop. These efforts also result in using reduced force when in human proximity and other efforts are to make robot surfaces softer and for this extensive testing has been done on injuring pig skin to study effects of injuries that can be caused by robots [37].

5.3 Programming Of Hardware And Software Components - Purpose And Concepts

In hardware, there exists the Franka robot which has an arm and the arm has joints described by J1...J7 . We also then have other objects in the world namely obstacles, which can be one or more human and then non living ones like a cube ,box or ball. In addition there are grasp objects which can be a cube or an item to pick.

The robot Panda has motion planning attributes which can be altered. Most of them are in MoveIT which is the motion planner tool described in detail in the last chapter. The state of robot arm is described by the coordinate position of last joint of the arm i.e. J7. We can as well alter the many other parameters like torque on the arm in simulations. The Figure 5.7 shows the robot panda with joints .

In the Software section ROS [30] ² is used which is the Robot Operating System and catkin builds the workspace. ROS is started using "roscore" command in terminal and this starts the ROS on local machine. MoveIT is then used to do motion planning for the robot and planning motion around the obstacles. The plan in MoveIT is then run in a simulator for which Gazebo Simulator was used, which is used to replicate real life conditions and run robot inside it. This gives an idea if the robot can perform as planned and expected in real life with torque on joints or other real world like conditions. To use MoveIT, workspace is built using catkin. Erdal's repos are used and essential in the workspace which are franka_ros , panda_moveit_config and panda_simulation, and they are imperative to build the workspace which contains the node program designed. These are the repositories used to build the package and thus run the

²<https://www.ros.org/>

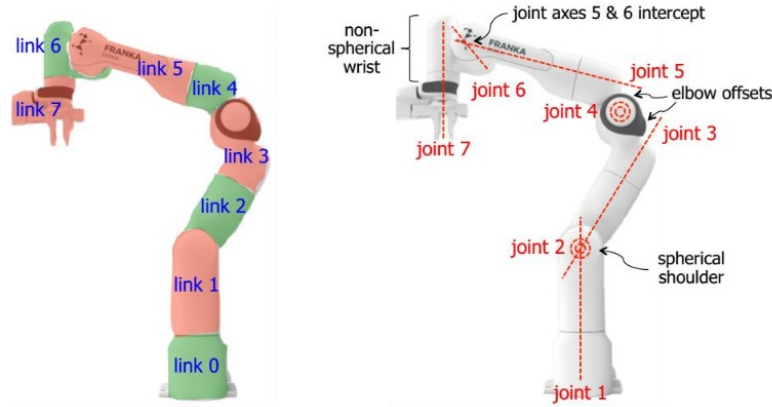


Figure 5.7 Robot Panda Joints Image Source : <https://www.chegg.com/homework-help/questions-and-answers/panda-franka-emika-shown-below-is-innovative-lightweight-robot-intended-friendly-and-safe-hu-q35002486>

program using ROS. The basic idea of robot Panda package which is made up of all the above mentioned tools and software is to execute a real time bidirectional connection between a workstation PC and the arm of the robot shown in simulator [32].

5.4 Refactoring and Reclassification

The work of this thesis needs the implementation of the concept of refactoring and reclassification of code created. The code created for the project in the implementation stage has gone through many phases of construction. These phases are classified as stages, in these stages code has developed in quality and complexity. The plan with which code construction had started, was changed and many amendments made for this, in addition to many extensions in the plan. This made the code very difficult to understand at some places. Corrective actions were taken which include refactoring and reclassification of project code. This is a concept of restructuring existing code while keeping the behavior and functionality of the code as it is.

Refactoring is a concept to improve the design and structure of the software. Few advantages of refactoring are improved code readability and reduced complexity. This concept also then helps with more extensions which can be part at a later stage and to maintain and demonstrate a simpler and cleaner and more expressive internal architecture of project code. Refactoring can also improve performance by saving memory and making the program run faster [3] [42].

5.5 Connecting Everything - Simulations

Gazebo Simulator is run alongside MoveIT motion planner which help replicate the MoveIT motion plan in Gazebo simulator and thus let user manipulate the robot using motion plan in Rviz. How is this achieved is as follows.

