ROBOTICS RESEARCH LAB
DEPARTMENT OF COMPUTER SCIENCES
UNIVERSITY OF KAISERSLAUTERN

Documentation

Script for the Lecture Foundations of Robotics

Prof. Dr. Karsten Berns

Contents

1. Introduction and Motivation

1.1 Robotics - An Interdisciplinary Research Area

In robotics, a variety of topics from engineering and natural science come together. To develop the electromechanical components of a robot, contributions in the fields of mechanics, materials science, design theory (sub-areas of mechanical engineering), control engineering, circuit design, and microelectronics (sub-areas of electrical engineering) are needed. This branch of robotics is also called *mechatronics*.

In the last 10 years, technical questions, in particular the mechanical design and the control design for kinematic chains, have been thoroughly explored and answered. The result of this work is very fast and precisely working robot arms, wheel, chain-driven platforms, and walking and climbing robots. Real-time computation of regulation and control algorithms require fast computer architectures that capable of processing complex data types in particular very quickly.

From the process computer point of view, a robot consists of a large number of signal sources and sinks, that allow the computer to perform a fast input/output action for signal processing and for require process control. In robotics research, many questions are often at the forefront. Such as under what conditions and with what algorithmic effort a robot can work in a real environment, can carry out the tasks assigned to it independently and correctly, and how it can react to unforeseen events.

The capabilities of a robot for free locomotion and manipulation in an uncertain actual environment are the subject of approaches to autonomous systems. For this purpose, the modeling of intelligent system behavior, cognitive skills, and motor skills are required. As a first approximation, achieving the autonomy of robots through the interaction of three system components made possible. These are: automatic planning of handling sequences and action sequences using task description, planning knowledge and environmental data, automatic execution of the action plans and control of the interaction of the robot with the real environment as well as monitoring of the executions of the plans by manufacturers interrelationships between actions and recorded sensor signals as well as detection of conflicts, errors, and their elimination. See figure 1.1

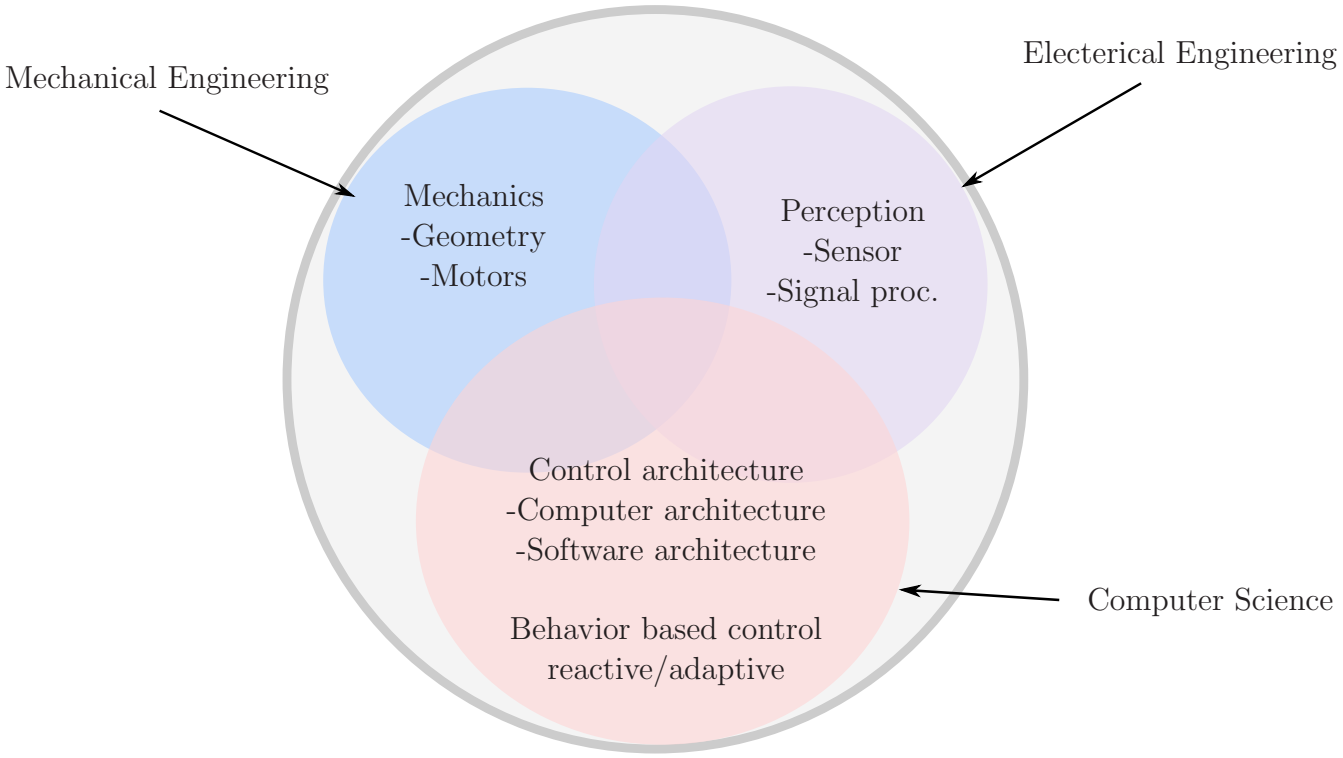


Figure 1.1: Interdisciplinary Research Area

Tools that in their origin are used in the research field of artificial intelligence to support knowledge acquisition, knowledge presentation, rule-based reasoning, induction and deduction, now are used to interpret impressions and dynamic processes as well as methods of machine learning. While the first models of autonomous robots are still on cubes of approximated worlds existed today we work on the modeling and processing of real world views. These relate in particular to the processing of via sensors signals that are taken from the physical environment, as well as their interpretation and the creation of causal relationships between action and reaction. In all of these approaches, the processing of noisy and fuzzy sensor data creates problems.

Inferential mechanisms and planning processes have to be extended to include the dimension "time" since their temporal logic plays a very important role in action chains and sensor data change their meaning depending on current operations. Since not all possible situations are modeled today, simulation systems are used to predict the results and situations provided by the robot's action plan. This results in the possibility of calculating the initial state, intermediate states, and target states of a robot action chain or a mission in advance as a model and, through targeted sensor measurements, a comparison of to undertake the expected situation and the real situation. A deterministic goal-oriented behavior of the robot is assumed here. One also speaks of so-called model-based autonomous robot systems. Event-controlled systems exist when the robot is supposed to operate in an unknown environment, and also non deterministic behavior can be planned. Such systems exhibit reactive behavior, i.e. the robot must have a reactive behavioral repertoire, e.g. avoid obstacles, unknown objects map, or discover its surroundings. Another question concerns the knowledge acquisition of such systems, i.e. questions of machine learning. Here lies the motivation is based on experience or through targeted training (partly supported by

a human operator) new skills and abilities (also called skills) and to implement them in control structures.

Another important area of robotics deals with the question of the system architecture of an intelligent robot. The architecture of robot systems can be analogous to the consideration of computer architectures can be seen. Robots are by their operating principle for the hardware and software and the structure of their construction from the individual hardware resources characterized. Their operating principle defines their functional behavior of the robot by establishing an information structure and a control structure. The information structure of the robot is determined by the type of information components as well as their internal representation and the amount that can be applied to them operations defined. The information structure of a robot can be specified as a set of abstract data types.

The control structure of a robot is determined by the specification of the algorithms for the interpretation and transformation of the information components of the robot. Important principles of modern robot architectures are those of explicit parallelism and a hierarchization of information components and the operations applied to them. In addition to strongly hierarchically structured architectures, behavior-oriented system approaches are increasingly being pursued, in which predominantly distributed or highly parallel reflexes and behavior patterns exist.

To support the programming, commissioning, and testing of industrial robots, geometric CAD modeling tools, planning tools for production planning, layout planning, and action planning worked. Today's advanced robot programming systems mostly consist of a CAD modeling system, an interactive graphic programming system, a graphic simulation system, a technical database, and a preprocessor and postprocessor for generating executable Program codes. Preprocessors generate a robot-independent intermediate code from a robot program defined in a high-level language, while postprocessors generate a robot-specific code generate code. Using an intermediate language, robots of different manufacturer types can be programmed. The problem of the description of physical and technical processes such as assembly, thermal welding processes, bending processes, and the handling of elastic materials.

Not only the engineering sciences mechanical and electrical engineering as well as computer science play a major role in the further development of robotics, but also research work in biology and medicine. Lately, it has been shown more and more that the methods of classical robotics do not even come close to the performance natural systems can achieve. Starting with studies of the fundamentals of locomotion control in animals, movement control and coordination in humans up to obtaining information from a large number of fused sensory impressions, concepts are developed to transfer them to robot systems.

1.2 Terms Definition

In the following, some elementary basic terms from the research area of robotics will be introduced.

1.2.1 Robot

Around 1920, Karel Capek defines the term *robota*, Slovakian word for an artificial being created to do hard/Heavy work. For Capek, a robot (unlike humans) is restlessly working. He developed a chemical substance for the production of robots, based on the rules that robots should serve people and do heavy work. In the end, he developed "perfect" robot, it does not obey humans anymore, and even they end up rebel and kill entire human life. [1]

The writer Asimov, who has written a multitude of robot stories, first published the in *Runaround* in 1942 three Assimov's Laws of Robotics, which are also valid for current robot research: [2]

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.

2. A robot must obey orders given it by human beings except where such orders would conflict with the first law.

3. A robot must protect its own existence as long as such protection does not conflict with the first or second law.

Ther are some general definitions for the term robot such as working machine, artificial human, combat/war machine. In science siction it calles a thinking machine.

There are also some technical definitions:

- Flexible manipulation apparatus (grippers/sensors)
- Machines that receive information (sensors) and has an impact on its environment (actuators)
- Machine with the ability to move itself and/or other objects

Robot – Machine Man (cyborg); Movements caused by electric waves (wireless), and electronically controlled automaton.[3]

A robot is defined as a reprogrammable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks.[4]

A machine resembling a human being and able to replicate certain human movements and functions automatically. A machine capable of carrying out complex series of actions automatically, especially one programmable by a computer[5].

[1] Rossum's Universal Robot from Karel Capek.
[2] Isaac Assimov All robot stories Bastei Lübbe Verlag ISBN: 3-404-24082
[3] Kleines Lexikon der Büchergilde 1973
[4] The Robot Institute of America, 1979
[5] Oxford Dictionary

1.2.2 Robotics

The development, control and application of robots is summarized under the term *Robotics*.

According to *NEUMANN, Lexikon der Informatik und Datenverarbeitung*:

> Robotics is an interdisciplinary field of research in which mechanical devices and suitable control units are at the center of complex tasks.
> Although robots are mostly presented with human like shape and sensory capabilities robotic science fiction, the robots used in practice are stationary manipulators that can be used by programming for changing industrial tasks, e.g. welding or painting work in the automotive industry.

According to the VDI guideline 2860 (1990), a robot is defined as follows: [6]

> A robot is a reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks.

This definition describes excellent industrial robot systems; however, it is less suitable for describing the robots currently being developed in research.

The definition of the term robot given in [Christaller 02] reflects the current state of development much better:

> Robots are sensorimotor machines that expand human skills. They consist of mechatronic components, sensors and computing architecture as well as a complex control. The complexity of a robot differs significantly from other machines due to the greater number of degrees of freedom and the variety of behaviors.

1.3 Skills and Components

A robot should be able to move itself and/or other objects.

The components of a robot are:

- Arms, wrists, and effector for object manipulation
- Wheels, legs or similar devices for mobility
- Actuators and controllers for joint movements
- Computer and complex software systems for decision making and control
- Sensor system for the measurement of the robot state and its environment

[6] VDI- guideline 2860: Assembly and handling technology; Handling functions, - facilities, terms, definitions, symbols; Düsseldorf 1990

1.3.1 Sensor System

The responsibilities are sensor system are:

- Measurement of contacts, forces, torques
- Position, velocity, acceleration
- Temperature, electrical voltage
- Distance measurement
- Shape, color, size and motion detection
- Texture, smell
- Detection of sound waves and voices

With regard to their locomotion properties, robots can be divided into two classes: on one hand, stationary systems, such as robotic arms that perform manipulation tasks in a work cell, and on the other hand, mobile systems, such as inspection robots for evaluating the condition of oil pipe lines.

Another classification criterion is the degree of autonomy. Starting with simple master/slave systems, in which a human operator specifies the movements of a robot, through semi-autonomous up to completely autonomous systems, who have to determine their behavior themselves depending on the situation. Especially in research, one is concerned with the development of autonomous robots, such as humanoids, which are supposed to replace humans in certain applications.

1.4 History of Robotics

The history of robotics or the robot-like automata is very old. As early as the 3rd century BC, an invention attributed to Heron of Alexandria Astonished and Astonished people. After the fire had burned for some time, those who offered sacrifices on an altar saw wine pouring onto the fire. The drive principle was pneumatic and hydraulic. The automatic Altar of Heron of Alexandria. In this Greek theater there was a statue of Bacchus that could rotate. At a certain point in front of an altar, fire was spat out. Wine and milk poured from the altar and little figures began to dance. (See figure 1.2)

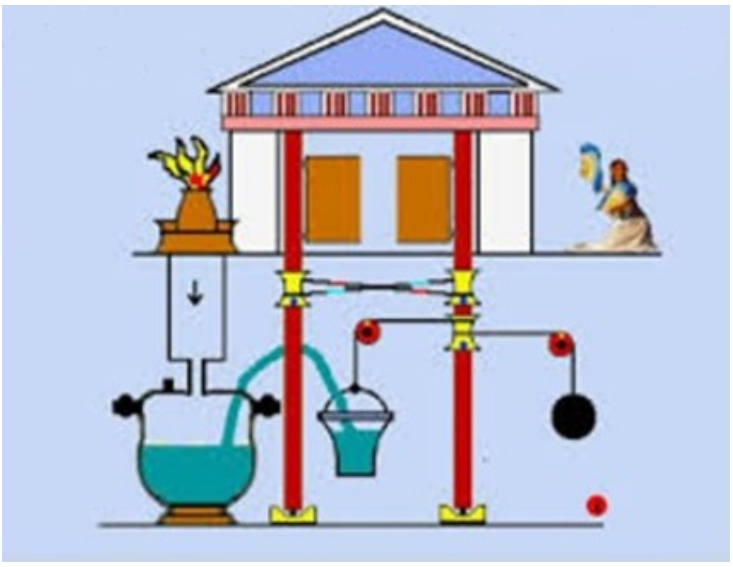


Figure 1.2: Heron Automatic Theater

The first research work on walking machines took place as early as 231, when the Chinese general Zhu Ge-Liang had a mechanical, wooden, four-legged device built. The aim of this development was to carry out food transports for the army. According to tradition, the machine should be able to carry 200-250 kg and reach a speed of 10 km per day.

The 18th century was a new era for the development of robotic systems. In 1738 the French engineer Jaques de Vaucanson presented his famous duck. Its mechanism consisted of over 1000 parts, partly housed in a pedestal. The mechanical duck was able to eat and excrete excrement. For its time, it was a truly amazing device.

Other machines were the flute player and the tambourine player. Tambourine player was a musical doll in human size that was able to Blow in the flute, and change shape of its lips and tongue and also move its finger.

In 1774, Pierre Jaquet-Droz, a Swiss watchmaker, developed three androids painter, writer and musician. The writer, who was sitting on a chair, was holding a goose feather in his right hand, and his left hand was resting on a little table. As he wrote, his eyes followed the letters. When realizing this extremely complex mechanical mechanism, Pierre Jaquet-Droz had to solve the problem of space (the writer was the size of a three-year-old child) and the control task movement control and coordination of arm, elbow and eyes. He achieved this by rotating a cylinder with three rows of cams attached to its circumference. In order to write the letters in basic and fine lines, the pen moved not only horizontally, but also vertically. See figure 1.3

Figure 1.3: Painter, writer, Musicion

Around 1805 Joseph Maria Jacquard invented the programmable loom. That was able to be controlled by punch cards and regarded as a specialized robot. A large number of automatons were created in the years that followed, such as, in 1810 Johann Gotfried and Friedrich Kaufmann developed a Trumpeter, in 1830 Christopher Spencer built a cam-controlled turning lathe, and in 1938 Williard Pollard and Harold Rosel bulit a programmable spraying machine.

At the beginning of the 20th, some robots were built such as Televox in Pittsburgh in 1927. Monitored water tanks in a high-rise building and switched on pumps if necessary. He was able to answer telephone inquiries about the water level. He was also able to run a vacuum cleaner and a fan, turn lights on and off, and open windows and close doors. Sabor II, which was developed in 1930, was used as an entertainment robot. Controlled by radio commands, Sabor was able to move his limbs, carry out various handouts and speak with the help of a built-in record player. Like Sabor, the robot Mr. Ohm Kilowatt (1933) was shown around as a technical curiosity and was also able to answer questions whispered in his ear.

The actual development of industrial robots began in the USA in 1954. George C. Devol applied for a patent for a programmable manipulator, which was granted in 1961 under patent number 2988237.

In 1956 Devol and Joseph Engelberger met and founded the company Unimation Inc. For the first time a market study was carried out for the use of robots, the 15 automobile assembly plants and another 20 others. This market study resulted in a compulsory booklet that led to the first functional prototype of an industrial robot in 1959. It powered hydraulic and controlled by a computer. Also in 1959 the company Planet Corp. presented the first commercial robot. This was controlled mechanically by cams and limit switches.

In 1961 Ford installs robot of type „Unimation“.

In 1968 the mobile robot Shakey was developed at the Stanford Research Institute (SRI). It was equipped with a large number of sensors, including cameras, pushbuttons, and was able to create a plan of its own to get from one room to another, taking into account obstacles. See figure 1.4

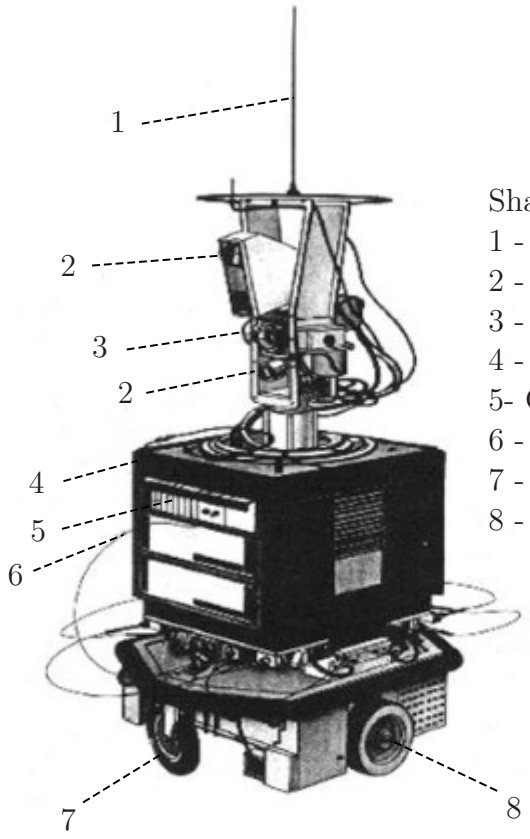


Figure 1.4: Robot Shakey

At the beginning of the 1970s, the first industrial robot installation in Germany at Daimler-Benz in Sindelfingen for welding the side panels of the S-Class is made. For this, 12 unimate robots were installed. The industry had very high expectations of these devices. Since then, numerous companies have emerged that offer high-performance robot systems to this day, such as the KUKA company.

In 1973 the first programming language for robots was developed at the Stanford Research Institute (SRI). The language AL followed in 1974. Ideas of this language were later developed in the programming language VAL from Unimation used. In 1975, the first fully electrical powered robot was made.

In 1978 the robot PUMA (Programmable Universal Machine for Assembly) was presented by Unimation. It is electrically powered and based on a general motors concept. Today robots are part of the basic equipment of every modern production line, especially in the automotive industry.[7]

In 1984, Ichiro Kato developed Wabot-2 at Waseda University Tokio. See figuer 1.5

[7]A list of robot companies with their products that offer their products in Germany can be found at http://www.roboter-info.de/deutsch/uebersicht.html

Figure 1.5: Wabot-2

In 1985 3-Finger Salisbury-Hand was developed in Massachusetts Institute of Technology. See figure 1.6

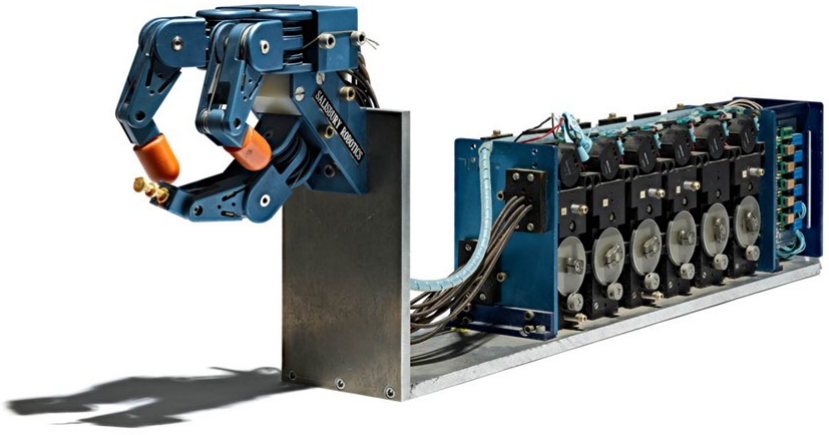


Figure 1.6: 3-Finger Salisbury-Hand

1.5 Steps in Robot Development

1. Programmable manipulators (1960 – 75)
    - Low computing power
    - Only fixed breakpoints (point-to-point programming)
    - Nearly no sensors involved (only pick-and-place)
2. Adaptive robots (1976 – 1982)
    - More sensors (e.g. cameras)
    - Adaptation to the environment
    - Use of high level programming languages (e.g. VAL)

- Low robot-intelligence (adaptive task performance)

3. Autonomous robots (since 1983)
   - High computing power (multi-processor systems)
   - Task-oriented programming
   - Demand for (machine) autonomy
4. Humanoid robots (since 1995)
   - High flexibility regarding environment and task
   - Learning ability and adaptability/flexibility
   - Self reflection
   - Emotion
5. Service robot as products (since 2000)
   - E.g. vacuum cleaner, mover, inspection machines
   - Cheap and reliable

Figure 1.7 shows the application area of robots:

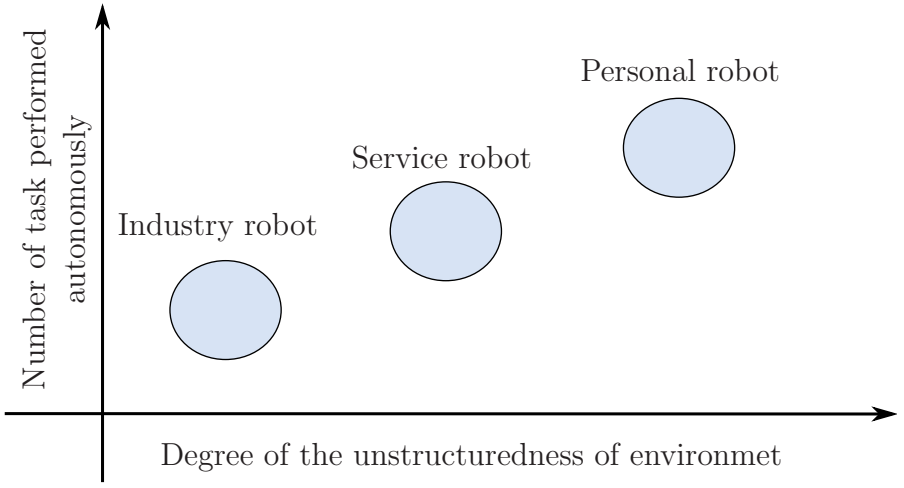


Figure 1.7: Application Areas

1.6 Industrial Robot

In industry, robot systems are mainly used in mass production for handling and monitoring tasks. A manipulating industrial robot is defined as follows according to World Robotics 2003 [8]

[8]International Federation of Robotics, United Nations, New York and Geneva,

> An automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes, which may be either fixed in place or mobile for use in industrial automation applications.

Classification is based on:

- Number of axes (3, 4, 5, ...)
- Type of control (PTP, cont. path, adaptive, tele-operative)
- Mechanical structure (SCARA, parallel, ...)

Some application areas of industrial robots are electronic board, welding, spraying, autonomous transport systems, and handling of hazardous materials.

The properties are:

- Usually stationary
- Less degree of freedom
- Easily programmable
- No perception of the environment
- Highly specialized
- Higher efficiency in cost and time compare to humans

1.6.1 Statistics of Industrial Robots

Figure 1.8 shows the worldwide annual supply of industrial robots from 2009 to 2017 and the prediced ammount for 2018 to 2021. It is predicted to increase about 30% in 2017, 10% in 2018 and 14% in average per year from 2018 to 2021.

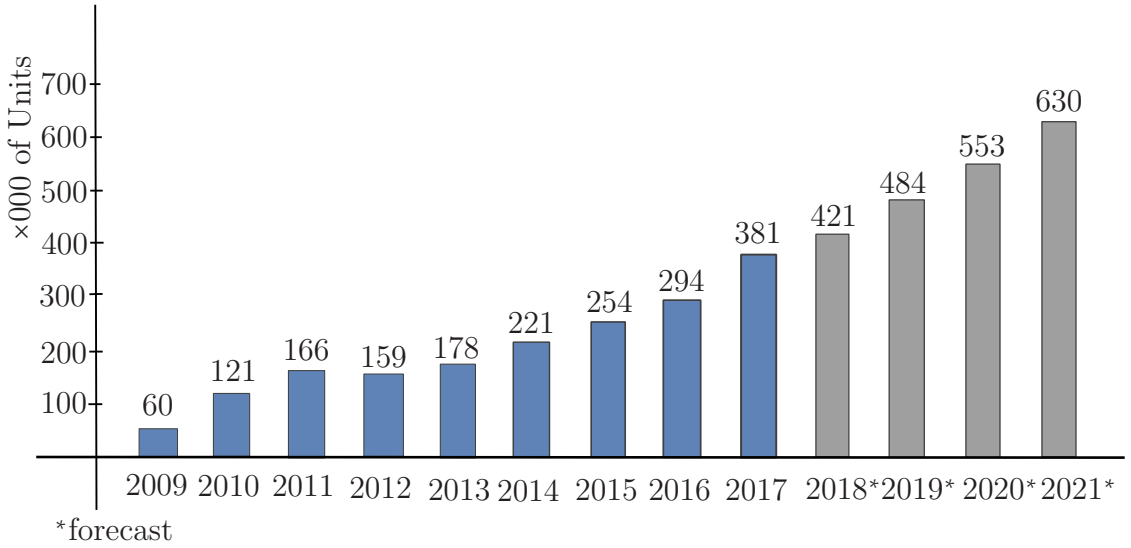


Figure 1.8: Worldwide annual supply of industrial robots 2009-2021*

1.6.2 Service Robot / Personal Robot

According to Word Robotics 2003, *Service Robot* is defined as a robot which operates semi- or fully autonomously to perform services useful to the well-being of humans and equipment, excluding manufacturing operations.

Classification is based on servicing humans, servicing equipment and others.

Application Areas are:

- Industrial robots
- Underwater robots
- Robots in construction and mining
- Robots in agriculture and forestry
- Computer based robotics assistance systems
- Personal robots in private environments
- Robots in operations
- Garbage collection
- Pipe inspection and maintenance
- Entertainment
- Elderly Care

2. Subsystems and Components of a Robot

In this chapter, the subsystems that make a robot up are presented. First, the technical structure of robots is introduced. For this purpose, joint types, the basic configuration for robots, and the drives that are often used are presented and the workspace problem is discussed. Electric drives and their power transmission are covered in more detail because they are most commonly used in robotic systems.

2.1 Mechanical Components

2.1.1 Workspace

Workspace consists of all points in the three-dimensional space, which are reachable by the robot hand. This requires three degrees of freedom in movement, i.e. at least three basic joints are necessary for a 3-D space.

The design of the robot arms and their joints is the *workspace of a robot*. Basic shape of workspace consists of all points, which are reachable by the robot hand without considering any restrictions by the joints or obstacles.