In the project workspace file structure described in the next chapter, there exists a launch file. The launch file parameters instructs the tools to be started like MoveIT and

Gazebo in this case. It is also specified with a program node filename to be launched. The parameters include name of the node file, package name and few other parameters. The launch file is then run from terminal. This is accomplished in following steps and after changing directory to workspace folder, command "source devel/setup.bash" is run after which command "roslaunch packagename launchfilename.launcher" is run, which launches the node file allowing it to communicate and use values from other tools like Gazebo. All this runs using ROS and using message exchanges. The node can then be programmed to listen to frames from simulator and also command the robot in simulator accordingly or as intended. ROS messages are sent from program node to simulator and back. This thus gives a brief overview of how everything works while a detailed explanation follows in the following chapter.

6 IMPLEMENTATION

6.1 Introduction

This chapter describes about what has been done in this thesis to achieve the objectives. This implies all that is practically done and this includes programming the node using ROS packages eventually and using Latex tool. Also described here is the structure of both components of the project in steps.

The work involves creating a workspace directory in file system which in this case is known as `panda_gazebo_workspace` and installing all necessary software like ROS. A folder named `"src"` is created within this workspace `panda_gazebo_workspace`, which is then added with other essential repositories cloned as packages, the likes of which are known as `franka_description`, `sample_applications` and `panda_simulation`. These are packages which are essential to building the workspace i.e. running the project and they are provided by Erdal and can be cloned / forked.

Parallel to these packages, a new sub workspace is created, known as `"zero"` in this case, which contain variable part of workspace i.e. all the work that is added specific to constructing and running node programs in implementation step. The `"zero"` folder constructed here contains a launch folder which contain the launch files for launching the project from terminal and in parallel it contains another folder named `"src"` (not to be confused with `"src"` folder created before), which contains the node files constructed as programs for running implementation in this project. These files are known in this case as `"robot_models_node"` for world model, `"worldsafety"` for all the test cases for world model and `"safetytest"` for safety model all having `cpp` file extension. In addition there also exist a `CMakeLists` file with a `txt` extension in the `zero` folder, which contains all configuration for successfully building the project.

The workspace is then built using command `"catkin build"`. This command runs the essential repositories inside the workspace and also the files in the sub package `zero` according to configuration files and builds the workspace to run the node program. This is all about the structure of the workspace and the packages used to build the program node.

This thesis work also introduces the concept of writing a thesis in the tool `"Latex"` [9]. The tool was written by Leslie Lamport in 1980s and is now a freely downloadable tool for use. This is a plain text editor which has a host of features to write a thesis. The

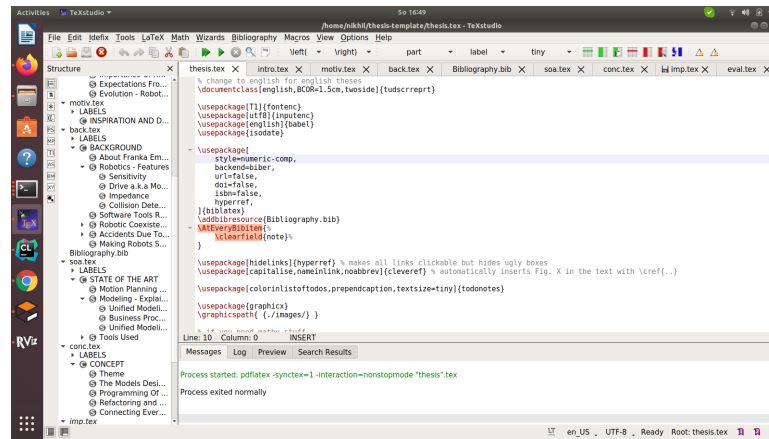


Figure 6.1

user has option to use markup tagging conventions to define the general structure of a document which can be an article, book, or letter. The text can be set with different fonts and enables setting citations and cross-references. This thesis experimented with two versions of latex editors TexStudio and MikTeX and both were very easy to produce an output file in pdf format.

Latex starter repository was forked from Gitlab repository on to local machine. The repository was named as "thesis-template". This contained a structure of thesis plan, essentially a root ".tex" document known as "thesis.tex" which initializes and connects other files for different chapters. The editor TexStudio, lets users set a root document and let users visualize other files. Other files in thesis work were allocated to each of the chapter thus defining the structure of the thesis and they were contained in one folder. Common markups were used to write the text of thesis in Latex which include `\section{}`, `\subsection{}` and many more. Bibliography was set using "Biber" as this is the latest version for setting bibliography format, instead of the older version known as "Bibtex" or "BibLatex", and for this there is a separate file for bibliography identified with a ".bib" extension. The file structure has a root document and other documents which are included in the root document along with much more details in root document, and it links the other documents of chapters to the main root document. All documents have a ".tex" extension. When run, the Latex tool produces a .pdf file in the directory of thesis repository. This is how typical window in tool for Latex looks like.