Depending on the basic form of the work area, we differentiate between the following basic configurations for industrial robots.

2.1.2 Basic Joint Types of Robotic Systems

The movements of a robot are only possible through its joints. A robot consists of a series of links that are connected by joints. The 4 most important joint types according to [Groover 87] are:

Rotational Joint (R) The axis of rotation forms a right angle with the axes of the two connected links. See figure 2.1.

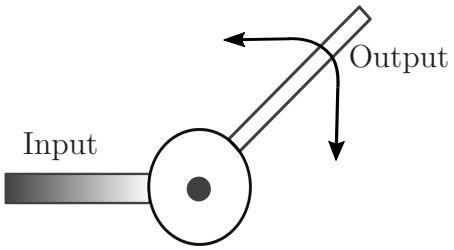


Figure 2.1: Rotational Joint

Torsion Joint (T) The axis of rotation of the torsion joint is parallel to the axes of the both links. See figure 2.2.

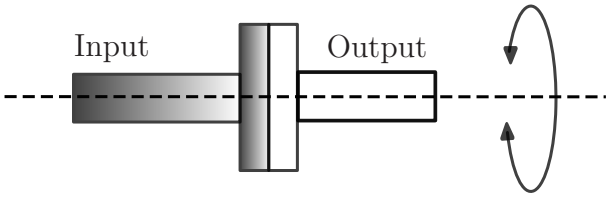


Figure 2.2: Torsion Joint

Revolute Joint (V) The input element runs parallel to the axis of rotation, the output element is at right angles to the axis of rotation. See figure 2.3.

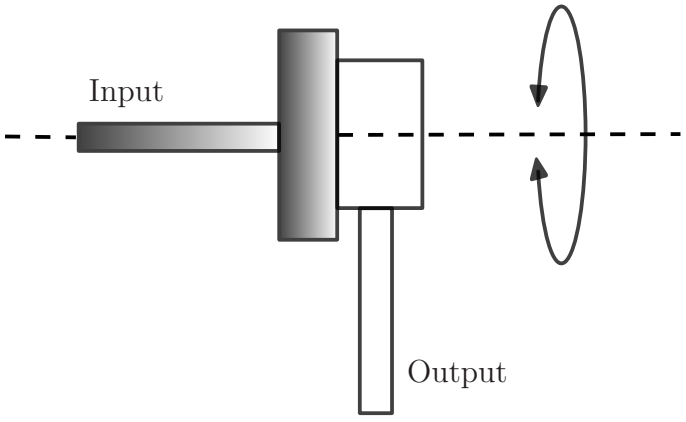


Figure 2.3: Revolute Joint

Linear Joint (L) Also translation joint, sliding joint or prismatic joint. Linear joints cause sliding or advancing movement along an axis. See figure 2.4.

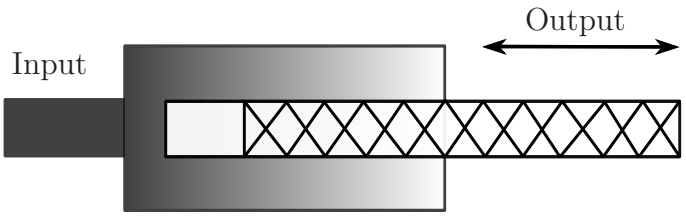


Figure 2.4: Linear Joint

2.1.3 Basic Robot Types

Cartesian Robot (LLL) The basic shape of operational space is a cuboid. See figures 2.5.

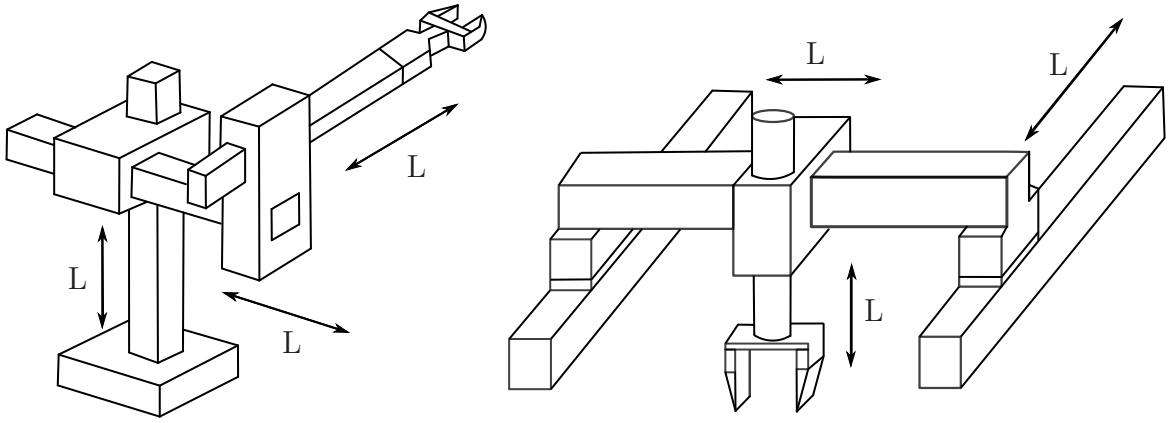


Figure 2.5: Cartesian Robot (LLL)

Robot Arm (LVL) Basic shape of operational space is a cylinder. Other possibilities are TLL, LTL. See figure 2.6.

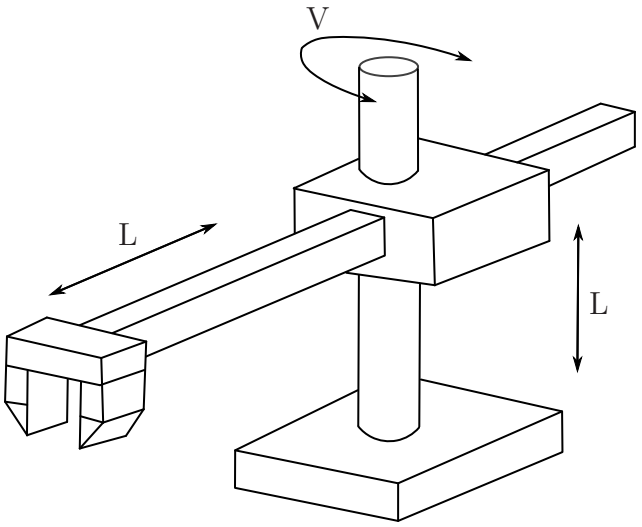


Figure 2.6: Robot Arm (LVL)

SCARA-Robot (RRLT) The basic shape is a cylinder. The SCARA robot (Selective Compliance Assembly Robot Arm) was developed for assembly in Japan in the early 1980s. Its mechanical structure was determined to move in the z-direction, but compliant in the x, y-plane. It is therefore particularly suitable for inserting objects with an insertion movement in the z-direction. See figure 2.7.

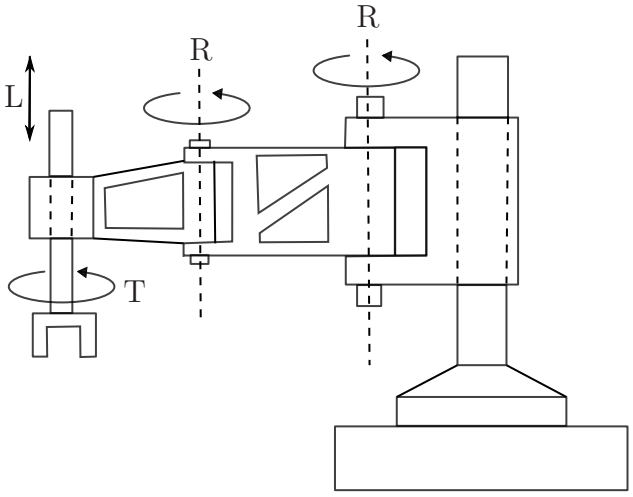


Figure 2.7: SCARA Robot (RRLT)

Universal Robot Arm (TRR) The basic shape of the operational space is a sphere.

Under the basic shape of the operational space we mean that operational space that would result if the mutual hindrance of the robot's arms and the limitations of the joint angles were not taken into account. See figure 2.8.

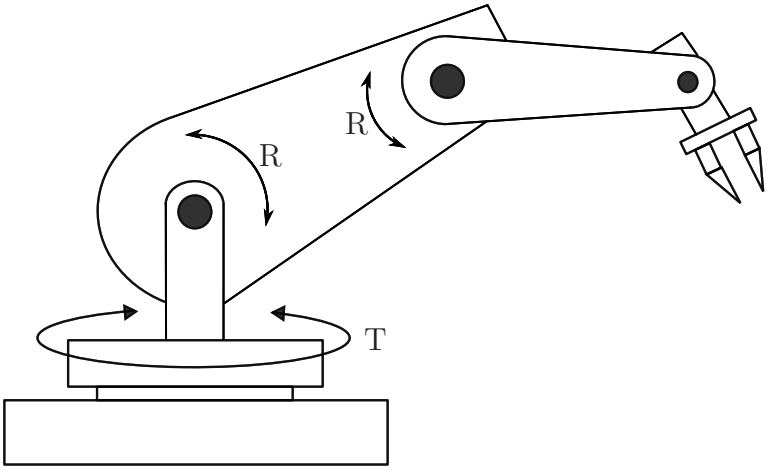


Figure 2.8: Universal Robot Arm (TRR)

2.1.4 Robotic Wrists

The robot should do work, for example it should grip objects. For this, it is not sufficient just to approach a suitable point in space corresponding to the position of the object. Rather, it is necessary to orient the gripper in the appropriate gripping direction. A robot position therefore consists of the position and the orientation of the gripper. In order to be able to orient the gripper, wrists are connected after the last robot arm. The last wrist has a flange for coupling an effector. Examples of effectors are: grippers, screwdrivers, welding tongs. The wrists, including the flange, are also known as the wrist. The orientation in space is described by three angles. Three degrees of freedom are therefore required to set any orientation. These can be reached through three joints, the wrists. These wrists have not yet been shown in the basic configurations for robots that have already been presented. Two basic forms are common for building a wrist. These are shown in Figure 2.9.

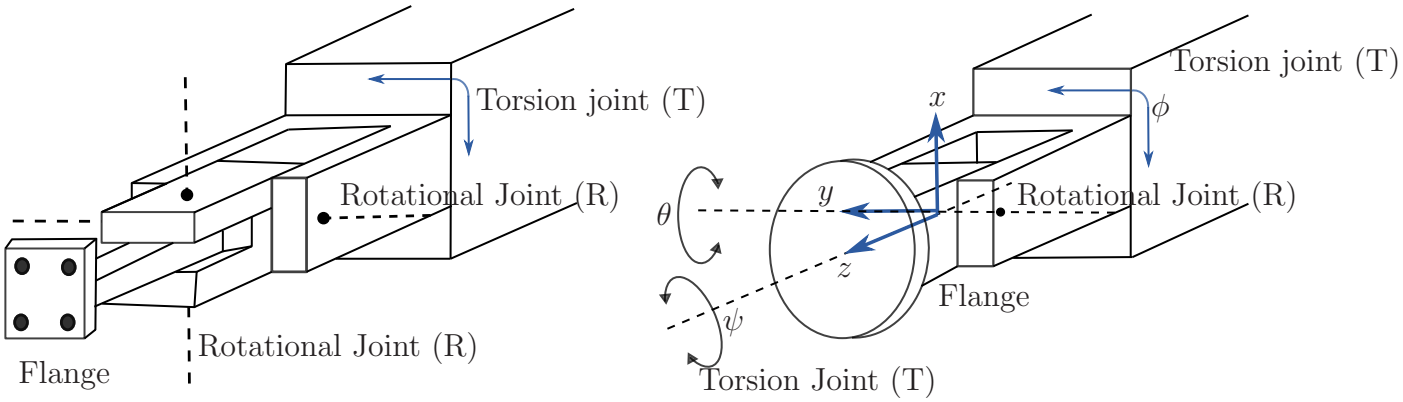


Figure 2.9: Basic wrist shapes

2.1.5 Degrees of Freedom and Joints

The *degree of freedom* is determined as the number of possible independent movements of an object in relation to a fixed coordinate system. The position of an object that can move freely in space is defined by its position. A *position* is determined by the position (3 values) and the orientation (3 values). Overall, the position is described by six values, i.e. the degree of freedom of an object in space.

The degree of freedom of a robot is the number of joints. The following applies:

1. In order to achieve the degree of freedom, at least joints are required.
2. Rotary joints are required for orientation, as liner joints would not change the orientation of the wrist.

A link does not bring an additional degree of freedom in all arrangements. For example, Two successive linear joints, the thrust axes of which point in the same direction, are equivalent to a single linear joint in terms of the degree of freedom (telescopic antenna). Since it is the maximum degree of freedom, more than 6 joints do not bring any additional degrees of freedom.

In addition to the degree of freedom, other aspects play a role in applications that may make the use of robots necessary. For example, an eight-axis robot Can be used in a space with obstacles reaching around obstacles. It achieves points that a six-axis robot (2.10) can theoretically achieve are also accessible, but with the special location of the obstacles become inaccessible.

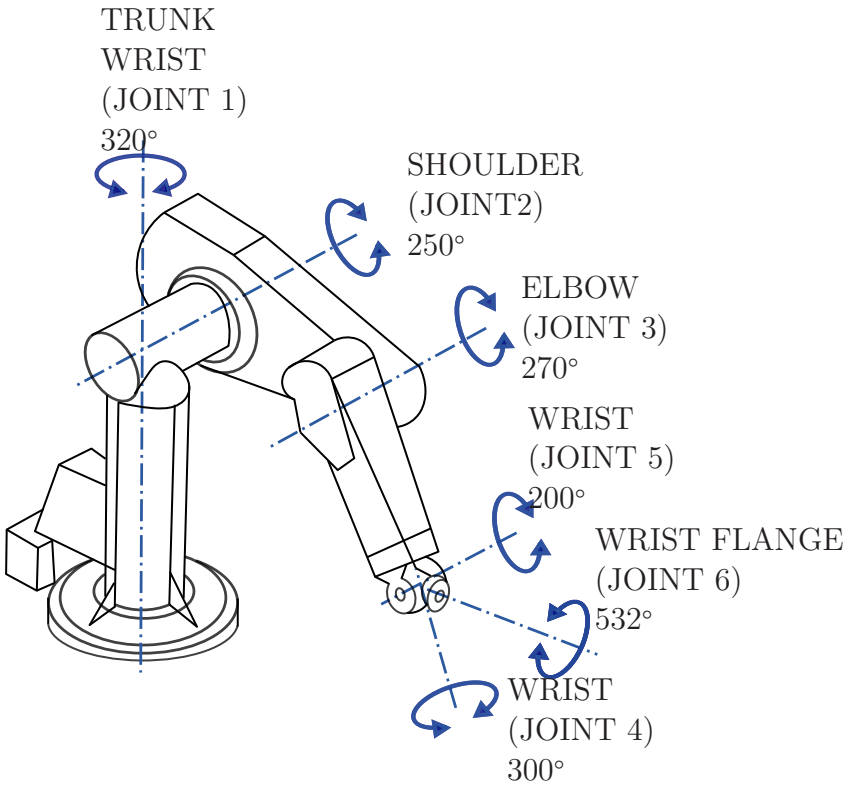


Figure 2.10: Puma-Robot(Type TVRRRT)

2.1.6 Robot Kinematics

There are six types of robot arms that are used today: [1].

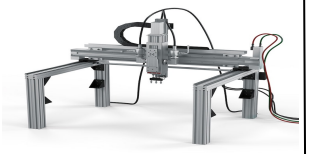
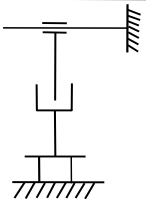
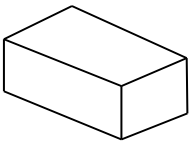
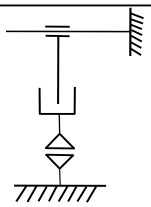
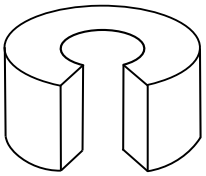
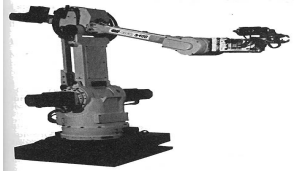
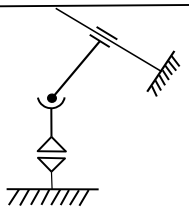
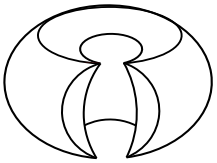
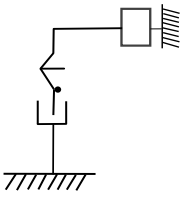
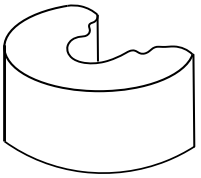
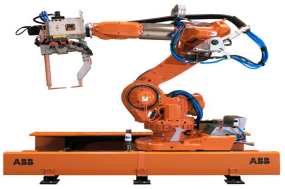
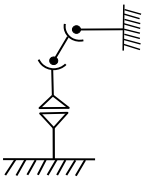
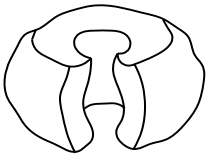
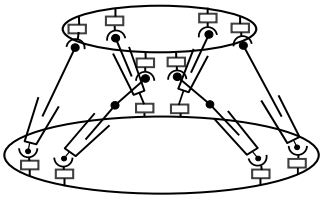
Robot	Axes	
Photo	Kenematic Structure	Workspace
 Cartesian Robot		
 Cylindrical Robot		
 Spherical Robot		
 SCARA Robot		
 Articulated Robot		
 Parallel Robot		

Figure 2.11: Robot Kinematics

[1] http://www.robot-welding.com/robots.htm

2.1.7 Actuators

An *Actuator* transmits the required energy to the axes of motion. The actuators must also compensate for the forces and moments caused by the weight of the links of the robot and the objects in the effector. So, energy is also required when the robot is not moving.

There are two main types of actuators:

- Fluid actuators (pneumatic acuators and hydraulic actuators)
- Electric actuators

Fluid actuators are linear, rotational, and are used in muscle principle

Electric actuators are linear, rotational, and are used in DC motor (brushless, brushed), AC motor, Stepper motors and Servo motor.

Typical properties of acuators are outlined below:

Pneumatic Actuators

Adjustments: Compressed air moves pistons, and there is no gear box. The advantages are it is cheap, has a simple structure, has a fast reaction time even in bad conditions, and is usable in harsh environments.

The disadvantages are it is noisy, there is no speed control when it is moving so it is difficult to control. It mostly operate only point-to-point, it has poor accuracy as air is compressible.

Pneumatic Actuators are used in small robots with fast work cycles and low forec. for example for palletizing smaller workpieces. See figure 2.12.

Figure 2.12: Pneumatic Actuators

Hydraulic Actuators

Control energy: Oil pressure pump and controllable valves; The advantages are very high forces and average speed. The disadvantages are it is noisy, additional space for hydraulics are needed, loss of oil leads to contamination, oil viscosity does not allow good reaction times and no high positioning or repetition accuracy. It is udes in large robots, for example for welding. See figure 2.13.

Figure 2.13: Hydraulic Actuators

Electric Actuators

Control energy: Electric energy.
The advantages are: it is small and required little space, it is easy to control and it is also possible to precisely travel over surfaces or curved paths because of high positioning and repeat accuracy. The disadvantages are small forces and no high speeds. It is used in small robots for tasks which require high accuracy, for example for assembling printed circuit boards. See figure 2.14.

Figure 2.14: Electric Actuators

2.1.8 Kinematics Module

The *kinematics module (control module)* of a robot allows the joints to be positioned. The basic tasks are:

- Forward calculation
- Backward calculation
- Teaching paths or path points

These methods are only briefly defined below and discussed in detail in a later chapter. In the *forward calculation*, the user directly specifies the joint coordinates (joint angle), which the kinematics module then sets. Since all joints move simultaneously from their starting position to the target position, the resulting trajectory of the hand is not known to the user unless it is calculated each time. In addition, if all joints move at maximum speed, then they will not reach their end position at the same time. However, there is the

”interpolated joint movement” in most robots. Here the speed of all joints is regulated by linear time interpolation, so that all joints reach their adjustment at the same time. This creates a more even movement.

With the *backward calculation*, the user specifies the position of the effector that the robot should approach or move through with a certain speed and acceleration. A position is indicated by the position and the orientation in the world coordinate system. The kinematics module must calculate the joint coordinates from the specified position and set this. The backward calculation is a complex task and requires high computing power, since it has to be carried out in real time (a few milliseconds). The joint positions to a point in space are also not clear.

In the case of backward calculations, we have a “point-to-point movement” or an ”interpolated movement”. In the case of point-to-point movement, the path between the starting point and the target point is usually not known to the user and is of no interest to them. In the interpolated movement, the effector is moved as precisely as possible on a selectable path from the starting point to the target point, for example in the case of linear interpolation on a straight connecting line between the starting point and the target point. The path is always implemented by calculating and controlling many intermediate points. A separate backward calculation is necessary for each of these intermediate points. Hence the above-mentioned real-time requirement for the backward calculation.

When (”Teaching”) the path points and the orientation, the user manually controls the robot arm in a sequence of target positions. Once a target position has been reached, the user causes the current joint position to be stored in the kinematics module. There is a point-to-point movement. The saved (”taught”) points can be used in robot programs and, for example, can be approached again later as often as required.

2.1.9 Joint Control

The robot represents a moving system with dimensions. When executing joint movements, these dimensions are accelerated and then decelerated again. The target point should be reached as precisely and quickly as possible. The robot should stand at the target point, i.e. all joints have reached speed 0. In the simplest case, a fixed acceleration is used until the target speed is reached, and in good time before the destination, the vehicle is braked again with a fixed deceleration. A ramp-shaped speed curve is thus obtained.

In more general cases, the user can specify intermediate points with target speeds. In any case, the kinematics module has a movement plan in which the target position and the target speed of the joints are known at all times. The actual and target positions are entered by means of angle sensors or other measuring devices in the joints.

The following requirements are placed on the regulation:

- Arm movements as fast as possible
- Movement along the specified path without swinging, in particular no overshooting in tight curves or at the target position
- Adaptation to loads in hand

- Holding the arm with the load at the target position (no drifting due to the weight of the load)

In principle, it is implemented using PID controlled systems. The following is included in the calculation of the manipulated variable:

- The difference between target and actual value (proportional part)
- The time integral over the difference between the setpoint and actual value (integral part)
- The rate of change of the difference between target and actual value, i.e. its derivation with respect to time (differential part).

2.1.10 Effectors

Effectors are the tools that are flanged to the root. Examples are grippers, drills, welding machines or, the camera, if the robot is only observing. The effectors must also be controlled, e.g. the opening and closing of a gripper up to a certain width. You also need sensors to record the status of the effectors, e.g. for the opening width or the closing force of a gripper. The effector itself is controlled by microprocessors. The control of the effector is integrated into the robot programming language. Important states of the effector are permanently reported to the robot control computer. In this way, for example, the holding of a workpiece during the robot movement can be monitored.

2.2 Sensors Classification: Proprioceptive

There are some differeneces between internal and external sensors. Internal sensors measure the state of the robot itself. External sensors record the properties of a robot's environment.

Examples of variables that measure or recognize internal states of a robot/machine are:

- Joint position
  - Potentiometer
  - Optical encoder
  - Differential transformer transducer
  - Magnetic-inductive encoder
- Joint velocity
  - Speed generator
  - Optical encoder
- Joint acceleration
  - Si-sensor
  - Piezo-electric sensor

- Orientation
  - Gyroscope
  - Geomagnetic sensor

Acquisition of external states (environment) are:

- Obstacle distance,
- Object identification,
- Object position
- Feel
  - Artificial skin
  - Sliding sensors
  - Force-torque-sensors
- Approach
  - Inductive, capacitive sensors
  - Optical sensors
  - Acoustic sensors

The query of the internal sensors is integrated into the control loops for the joints. It takes place through the microprocessors available for each Link. The control of the external sensors must be possible in the robot programming language. The control and evaluation of such sensors can require large sensor computers (image processing!). See images 2.15.

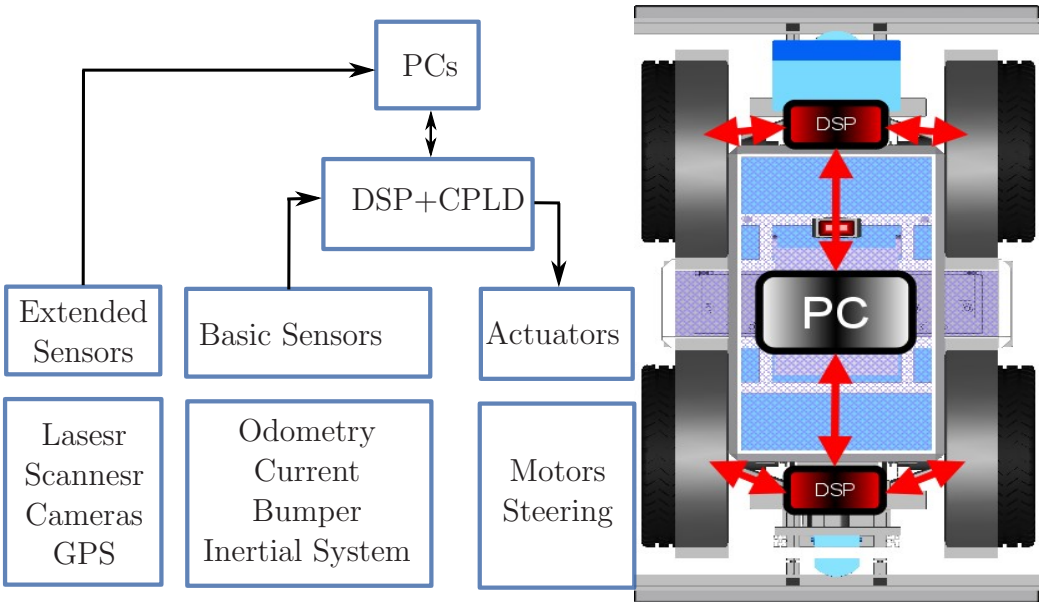


Figure 2.15: Electronics and Computing Architecture

2.3 Closed Loop Control

Closed Loop Control observes process via feedback (closed Loop), is able to react the noise.*process* is a part of the system which needs to be controlled. the value which the output should trace is called *Reference variable* w, and the difference between w and process variable r is called *Control difference* e. See figure 2.16.

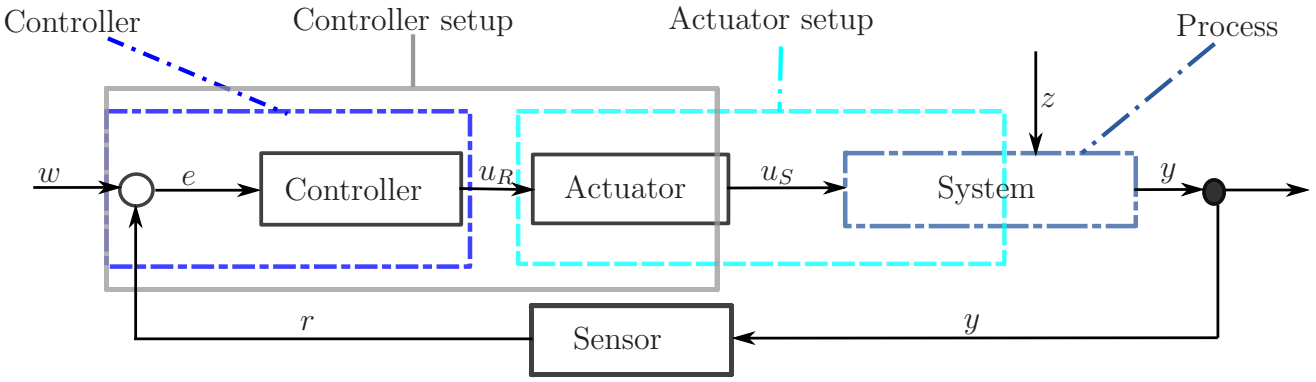


Figure 2.16: Closed Loop Control

2.3.1 Goals and Attributes of a Closed Loop Control

Selection of controller type and calculation of its parameters for:

- Steady state accuracy:control difference vanishes for $t \to \infty$
- Speed: actual value follows desired value as good (fast) as possible
- Stability: no instability of control system due to feedback of system output
- Robustness: small changes of parameters of the control plant do not changes the properties of the control system → approximations for calculation of control parameters are feasible.