Here is how a Latex root document looks like and short description for each section.

6.2 WORLD MODEL IMPLEMENTATION

This task is about implementation of the designed world model using C++ code. The program written, described as a node is used to display the values of position and orientation of robot joints, in addition to world model program implementation which include other classes and object values to be added to construct world model node and its features as described previously in theoretical model using UML diagrams.

The program is connected with Rviz motion planner and to Gazebo simulator by

launching a launch file which launches the Rviz ¹ and Gazebo ² and the node file ³. The launch file launches all three components and connects them together i.e. the world model node program and MoveIT motion planner as well as Gazebo simulator. The robot can be moved in MoveIT motion planner using the different colored (Red, Green, Blue, Orange) arrows and once Plan and Execute button is hit, the motion is executed in Gazebo simulator as well, thus confirming movement of a real robot simulation.

The program code is designed to take values from the robot in Gazebo simulator and display them in output console. The program also builds the object structure by initializing values and assigning them values thus constructing the world model class diagram. So after launching the Rviz and Gazebo using launch file, the program is built and run and thus builds world model node and displays the position and orientation values of robot joints as intended in output.

6.2.1 Programming - The Implementation Of World Model

WORLD MODEL - To implement the World Model, files were created each for one of the classes. They were created as files to be included as header files(.h) in the main node program called in this case robot_models_node.cpp. Each header file describe the implementation of one class and its functionalities by using variables which are then objectified in the main node program. This helps to build object structure in the main node program for world model.

The main program is used to add objects to the classes and used to input the values into variables using those objects and then used to run the implementation to display output.

The following paragraphs describe the modules of the program used in implementation.

```
int main(int argc, char** argv)
{
    ros::init(argc, argv, "ROSNODE1");
    ros::NodeHandle nodehandle("namespace");
    ros::AsyncSpinner spinner(1);
    spinner.start();
    ...// code continued
}
```

Here the main function is defined and the ROS node defined and initialized. Also a Node handle is specified and a spin loop is initiated to start the program.

```
abstractobject o1;
graspobject o2;
hand o3;
humanspace o4;
joint o5g;
obstacle o6;
robot robot1;
robotfpe o8;
```

¹<http://wiki.ros.org/rviz/UserGuide>

²<http://gazebo.org/>

³https://git-st.inf.tu-dresden.de/nikaviator/zero/-/blob/master/src/robot_models_node.cpp

```
world o9;
worldobject o10;
...// code continued
```

This is an example of declaring objects for all the classes which are included into the main node file "robot_models_node.cpp", using header files. Here objects are defined which are later used to input values into variables defined in the classes.

```
o8.h->grippos= {12.1, 14.3, 18.2};
o9.c->robname="Robot_FPE";
o9.c->of=true;
o9.c->moving=false;
...// code continued
```

This is an example of allotting values to the variables in the classes, included as header files inside the main program. In most cases pointers are used because they are imperative to implement association and aggregation functionalities in program. Following sections describe association and aggregation relationships and how they are essential in implementing useful functionalities in the project code.

Association is used where classes are together related by a relation which is not inheritance but only associated to each other because they have access to each other's variables. In this relation there is usually a bigger class which needs association relation from a smaller class. The bigger class is a more major part of project, using variables from the smaller class. The bigger class gets access to variables of smaller class and also all the classes inheriting from the bigger class get access to variables from the smaller class. The variables are essential to be described in the smaller class and cannot be directly defined in the major class because it adds up to a clean and consistently understandable code. In this case the class RobotFPE defined by object "o8" is in association relation with class hand and using variable of class hand i.e. grippos using a pointer variable. Similarly class quaternion is associated to worldobject class by association relation because the variables of class quaternion i.e.x,y,z,w are used to define orientation which are used in the class worldobject. The class quaternion.h is included as a header file in class worldobject which then allows access to variables of class quaternion to class worldobject and all classes inheriting from worldobject and this access is done using pointers.

Another relation is aggregation and this is a more critical relation and also use pointer references. It is used where a class variable, if do not exist, must not let its child classes variables be instantiated and used. In this case world class is in aggregation relation to Abstract Object class, humanspace class and robot class. The class Abstract Object which have child classes as Grasp Object and Obstacle class are also related by aggregation to the class world. So this means if the world class were not to exist the Grasp Object and Obstacle Class variables cannot be initialized or used because they use the variables inherited from class Abstract Object which itself cannot exist because the class world do not exist and these two classes are in aggregation relation. The related variables are all linked using pointer implementations and they help to implement both association and inheritance relations. So pointers are used to take input values for child classes i.e. Grasp Object and Obstacle using pointer variables of parent class Abstract Object which are used as a reference with variables of world class.

```
tf2ros::Buffer tfBuffer;
tf2ros::TransformListener tfListener(tfBuffer);
...// code continued
```

This introduces tf2 as a library⁴. tf2 is used to write broadcaster and listener in C++ and Python programs and essentially used to detect position of variable in one frame relative to second frame, all this are relative to standard "world" frame. This lets the program compute difference between two frames relative to world frame and then the information used to match movements from one frame to another thus enabling motion in robot as desired or for example in a game of moving one turtle behind another. tf2 library is used to create three coordinate frames, a world frame which is a standard, then an x frame which suppose can be a keystroke frame, and a y frame which suppose can be system listener frame. tf2 broadcaster is used to publish the x coordinate frame and a tf2 listener listens to it and computes the difference in the frame x and standard world frame and then enables to compute the real difference in position of two frames and this information can be listened by a variable defined in the program. This information can be used to output on console or can be used to match movements of two frames by moving the frame y to a difference relative to world frame thus matching frame x and y. To use tf2 first a buffer is allocated and used. After this a listener object is defined for tf2 which will be later used to listen the values and use them in program.

```
while (nodehandle.ok()) {
for (const auto topic : ROSNODE1::topics)
{
geometrymsgs::TransformStamped transformStamped1;
geometrymsgs::TransformStamped transformStamped2;
geometrymsgs::TransformStamped transformStamped3;
...
// Rest of the statements similar and continue until transformStamped7
}
...// code continued
}
```

while loop is started and objects named transformStamped1...n are defined here one for each joint.

```
try {
transformStamped1=tfBuffer.lookupTransform("world",o5a.name,ros::Time(0));
transformStamped2=tfBuffer.lookupTransform("world",o5b.name,ros::Time(0));
transformStamped3=tfBuffer.lookupTransform("world",o5c.name,ros::Time(0));
...
// Rest of the statements proceed until transformStamped7 object and o5g joint
}
catch (tf2::TransformException &ex) {
ROS_WARN("%s", ex.what());
ros::Duration(0.1).sleep();
continue;
}
```

A try catch block is used to compare and connect the attribute of robot for which we seek values i.e. in this case joint 1...7,relative to standard world and store them in the defined objects transformStamped1,2,3....

Then exceptions are caught in catch block and some statements are defined for this .This is in case some things don't work. The objects store the values in tfBuffer after the lookup command compares the joint values with standard frame values and computes the difference before storing the final position in the object variables.

```
o5a.p1[0]=transformStamped1.transform.translation.x;
o5a.p1[1]=transformStamped1.transform.translation.y;
o5a.p1[2]=transformStamped1.transform.translation.z;
```

⁴<http://wiki.ros.org/tf2/Tutorials/Introduction>

```

ROSINFOSTREAM("pandalink1_Position_is"<<"x="<< o5a.p1[0]<<" ,y="<<o5a.p1[1]<<" ,
z="<<o5a.p1[2]);

o5a.or2.w=transformStamped1.transform.rotation.w;
o5a.or2.x=transformStamped1.transform.rotation.x;
o5a.or2.y=transformStamped1.transform.rotation.y;
o5a.or2.z=transformStamped1.transform.rotation.z;
ROSINFOSTREAM("pandalink1_Orientation_is"<<"w="<< o5a.or2.w<<" ,x="<<o5a.or2.x<<" ,
y="<<o5a.or2.y<<" ,z="<<o5a.or2.z);
...// code continued

```

Here in these steps, the object variables of the program take input the objects values from object transformStampedx and store them in the node program variable and which then outputs these values.

This program thus listens to values from simulation and displays using appropriate output statements. The robot in Gazebo is thus connected to Rviz motion planner using ROS which enables the node program `robot_models_node.cpp`, to listen to desired values in this case position and orientation of joints .