See figure 2.17.

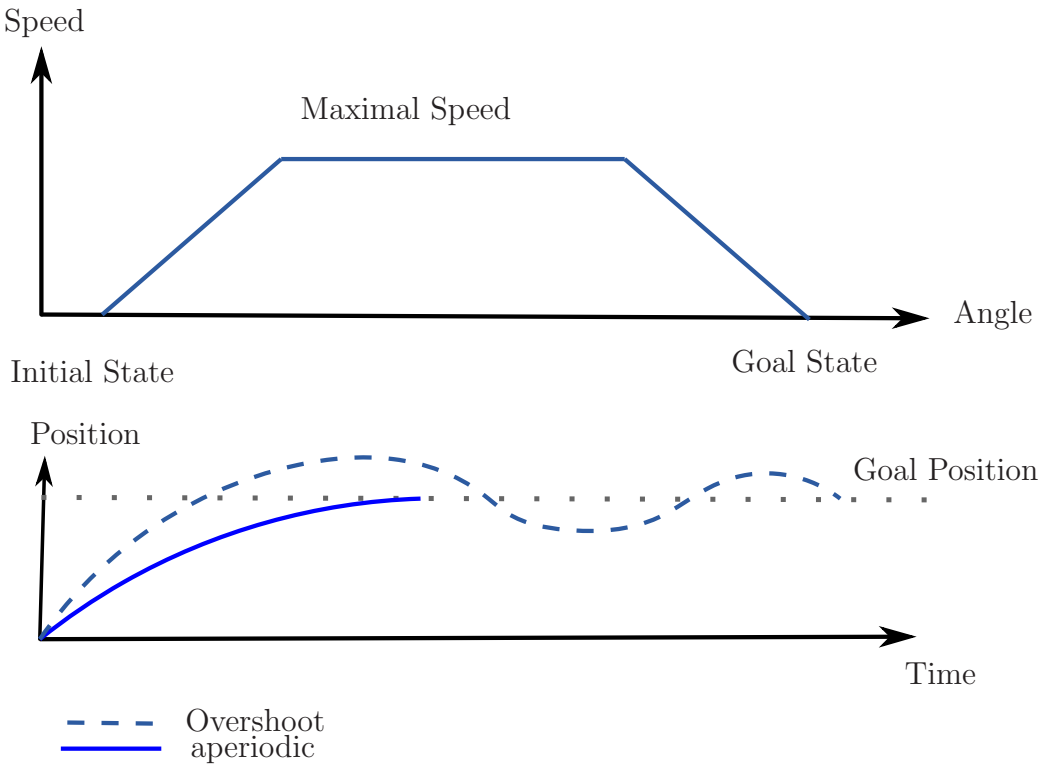


Figure 2.17: Goals and Attributes of a Closed Loop Control

2.4 Simulation

Why simulation?

- Control and perception algorithms can be developed before the robot exists
- Safe testing of algorithms
- Tests in simulation are faster (several tests in parallel on a computer cluster)
- Test can be repeated under absolutely the same conditions
- Test environments can be exchanged
- Different light and weather conditions can be generated

The problems with simulation are:

- High effort for a realistic simulation of sensor systems (real-time requirements not fulfilled)
- Physical Engine are weak in the modelling of dynamics
- High effort for the development of simulators
- Adequate interfaces to the control system must be implemented
- Still large differences between simulation and real robots in its operational environment

3. Introduction to Spatial Kinematics

3.1 Description of Objects and Object Poses in 3D Euclidean Space (E_3)

3.1.1 Coordinate Systems

Base Coordinate System(BCS)

The *base coordinate system* is a 3-dimensional coordinate system and is defined by orthogonal unit vectors $\vec{e}_{B_x}$, $\vec{e}_{B_y}$, $\vec{e}_{B_z}$. See image 3.1.

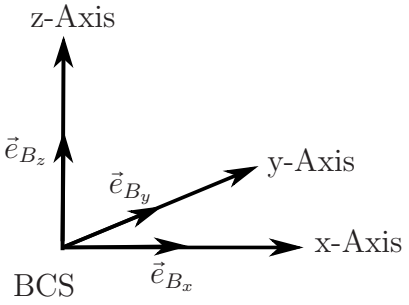


Figure 3.1: Base Coordinate System(BCS)

Object Coordinate System (OCS)

Every rigid body can be related to a local coordinate system. This local coordinate system is defined by orthogonal unit vectors $\vec{e}_{O_x}$, $\vec{e}_{O_y}$, $\vec{e}_{O_z}$. See figure 3.2.

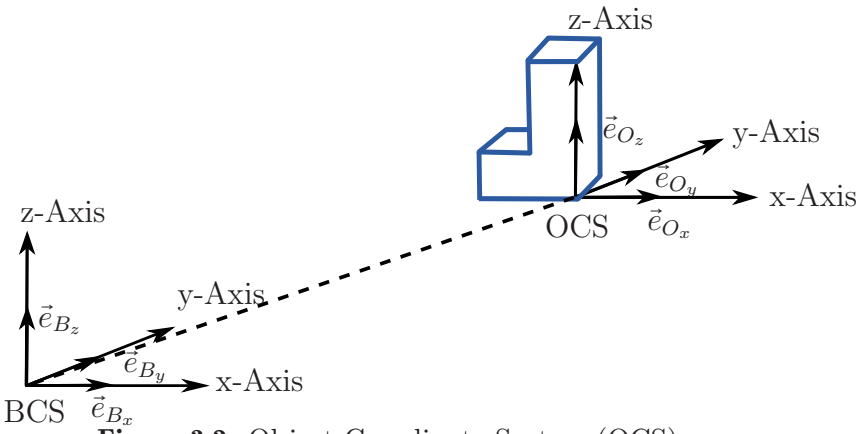


Figure 3.2: Object Coordinate System (OCS)

Orthogonal, Cartesian Coordinate Systems:

A general difference is made between counter-clockwise and clockwise coordinate systems. The direction of rotation is determined by definition:

- **Counterclockwise rotating coordinate system**
  right hand rule: thumb x , index finger y , middle finger z

  $\vec{e}_x \times \vec{e}_y = \vec{e}_z$ with cros product $\times$. See figures 3.3 and 3.4.

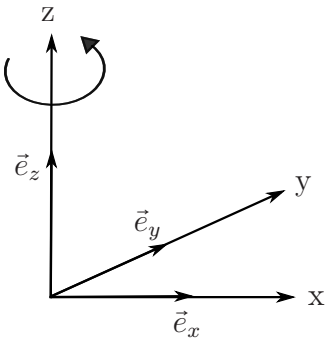


Figure 3.3: Right-rotating coordinate system

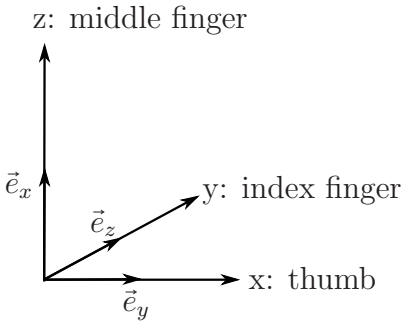


Figure 3.4: Right hand rule

- **Clockwise rotating coordinate system:**
  Left-hand-rule: thumb x , index finger y , middle finger z

  $\vec{e}_x \times \vec{e}_y = -\vec{e}_z$ with cros product $\times$. See images 3.5 and 3.6.

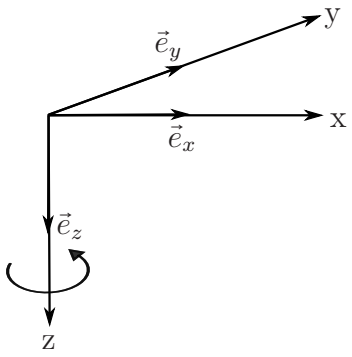


Figure 3.5: Left-rotating coordinate system

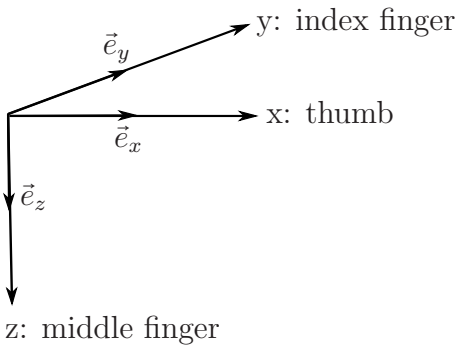


Figure 3.6: Left hand rule

In this lecture always a clockwise coordinate system is assumed, unless otherwise specified.

Object Poses in Space
Location, orientation and pose of an object define as following:

Location of an object related to a BCS:
Position vector from origin of BCS to origin of OCS

Orientation of an object in relation to a BCS:
Mapping of unit vectors of OCS to the unit vectors of BCS using rotation matrix

Pose of an object:
Position vector and rotation matrix of the OCS related to the BCS

3.1.2 Transformations

In addition to BCS, various other local coordinate systems are used for describing robotic applications:

- OCS: Object coordinate system
- ECS: Effect coordinate system (TCP-Tool center point)
- SCS: sensor coordinate system
  ⋮

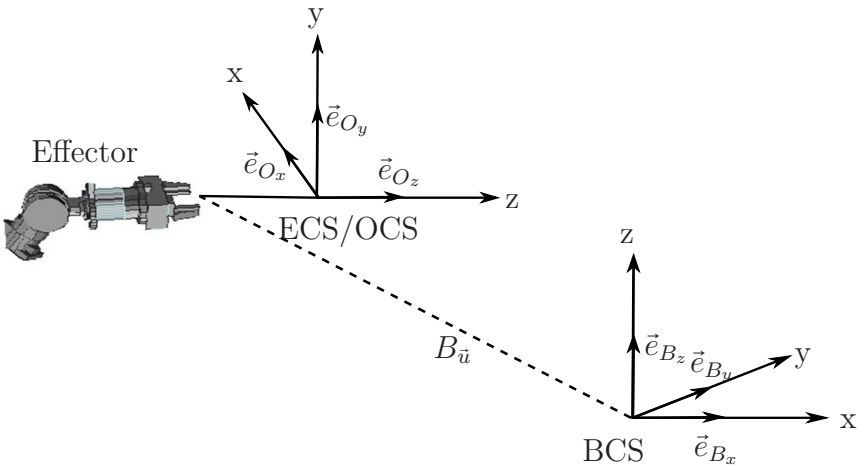


Figure 3.7: Example for Transformation

Possible Transformations

There are some possible transformations:

- Translation vector: ${}^{B}\vec{u} = {}^{B}a \cdot \vec{e}_{B_x} + {}^{B}b \cdot \vec{e}_{B_y} + {}^{B}c \cdot \vec{e}_{B_z}$
- Rotation matrix: $R = R_\alpha \cdot R_\beta \cdot R_\gamma$
- Rotation angle around coordinate axes x, y, z: α_x, β_y, γ_z

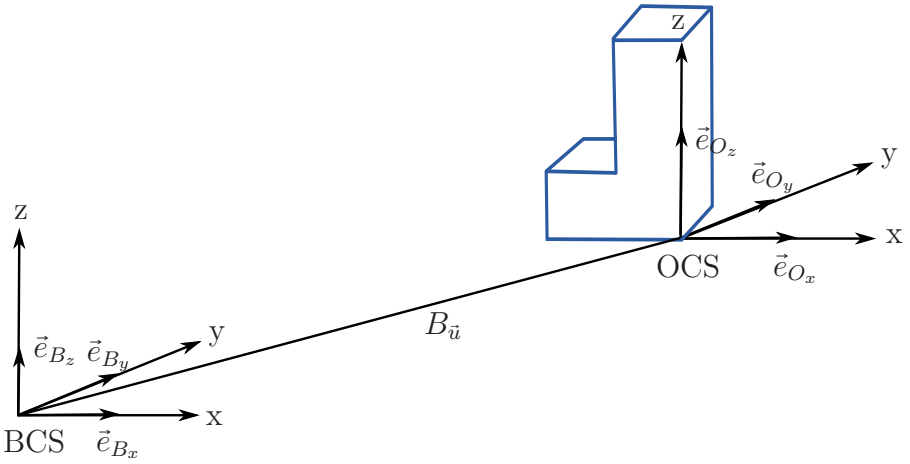


Figure 3.8: Possible Transformation

3.1.3 Rotation of a Coordinate System

Let BCS and OCS be two coordinate systems such that be rotated against each other with unit vectors $\vec{e}_{B_x}$, $\vec{e}_{B_y}$, $\vec{e}_{B_z}$. and $\vec{e}_{O_x}$, $\vec{e}_{O_y}$, $\vec{e}_{O_z}$. The position vector of a point P is given, either with regard to the OCS (${}^{O}\vec{p}$) or with regard to the BCS (${}^{B}\vec{p}$). We are looking for the position vector with respect to the other coordinate system. See figure 3.9.