6.3 Future Work Implementing Application Model Programmatically

The thesis work i only completed up to implementing world model and safety model. The application model is left pending to be completed in future.

7 EVALUATION

This section demonstrates the results for the program implemented in the previous chapter as well as the results for the test cases implemented.

7.0.1 World Model Implementation Results

The implementation section gives out results for the Position and Orientation of Joints of Robot Franka Panda Emika .

They are 7 in number but only about 3 are shown here as a sample. Here are the results .

```
pandalink1 Position is x=0,y=0,z=0.333
pandalink2 Position is x=0,y=0,z=0.333
pandalink3 Position is x=-0.120566,y=-1.81861e05,z=0.625095
....//code continued
pandalink1 Orientation is w=1,x=0,y=0,z=7.54195e05
pandalink2 Orientation is w=0.693616,x=-0.693595,y=-0.137573,z=-0.137468
pandalink3 Orientation is w=0.980906,x=1.16121e05,y=-0.194483,z=8.93912e05
....//code continued
```

The robot was moved in Gazebo Simulator after a motion planning trajectory execution in Rviz motion planner. The node was again built and run giving new values for position and orientation of joints.

```
pandalink1 Position is x=0,y=0,z=0.333
pandalink2 Position is x=0,y=0,z=0.333
pandalink3 Position is x=-0.00290409,y=0.296506,z=0.442232
....//code continued
pandalink1 Orientation is w=0.703636,x=0,y=0,z=0.710561
pandalink2 Orientation is w=0.120731,x=-0.695508,y=-0.127549,z=0.696724
pandalink3 Orientation is w=0.443721,x=-0.314109,y=0.478016,z=0.68989
....//code continued
```

This shows a changed set of values for position and orientation for all joints of Robot Franka Panda .

7.1 World Model Test Implementation

The world model program implementation node constructed and explained in above section and known as `robot_models_node`. This program needs to be tested for some

key points. For this another program called a node "worldsafety" is constructed. There are many tests done for this program which include various tests which are described in following sections.

Static tests

After this stage the program is once again built and run and this now displays the new changed values for robot joints position and orientation in Gazebo simulator. The results are then shared in evaluation section. More details about the tools and attributes they can handle are described below.

Bibliography

- [1] A. McMurrey" "David. *Features of Industrial Robots*. URL: https://www.tu-chemnitz.de/phil/english/sections/linguist/independent/kursmaterialien/TechComm/acchtml/class_ex.html.
- [2] Souri Alireza, ali Sharifloo Mohammad, and Norouzi Monire. ""Formalizing Class Diagram In UML"". In: (2007).
- [3] Kaur Amandeep and Kaur Manpreet. ""Analysis of Code Refactoring Impact on Software Quality"". In: (2016).
- [4] A Kulkarni Amith et al. ""Recent Development of Automation in Vehicle Manufacturing Industries"". In: (2019).
- [5] Dhillon B.S. *"Robot Reliability and Safety"*. Springer Verlag, 2015.
- [6] Siciliano Bruno, Sciavicco Luigi Lorenyo, and Villani Giuseppe Oriolo. *"Robotics Modelling, Planning and Control Advanced Textbooks in Control And Signal Processing Book "*. Springer, 2009.
- [7] Siciliano Bruno, Sciavicco Lorenzo, and Villani Giuseppe Oriolo Luigi. *"Robotics : Modelling Planning and Control "*. Springer, 2010.
- [8] CeTi Book. URL: https://wwwpub.zih.tu-dresden.de/~scheuner/ad0bcdbfabcb688174b767fb01/content/files/CeTI_Book_v18.pdf.
- [9] *Creating a document in LaTeX*. URL: https://www.tu-chemnitz.de/phil/english/sections/linguist/independent/kursmaterialien/TechComm/acchtml/class_ex.html.
- [10] Rus Daniela and T.Tolley Micheal. ""Design, fabrication and Control of Soft Robots"". In: (2015).
- [11] M. Ebert Dirk and D. Henrich Dominik. ""Safe Human-Robot-Cooperation: Image-based collision detection for Industrial Robots"". In: (2002).
- [12] Matheson Eloise et al. ""Human–Robot Collaboration in Manufacturing Applications: A Review"". In: (2019).
- [13] Sabah Al-Fedaghi. ""Diagramming the Class Diagram: Toward a Unified Modeling Methodology"". In: (2017).
- [14] GmbH Franka Emika. *"Franka Panda User Guide"*. Franka Emika GmbH, 2018.
- [15] Gołda Grzegorz, Kampa Adrian, and Paprocka Iwona. ""Analysis Of Human Operators And Industrial Robots Performance And Reliability"". In: 9 (2018).
- [16] Zhang Hong. ""Visual Motion Planning for Mobile Robots"". In: 18 (2002).