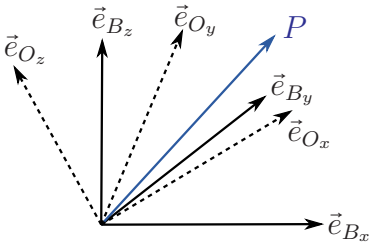


Figure 3.9: Rotation of a coordinate system

The following applies:

$$
{}^{B}\vec{p} = {}^{B}p_x \cdot \vec{e}_{B_x} + {}^{B}p_y \cdot \vec{e}_{B_y} + {}^{B}p_z \cdot \vec{e}_{B_z} \quad \text{with} \quad {}^{B}\vec{p} = \begin{bmatrix} {}^{B}p_x \\ {}^{B}p_y \\ {}^{B}p_z \end{bmatrix}
$$

$$
{}^{O}\vec{p} = {}^{O}p_x \cdot \vec{e}_{O_x} + {}^{O}p_y \cdot \vec{e}_{O_y} + {}^{O}p_z \cdot \vec{e}_{O_z} \quad \text{with} \quad {}^{O}\vec{p} = \begin{bmatrix} {}^{O}p_x \\ {}^{O}p_y \\ {}^{O}p_z \end{bmatrix}
$$

${}^{O}\vec{p}$ projection to base vectors of BCS yields to BCS coordinates:

$$
\begin{aligned}
{}^{B}p_x &= \vec{e}_{B_x} \cdot {}^{O}\vec{p} = \vec{e}_{B_x} \cdot \vec{e}_{O_x} \cdot {}^{O}p_x + \vec{e}_{B_x} \cdot \vec{e}_{O_y} \cdot {}^{O}p_y + \vec{e}_{B_x} \cdot \vec{e}_{O_z} \cdot {}^{O}p_z \\
{}^{B}p_y &= \vec{e}_{B_y} \cdot {}^{O}\vec{p} = \vec{e}_{B_y} \cdot \vec{e}_{O_x} \cdot {}^{O}p_x + \vec{e}_{B_y} \cdot \vec{e}_{O_y} \cdot {}^{O}p_y + \vec{e}_{B_y} \cdot \vec{e}_{O_z} \cdot {}^{O}p_z \\
{}^{B}p_z &= \vec{e}_{B_z} \cdot {}^{O}\vec{p} = \vec{e}_{B_z} \cdot \vec{e}_{O_x} \cdot {}^{O}p_x + \vec{e}_{B_z} \cdot \vec{e}_{O_y} \cdot {}^{O}p_y + \vec{e}_{B_z} \cdot \vec{e}_{O_z} \cdot {}^{O}p_z
\end{aligned}
$$

The conversion from BCS to OCS coordinates takes place in the same way:

$$
\begin{aligned}
{}^{O}p_x &= \vec{e}_{O_x} \cdot {}^{B}\vec{p} = \vec{e}_{O_x} \cdot \vec{e}_{B_x} \cdot {}^{B}p_x + \vec{e}_{O_x} \cdot \vec{e}_{B_y} \cdot {}^{B}p_y + \vec{e}_{O_x} \cdot \vec{e}_{B_z} \cdot {}^{B}p_z \\
{}^{O}p_y &= \vec{e}_{O_y} \cdot {}^{B}\vec{p} = \vec{e}_{O_y} \cdot \vec{e}_{B_x} \cdot {}^{B}p_x + \vec{e}_{O_y} \cdot \vec{e}_{B_y} \cdot {}^{B}p_y + \vec{e}_{O_y} \cdot \vec{e}_{B_z} \cdot {}^{B}p_z \\
{}^{O}p_z &= \vec{e}_{O_z} \cdot {}^{B}\vec{p} = \vec{e}_{O_z} \cdot \vec{e}_{B_x} \cdot {}^{B}p_x + \vec{e}_{O_z} \cdot \vec{e}_{B_y} \cdot {}^{B}p_y + \vec{e}_{O_z} \cdot \vec{e}_{B_z} \cdot {}^{B}p_z
\end{aligned}
$$

Matrix Notation

$$
{}^{B}_{O}R_1 = \begin{bmatrix} \vec{e}_{B_x} \cdot \vec{e}_{O_x} & \vec{e}_{B_x} \cdot \vec{e}_{O_y} & \vec{e}_{B_x} \cdot \vec{e}_{O_z} \\ \vec{e}_{B_y} \cdot \vec{e}_{O_x} & \vec{e}_{B_y} \cdot \vec{e}_{O_y} & \vec{e}_{B_y} \cdot \vec{e}_{O_z} \\ \vec{e}_{B_z} \cdot \vec{e}_{O_x} & \vec{e}_{B_z} \cdot \vec{e}_{O_y} & \vec{e}_{B_z} \cdot \vec{e}_{O_z} \end{bmatrix} \qquad {}^{O}\vec{p} = \begin{bmatrix} {}^{O}p_x \\ {}^{O}p_y \\ {}^{O}p_z \end{bmatrix}
$$

Scalar Product: $\vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| \cos(\alpha)$

$$
{}^{O}_{B}R_2 = \begin{bmatrix} \vec{e}_{O_x} \cdot \vec{e}_{B_x} & \vec{e}_{O_x} \cdot \vec{e}_{B_y} & \vec{e}_{O_x} \cdot \vec{e}_{B_z} \\ \vec{e}_{O_y} \cdot \vec{e}_{B_x} & \vec{e}_{O_y} \cdot \vec{e}_{B_y} & \vec{e}_{O_y} \cdot \vec{e}_{B_z} \\ \vec{e}_{O_z} \cdot \vec{e}_{B_x} & \vec{e}_{O_z} \cdot \vec{e}_{B_y} & \vec{e}_{O_z} \cdot \vec{e}_{B_x} \end{bmatrix} \qquad {}^{B}\vec{p} = \begin{bmatrix} {}^{B}p_x \\ {}^{B}p_y \\ {}^{B}p_z \end{bmatrix}
$$

$$
\begin{aligned}
{}^{B}\vec{p} &= {}^{B}_{O}R_1 \cdot {}^{O}\vec{p} \qquad & {}^{O}\vec{p} &= {}^{B}_{O}R_1^{-1} \cdot {}^{B}\vec{p} \\
{}^{O}\vec{p} &= {}^{O}_{B}R_2 \cdot {}^{B}\vec{p} \qquad & {}^{B}\vec{p} &= {}^{O}_{B}R_2^{-1} \cdot {}^{O}\vec{p}
\end{aligned}
$$

Therefore: $R_1 = R_2^{-1}$, $R_2 = R_1^{-1}$ and $R_2 = R_1^T$ are Orthogonal Matrices.

Rotation around the x-Axis with an angle α

Using scalar product: $\vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| \cos(\alpha)$:

$$
\begin{aligned}
\vec{e}_{B_x} \cdot \vec{e}_{O_x} &= 1 & \vec{e}_{B_x} \cdot \vec{e}_{O_y} &= 0 & \vec{e}_{B_x} \cdot \vec{e}_{O_z} &= 0 \\
\vec{e}_{B_y} \cdot \vec{e}_{O_x} &= 0 & \vec{e}_{B_y} \cdot \vec{e}_{O_y} &= \cos(\alpha) & \vec{e}_{B_y} \cdot \vec{e}_{O_z} &= \cos(\alpha) \\
\vec{e}_{B_z} \cdot \vec{e}_{O_x} &= 0 & \vec{e}_{B_z} \cdot \vec{e}_{O_y} &= \cos(-\alpha) & \vec{e}_{B_z} \cdot \vec{e}_{O_z} &= \cos(\alpha)
\end{aligned}
$$

In which:
$c(\alpha) = \cos(90 + \alpha) = -\sin(\alpha)$
$\sin(-\alpha) = -\sin(\alpha)$

Then:

$$
R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C\alpha & -S\alpha \\ 0 & S\alpha & C\alpha \end{bmatrix} \text{ with } C\alpha = \cos\alpha \text{ and } S\alpha = \sin\alpha
$$

After rotation with angle α around x-axis, y and z correspond to v and w respectively.

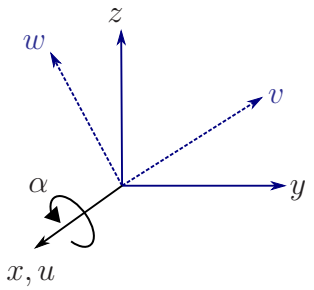


Figure 3.10: Rotation around the x-axis with an angle α

3.1.4 Rotation Matrix

Geometric Derivation

- Frame $OX_0Y_0Z_0$ resulted from frame $BX_BY_BZ_B$ through rotating around axis z with angle α.
- Calculation of the coordinates of point $P_0 = ({}^{O}x,\ {}^{O}y,\ {}^{O}z)^T$ in coordinate system B is shown in the figure 3.11.

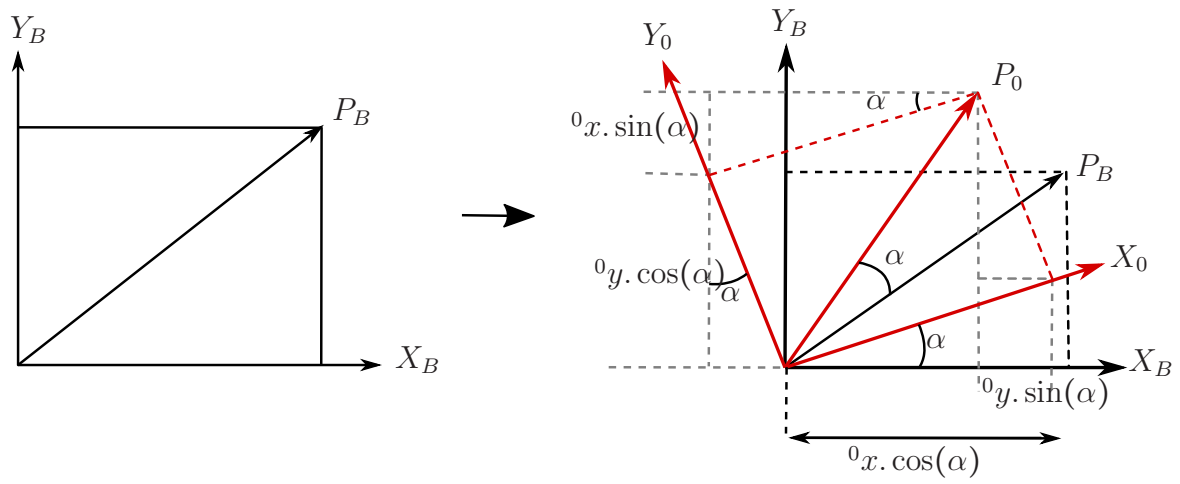


Figure 3.11: Rotation: Geometric Derivation

Rotation around z-Axis with angle α

For a rotation by an angle α around z-axis:

- A point P_0 with the coordinates $({}^Ox,\ {}^Oy,\ {}^Oz)^T$ in the frame OCS, in the basic frame BCS receives the coordinates :

$$\begin{aligned} {}^Bx &= {}^Ox\cos(\alpha) - {}^Oy\sin(\alpha) \\ {}^By &= {}^Ox\sin(\alpha) + {}^Oy\cos(\alpha) \\ {}^Bz &= {}^Oz \end{aligned}$$

- The z-coordinate remains fixed because the axis of rotation is z-axis.

Notation in Matrix Form:

$${}^B\vec{p} = \begin{bmatrix} {}^Bx \\ {}^By \\ {}^Bz \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} {}^Ox \\ {}^Oy \\ {}^Oz \end{bmatrix} = {}^B_OR_z(\alpha) \cdot {}^O\vec{p}$$

Rotation matrix $R_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$ describes this rotation around z axis. The rotation around the x or y axis can also be described in the same way:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \qquad R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

Properties:

Properies of a rotation matrix R:

- R is an affine mapping of $\mathbb{R}_3 \to \mathbb{R}_3$
- R is a real matrix
- R is quadratic
- R is invertible
- R is orthogonal (row or column vectors orthogonal to each other)

Let R be a rotation matrix. Then:

- Rank $R_g(R) = 3$
- $R^T = R^{-1}$
- $R \cdot R^{-1} = R^{-1} \cdot R = \mathcal{I}$
- $det(R) = 1$

3.1.5 Several Elementary Rotations

Basic Rotations

From now on, ${}^B\vec{e}_x = {}^0\vec{e}_x = x_0$ applies. Analog for ${}^B\vec{e}_y = {}^0\vec{e}_y = y_0$ and ${}^B\vec{e}_z = {}^0\vec{e}_z = z_0$.

Let OCS result be based on 2 rotations from BCS.

$$R_y(-90°) = \begin{bmatrix} \cos(-\frac{\pi}{2}) & 0 & \sin(-\frac{\pi}{2}) \\ 0 & 1 & 0 \\ -\sin(-\frac{\pi}{2}) & 0 & \cos(-\frac{\pi}{2}) \end{bmatrix} \qquad R_{x'}(180°) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\pi) & -\sin(\pi) \\ 0 & \sin(\pi) & \cos(\pi) \end{bmatrix}$$

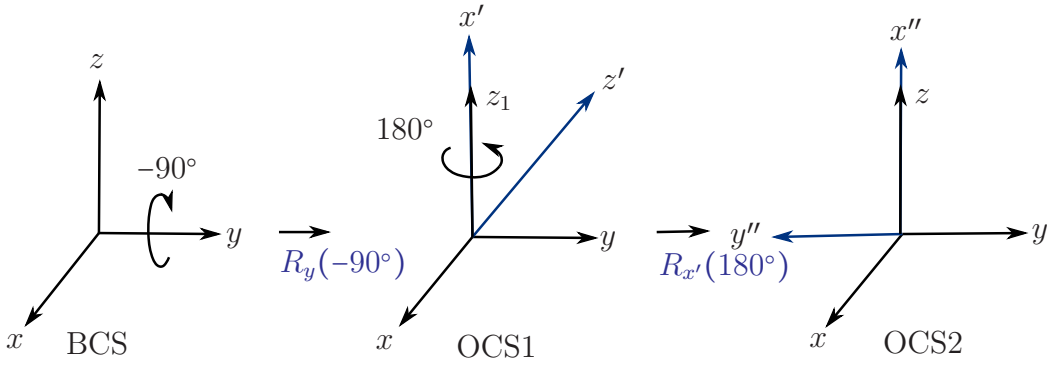


Figure 3.12: Several Elementary Rotation

Vector Coordinates due to a new Frame

Calculation of ${}^B\vec{u}$ from ${}^{O}2\vec{u}$ (also see figure 3.12):

$${}^B\vec{u} = {}^B_{O1}R_y(-90°) \cdot {}^{01}\vec{u} = {}^B_{O1}R_y(-90°) \cdot {}^{O1}_{O2}R_{x'}(180°) \cdot {}^{O2}\vec{u} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} {}^{O2}\vec{u}_1 \\ {}^{O2}\vec{u}_2 \\ {}^{O2}\vec{u}_3 \end{bmatrix} = \begin{bmatrix} {}^{O2}\vec{u}_3 \\ -{}^{O2}\vec{u}_2 \\ {}^{O2}\vec{u}_1 \end{bmatrix}$$

Example: Transformation on rotation of the unit vectors

$$
\begin{aligned}
{}^{B}\vec{e}_{O2_x} &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\
{}^{B}\vec{e}_{O2_y} &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \\
{}^{B}\vec{e}_{O2_z} &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\end{aligned}
$$

Interpretation of Several Rotations

Pre-multiplication : $R = (R_n(R_{n-1} \dots (R_2 R_1) \dots))$
Interpretation: Rotation of the current coordinate system around fixed axes of the original coordinate system.

Post-multiplication : $R = ((\dots (R_1 R_2) \dots R_{n-1}) R_n)$
Interpretation: rotation around an axis of the current coordinate system.

3.2 Different Notations for Rotations

In rocotics, there are many different notations for rotation axes and their order. They all are equivalent, but the have different benefits. in the following we mention some of them and their benefits:

- **Rotation around unique axis**
  It is a Trade-off between others.
- **Euler angels**
  Follows chained Joint-Setup.
- **Roll-Pitch-Yaw**
  it is easy to interpret by humans.
- **Quaternions**
  It is computationally fast.
- **Exponential coordinates**
  It is more similar to its kinematic.

3.2.1 Rotation Around unique Axis

In this model, instead of rotation with BCS-axis, rotate around unique Axis.

Our goal is to find $\vec{g} \in \mathbb{R}^3$, $\|\vec{g}\| = 1$, $\theta \in [0, 2\pi)$, such that:
For BCS $x, y, z \in \mathbb{R}^3$, and arbitrary α, β and $\gamma \in [0, 2\pi)$, the following holds:

$$R_{\vec{g}}(\theta) = R_z(\gamma).R_y(\beta).R_x(\alpha)$$

Rodrigues Formula:

Given a transformation matrix

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

we can obtain $\hat{g} = \frac{1}{2S\theta}(R - R^T)$ and $\theta = \cos^{-1}(\frac{tr(R)-1}{2}) \in [0, \pi]$.
Hence,

$$R = R_{\vec{g}}(\theta) = \begin{bmatrix} g_1^2\eta\theta + C\theta & g_1g_2\eta\theta - g_3S\theta & g_1g_3\eta\theta + g_2S\theta \\ g_1g_2\eta\theta + g_3S\theta & g_2^2\eta\theta + C\theta & g_2g_3\eta\theta - g_1S\theta \\ g_1g_3\eta\theta - g_2S\theta & g_2g_3\eta\theta + g_1S\theta & g_3^2\eta\theta + C\theta \end{bmatrix}$$

with:

$$S\theta = \sin\theta, \quad C\theta = cos\theta, \quad \eta\theta = (1 - \cos\theta), \quad \vec{g} = (g_1, g_2, g_3)^T = \frac{1}{2S\theta}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}$$

$\hat{g}$ is the skew-symmetric matrix corresponding to the vector $\vec{g}$:

$$\hat{g} = \begin{bmatrix} 0 & -g_3 & g_2 \\ g_3 & 0 & -g_1 \\ -g_2 & g_1 & 0 \end{bmatrix}$$

the matrix R can be decomposed to:

$$R = R_{\vec{g}}(\theta) = \cos(\theta)\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + (1 - \cos(\theta))\begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix}\begin{bmatrix} g_1 & g_2 & g_3 \end{bmatrix} + \sin(\theta)\begin{bmatrix} 0 & -g_3 & g_2 \\ g_3 & 0 & -g_1 \\ -g_2 & g_1 & 0 \end{bmatrix}$$

To be equal to:

$$R = R_{\vec{g}}(\theta) = C\theta\mathcal{I}_3 + (1 - \cos(\theta))\vec{g}\vec{g}^T + S\theta\hat{g}$$

Euler Theorem:

Every rotation matrix R_3 is equivalent to a rotation around a fixed axis

$$\vec{g} \in \mathbb{R}^3, \|\vec{g}\| = 1$$

and a rotation angle $\theta \in [0, 2\pi)$.

Proof:

Let

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

and

$$R_{\vec{g}}(\theta) = \begin{bmatrix} g_1^2\eta\theta + C\theta & g_1g_2\eta\theta - S\theta & g_1g_3\eta\theta + g_2S\theta \\ g_1g_2\eta\theta + g_3S\theta & g_2^2\eta\theta + C\theta & g_2g_3\eta\theta - g_1S\theta \\ g_1g_3\eta\theta - g_2S\theta & g_2g_3\eta\theta + g_1S\theta & g_3^2\eta\theta + C\theta \end{bmatrix}$$

our aim it to prove that $R = R_{\vec{g}}(\theta)$.

The following applies to the trace of the matrices:

$$trR = r_{11} + r_{22} + r_{33} \stackrel{!}{=} 3\cos\theta + (1 - \cos\theta)\sigma g_i^2 = 1 + 2\cos\theta$$
$$\Rightarrow \theta = \cos^{-1}\left(\frac{trR - 1}{2}\right) \in [0, \pi]$$

This equation can be solved for θ, because the eigenvalues λ_i of R have amount 1 and therefore:

$$-1 \leq trR = \sum \lambda_i \leq 3$$

To determine the axis of rotation, we use the remaining matrix entries:

$$\left.\begin{aligned} r_{32} - r_{23} &\stackrel{!}{=} 2g_1S\theta \\ r_{13} - r_{31} &\stackrel{!}{=} 2g_2S\theta \\ r_{21} - r_{12} &\stackrel{!}{=} 2g_3S\theta \end{aligned}\right\} \stackrel{\theta \neq 0}{\Longrightarrow} \vec{g} = \frac{1}{2S\theta}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}$$

If $R = \mathcal{I}_3$, then $trR = 3$, and therefore $\theta = 0$. In this case, $\vec{g}$ could be any vector, then $R_{\vec{g}}(0) = \mathcal{I}_3$

3.2.1.1 Representation of Orientation

Roll-Pitch-Yaw:

- xyz-System
  Used in aerospace, in mobile robotics.

Euler-angles (Examples):

- $zx'z''$-system:
  usual mathematical definition.

- $zy'x''$-system:
  is used in programming of numerically controlled manipulators, for example IRDATA (wrist TRR).

- $zy'z''$-system:
  is used in programming language VAL, and PUMA-robot

3.2.1.2 Computation of Roll-Pitch-Yaw-Angles

Multiplying from the right with $R_x(\alpha)^{-1}$ holds:

$$R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot R_x(\alpha)^{-1} = R_s \cdot R_x(\alpha)^{-1}$$

is simplified to:

$$R_z(\gamma) \cdot R_y(\beta) = R_s \cdot R_x(\alpha)^T$$

See exercise!

Example:
Example calculation:
Calculation of Homogenuous Transformation Matrices gives the following equations for Roll-Pitch-Yaw:

$$\begin{bmatrix} C\beta & 0 & S\beta \\ S\beta \cdot S\alpha & C\alpha & -S\alpha \cdot C\beta \\ -C\alpha \cdot S\beta & S\alpha & C\alpha \cdot C\beta \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ C\gamma & S\gamma & 0 \\ -S\gamma & C\gamma & 0 \end{bmatrix}$$

Or equivalently:

row.column

$$\begin{aligned}
1.1: \quad & C\beta = 0 \\
1.2: \quad & 0 = 0 \\
1.3: \quad & S\beta = 1 \\
2.1: \quad & S\beta \cdot S\alpha = C\gamma \\
2.2: \quad & C\alpha = S\gamma \\
2.3: \quad & -S\alpha \cdot C\beta = 0 \\
3.1: \quad & -C\alpha \cdot S\beta = -S\gamma \\
3.2: \quad & S\alpha = C\gamma \\
3.3: \quad & C\alpha \cdot C\beta = 0
\end{aligned}$$

Angle β: From (1.1), (1.3) it follows that:

$$\beta = 90^\circ$$

Angle α and γ: From (2.2) and (3.2) it follows that:

$$\gamma = 90° - \alpha$$

With $\beta = 90°$ you can simplify (2.1), (2.3), (3.1), (3.3) to (2.2) and (3.2). There is no equation for α or γ, such that α (or γ) can be arbitrarily chosen.

Choose $\alpha = 0°$, than there are solutions (0°, 90°, 90°)

3.3 Axes of Rotation in Robotics

Rotation axes are usually BCS. Convention of rotation axes and their order are usually in Euler-angles and Roll-Pitch-Yaw.

3.3.1 Euler-Angles

Euler-Angles (zxz)

- Rotation α around die z-axis of BCS $R_z(\alpha)$
- Rotation β around the new x-axis x' $R_{x'}(\beta)$
- rotation γ around the z-axis z'' $R_{z''}(\gamma)$

$$\begin{aligned} R_s &= R_z(\alpha) \cdot R_{x'}(\beta) \cdot R_{z''}(\gamma) \\ R_s(\alpha,\beta,\gamma) &= \begin{bmatrix} C\alpha C\gamma - C\beta S\gamma S\alpha & -C\alpha S\gamma - C\beta C\gamma S\alpha & S\alpha S\beta \\ S\alpha C\gamma + C\beta S\gamma C\alpha & C\alpha C\beta C\gamma - S\alpha S\gamma & -C\alpha S\beta \\ S\gamma S\beta & C\gamma S\beta & C\beta \end{bmatrix} \end{aligned}$$

Euler-angles (zyz)

- Rotation α around the z-axis of BCS $R_z(\alpha)$
- Rotation β around the new y-axis y' $R_{y'}(\beta)$
- rotation γ around the new z-axis z'' $R_{z''}(\gamma)$

$$\begin{aligned} R_s(\alpha,\beta,\gamma) &= R_z(\alpha) \cdot R_{y'}(\beta) \cdot R_{z''}(\gamma) \\ R_s(\alpha,\beta,\gamma) &= \begin{bmatrix} C\alpha C\beta C\gamma - S\alpha S\gamma & -C\alpha C\beta S\gamma - S\alpha C\gamma & C\alpha S\beta \\ S\alpha C\beta C\gamma + C\alpha S\gamma & -S\alpha C\beta S\gamma + C\alpha C\gamma & S\alpha S\beta \\ -S\beta C\gamma & S\beta S\gamma & C\beta \end{bmatrix} \end{aligned}$$

Rotation around changed axes $R_{z,\alpha}$, $R_{y',\beta}$, $R_{z'',\gamma}$

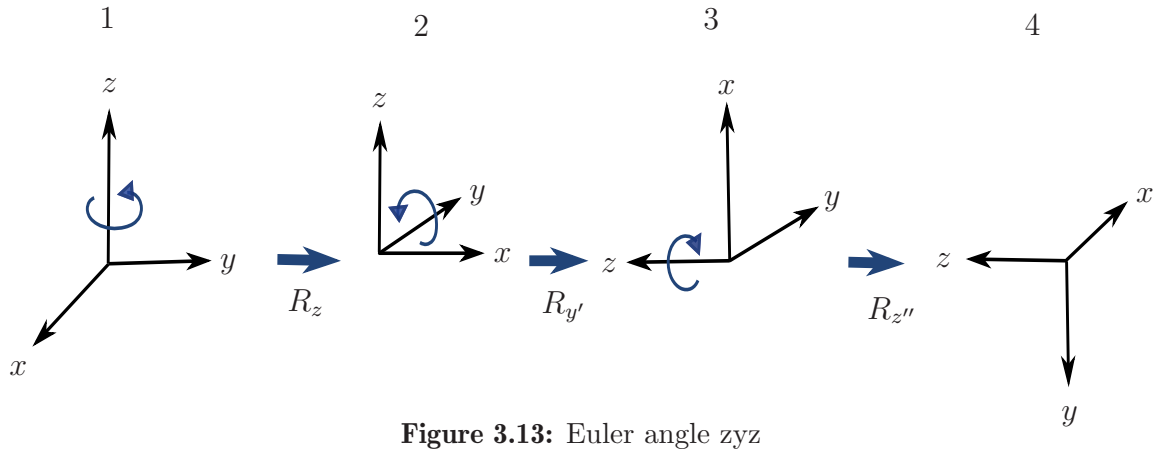


Figure 3.13: Euler angle zyz

Example

Euler-angles: $R_s = R_z(0°) \cdot R_{y'}(90°) \cdot R_{z''}(90°)$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

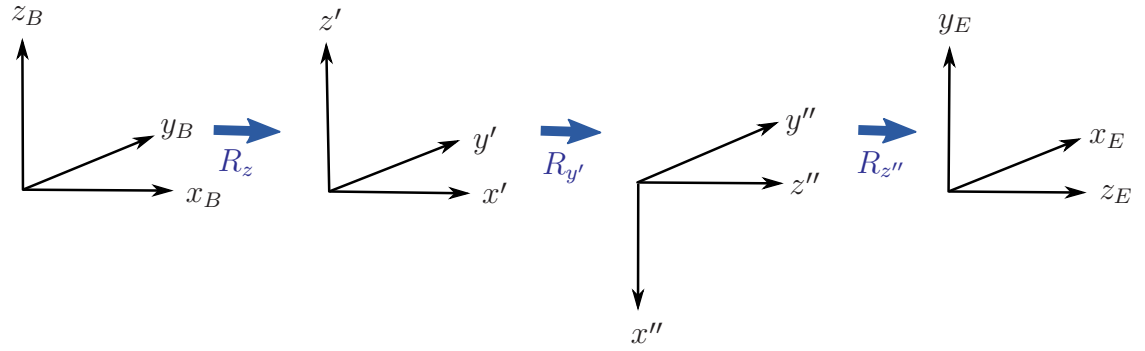


Figure 3.14: Euler-angle: Example

3.3.1.1 Derivation

Theorem:
If two right-handed Cartesian coordinate systems $R = \{U, e_1, e_2, e_3\}$ and $\overline{R} = \{U, \bar{e}_1, \bar{e}_2, \bar{e}_3\}$ with a common origin exist, then there exists an orthogonal matrix A that maps R to $\overline{R}$.

Proof:
All orientations can be described with Euler angles.