- [17] M da Fonseca Ijar and N Pontuschka Maurício. ""The State-of-the-art in Space Robotics"". In: (2015).
- [18] Edward Colgate J., Wannasuphoprasit Witaya, and A. Peshkin Michael. ""Cobots: Robots For Collaboration With Human Operators"". In: (1996).
- [19] Osis Janis and Donins Uldis. ""Formalization of the UML Class Diagrams"". In: (2010).
- [20] Cheng Jingyuan, Chen Xiaoping, and Lukowicz Paul. ""Towards Coexistence of Human and Robot: How Ubiquitous Computing Can Contribute?"". In: (2014).
- [21] Wienke Johannes and Wrede Sebastian. ""Failures In Robotics And Intelligent Systems"". In: (2017).
- [22] J. Craig John. *"Introduction To Robotics "*. Vol. 3. Pearson Education International, 2005.
- [23] Kim Junsu, Moon Hongbin, and Jung Hosang. ""Drone-Based Parcel Delivery Using the Rooftops of City Buildings: Model and Solution"". In: (2020).
- [24] D. Riek Laurel. ""Healthcare Robotics"". In: (2017).
- [25] Görner Michael et al. ""Movelt! Task Constructor for Task-Level Motion Planning"". In: (2019).
- [26] Peshkin Michael and Edward Colgate J. ""Feature: Cobots"". In: 26 (1999).
- [27] Vasic1 Milos and Billard Aude. ""Safety Issues in Human-Robot Interactions"". In: (2013).
- [28] Visinsky Monica, D. Walker Ian, and R. Cavallaro Joseph. ""Fault Detection and Fault Tolerance in Robotics"". In: (1992).
- [29] Ben-Ari Mordechai and Mondada Francesco. *"Elements of Robotics"*. Springer, 2018.
- [30] Quigley Morgan et al. ""ROS: an open-source Robot Operating System"". In: (2009).
- [31] Nikolaus Muellner. ""Three Decades after Chernobyl: Technical or Human Causes?"" In: *The Technological and Economic Future Of Nuclear Power* (2019).
- [32] Koenig Nathan and Howard Andrew. ""Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator"". In: (2004).
- [33] B.M. de Koster René. ""Automated and Robotic Warehouses: Developments and Research Opportunities"". In: (2018).
- [34] H Hosseini S and M Goher K. ""Personal Care Robots for Older Adults: An Overview"". In: (2016).
- [35] Ahsan Badruddin S. and M. DildarAli S. ""Recent Developments in the Optimization of Space Roboticsfor Perception in Planetary Exploration"". In: (2015).
- [36] Robla-Gomez S. et al. ""Working Together: A Review on SafeHuman-Robot Collaboration inIndustrial Environments"". In: (2017).
- [37] Haddadin Sami et al. ""On Making Robots Understand Safety:Embedding Injury Knowledge Into Control"". In: (2007).
- [38] A. White Stephen. ""Introduction To BPMN"". In: (2004).
- [39] Leigh Anderson Susan. ""Asimov's "Three Laws of Robotics" and Machine Metaethics"". In: (2005).
- [40] Kerezović Tanja et al. ""Human Safety In Robot Applications – Review Of Safety Trends"". In: (2013).

- [41] Niemueller Tim and Widyadharma Sumedha. *"Artificial Intelligence – An Introduction to Robotics"*. 2003.
- [42] Mens Tom. *"A Survey of Software Refactoring"*. In: (2004).
- [43] Gao Yang and Chien Steve. *"Review on Space Robotics: Towards Top-Level Science through Space Exploration "*. In: 2 (2017).

Statement of authorship

I hereby certify that I have authored this Master Thesis entitled *Design and Implementation of a Model-based Architecture for Cobotic Cells* independently and without undue assistance from third parties. No other than the resources and references indicated in this thesis have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the intellectual preparation of the present thesis. I am aware that violations of this declaration may lead to subsequent withdrawal of the degree.

Dresden, 27th October 2020

Nikhil Ambardar