Euler Angles Derivation

1. Rotation around e_3 with the position angle α, so that e_1 is mapped onto f_1.

- f_1 is constructed by a positive rotation with α with $0 \leq \alpha < \pi$, lies on P.
- R transforms into to $R = \{U, f_1, f_2, f_3 = e_3\}$
- The following applies: $A_1 = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} R_1 = RA_1$
- and also: $f_1 \perp e_3$ and $f_1 \perp \bar{e}_3$. See figure 3.15

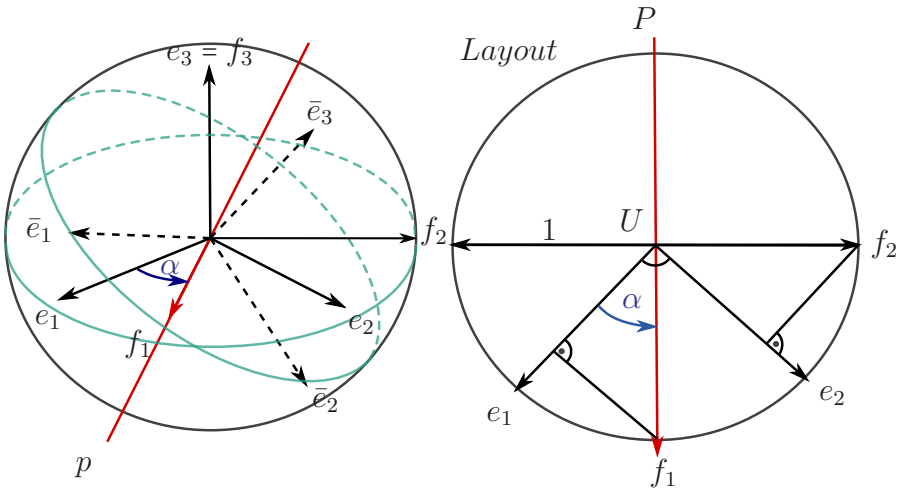


Figure 3.15: Euler-Angles: Coordinate system

Plane E_1 (spanned by e_1 and e_2) intersects E_2 (spanned by $\bar{e}_1$ and $\bar{e}_2$) in line P

2. Rotate R_1 around axis f_1 with angle β so that $e_3 = f_3$ falls together with $\bar{e}_3$. See figure 3.16.

- R transforms into $R_2 = \{U; f_1 = h_1, h_2, h_3 = \bar{e}_3\}$
- The following applies: $A_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & -\sin(\beta) \\ 0 & \sin(\beta) & \cos(\beta) \end{bmatrix} R_2 = R_1 A_2$
- f_2 is mapped onto h_2.
- h_2 lies in the plane spanned by $\bar{e}_1$ and $\bar{e}_2$.

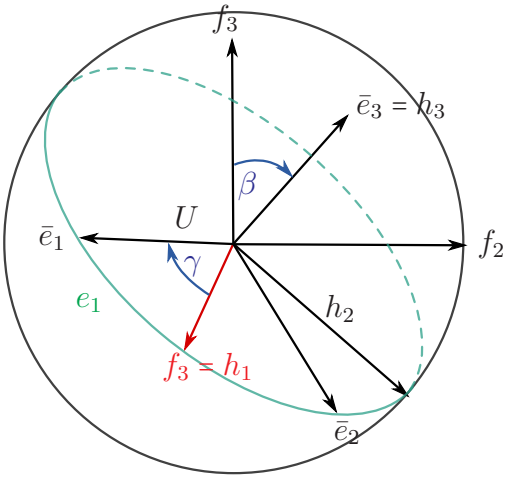


Figure 3.16: Euler-Angles: Coordinate system

3. Rotate R_3 by the angle γ, so that R_2 falls together with $\overline{R}$.

   - The following holds:
   $$A_3 = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} R_3 = R_2 A_3$$

Put together, the following applies: $\overline{R} = (R_1 A_2) A_3 = (R A_1)(A_2 A_3)$

Let $A = A_1 A_2 A_3$, then $\overline{R} = RA$ with

$$A = \begin{bmatrix} C\alpha C\gamma - S\alpha C\beta S\gamma & -C\alpha S\gamma - S\alpha C\beta C\gamma & S\alpha S\beta \\ S\alpha C\gamma - C\alpha C\beta S\gamma & -S\alpha S\gamma + C\alpha C\beta C\gamma & -C\alpha S\beta \\ S\beta S\gamma & S\beta C\gamma & C\gamma \end{bmatrix}$$

Through equating coefficients it is possible to uniquely identify α, β, γ with $0 \leq \alpha < \pi$:

$$\begin{aligned} a_{13} &= \sin(\alpha)\sin(\beta) \\ a_{23} &= -\sin(\beta)\cos(\alpha) \\ a_{33} &= \cos(\beta) \\ a_{31} &= \sin(\beta)\sin(\gamma) \\ a_{32} &= \sin(\beta)\cos(\gamma) \end{aligned}$$

Rotation Axis and Angle of Rotation

Every orthogonal (3×3) matrix $A = (a_{ik})$ with $\det(A) = 1$ describes a rotation around an axis g by a rotation angle α. The following applies to the angle of rotation:

$$\cos(\alpha) = \frac{1}{2}(a_{11} + a_{22} + a_{33} - 1)$$

And, if $\alpha \neq 0°$ and $\alpha \neq 180°$, the axis of rotation g is determined by

$$\begin{aligned} g_1 &= (a_{32} - a_{23}) \\ g_2 &= (a_{13} - a_{31}) \\ g_3 &= (a_{21} - a_{12}) \end{aligned}$$

Rodriques´ Theorem:

$$\vec{q}' = \vec{q}\cos(\alpha) + \sin(\alpha)(\vec{k} \times \vec{q}) + (1 - \cos(\alpha))(\vec{k} \cdot \vec{q})\vec{k}$$

Rotation of the vector $\vec{q}$ around the axis, which is described by the vector $\vec{k}$, with the angle α.

3.3.2 Roll-Pitch-Yaw

- Roll γ around x-Axis of BCS $R_x(\gamma)$
- Pitch β around y-axis of BCS $R_y(\beta)$
- Yaw α around z-Axis of BCS $R_z(\alpha)$

$$R_s(\alpha, \beta, \gamma) = R_z(\alpha)R_y(\beta)R_x(\gamma)$$

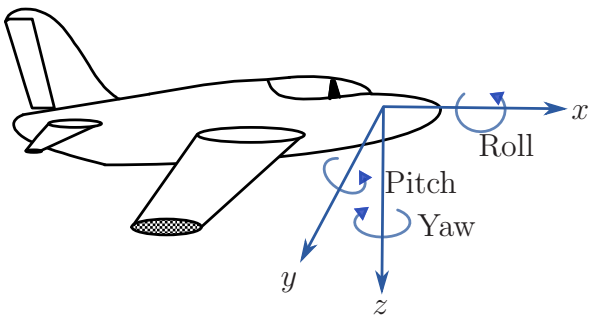


Figure 3.17: Roll-Pitch-Yaw

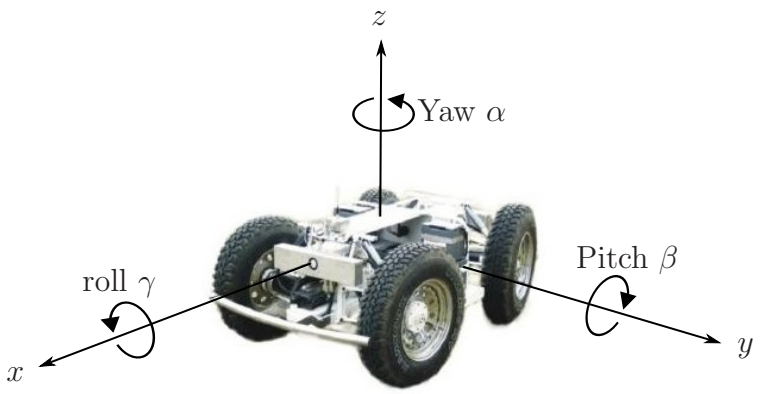


Figure 3.18: Roll-Pitch-Yaw in Robotics

Roll-Pitch-Yaw-Rotation matrix

Rotation matrix R_s related to the basic coordinate system

$$R_s = \begin{bmatrix} C\alpha C\beta & C\alpha S\beta S\gamma - S\alpha C\gamma & C\alpha S\beta C\gamma + S\alpha S\gamma \\ S\alpha C\beta & S\alpha S\beta S\gamma + C\alpha C\gamma & S\alpha S\beta C\gamma + C\alpha S\gamma \\ -S\beta & C\beta S\gamma & C\beta C\gamma \end{bmatrix}$$

Important:
Interpretation: rotation around unchanged axes! Write down from right to left!

Example:

Roll-Pitch-Yaw:

$$\begin{aligned} R_s &= R_z(90°) \cdot R_y(90°) \cdot R_z(0°) \\ &= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix} \end{aligned}$$

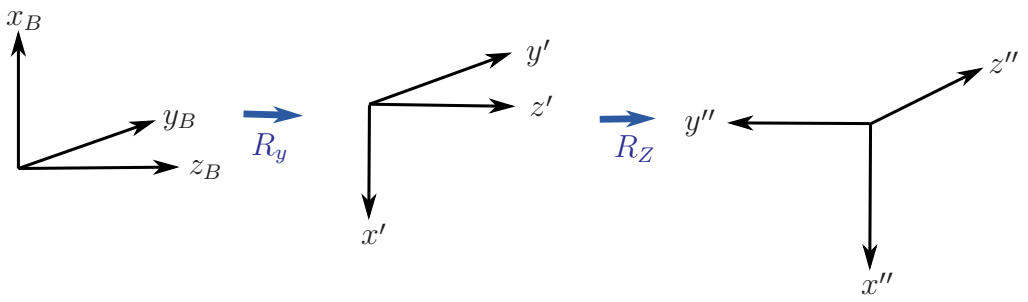


Figure 3.19: Roll-Pitch-Yaw Example

3.4 Object Pose in a 3D Euclidian Space E_3

3.4.1 Orientation of a Rigid Body

Every orientation of a rigid body in E_3 can be achieved by 3 rotations around the coordinate axes. Each rotation around an axis can be represented by a 3×3 rotation matrix. The individual rotations are linked by multiplying the corresponding matrices. The matrix multiplication is not commutative, but associative.
Interpretation: $R_1 \cdot R_2 \cdot \cdots \cdot R_n$ from left to right.
The rotation of R_i refers to the coordinate system defined by the matrix product on the left. R_1 rotates BCS.

Euler angle: $R_s = R_z(\alpha) \cdot R_{y'}(\beta) \cdot R_{z''}(\gamma)$

6-dimensional description vector

The pose of an object in E_3 can be described by a 6-tuple $(x, y, z, \alpha, \beta, \gamma) \in \mathbb{R}^6$

$$^B\vec{u} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + {}^B_O R(\alpha, \beta, \gamma)\, {}^O\vec{u}$$

- x, y, z: Coordinates of the origin of OCS relative to the BCS (describing position).
- α, β, γ: Angle of rotation, corresponding to the axes of rotation (describing orientation).
- Position vector and rotation matrix are intuitive and mainly used for the pose description of objects and end effectors.
- Cons: Vector and matrix operators are separate.

3.4.2 Homogeneous Coordinate Transformation Matrix

By means of matrix operations with homogeneous matrices, the translations common in robotics can be described uniformly. Translations and rotations operators can be replaced through *homogeneous matrices*. It means cartesian coordinates can transfer to to homogeneous coordinates.

Let P be a point with cartesian coordinates (p_x, p_y, p_z) and $s \in \mathbb{R}$, then,

$$P' = (sp_x, sp_y, sp_z, s) \in \mathbb{R}^4$$

is the representation of P in homogeneous coordinates:

For each P, there is an infinite number of homogeneous points P'.

Homogeneous Coordinates

Conversely, the cartesian coordinates of the homogeneous point $P(x, y, z, s)$ are equal to $(\frac{x}{s}, \frac{y}{s}, \frac{z}{s})$.

In general, the use of homogeneous coordinates in E_3 allows the development of 4×4 transformation matrices that contain the rotation, translation, scaling and perspective transformations. In the case of homogeneous coordinates, transformations of n-dimensional physical vectors into an n+1-dimensional space are carried out. In robotics, $s = 1$.

Object Pose in Homogeneous Coordinates

Let us scaling factor s and perspective transformation P which Here is $(0, 0, 0)^T$. Also let us define translation vector $u = (u_x, u_y, u_z)$ with the position is the origin of OCS relative to BCS.
With rotations $R = (n_x, n_y, n_z)$, (o_x, o_y, o_z) and (a_x, a_y, a_z) and the orientation of 3 unit vectors in x, y and z direction of the OCS. with all these assumption:

$$A = \begin{bmatrix} R & \vec{u} \\ \vec{P} & s \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & u_x \\ R_{21} & R_{22} & R_{23} & u_y \\ R_{31} & R_{32} & R_{33} & u_z \\ P_1 & P_2 & P_3 & s \end{bmatrix} \Rightarrow \begin{bmatrix} n_x & o_x & a_x & u_x \\ n_y & o_y & a_y & u_y \\ n_z & o_z & a_z & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Homogeneous Translation Matrix T

Let $\vec{p}$ and $\vec{p}'$ be position vectors in homogeneous coordinates, and Let $\vec{a}$ be a homogeneous translation vector $\vec{a} = (a_x, a_y, a_z, 1)^T$. Then, a cartesian translation $p = a + p'$ can be represented by homogeneous translation matrix T:

$$p = T(a_x, a_y, a_z)p' \quad \text{with T=} \quad \begin{bmatrix} 1 & 0 & 0 & a_x \\ 0 & 1 & 0 & a_y \\ 0 & 0 & 1 & a_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Rightarrow T(a_x, a_y, a_z)^{-1} \quad = \quad \begin{bmatrix} 1 & 0 & 0 & -a_x \\ 0 & 1 & 0 & -a_y \\ 0 & 0 & 1 & -a_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Local (Anisotropic) Scaling

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} ax \\ by \\ cz \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} ax \\ by \\ cz \end{bmatrix}$$

Global (Isotropic) Scaling

(s>1 down scaling, s<1 up scaling)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ s \end{bmatrix} \rightarrow \begin{bmatrix} \frac{x}{s} \\ \frac{y}{s} \\ \frac{z}{s} \end{bmatrix}$$

Homogeneous Rotation Matrices

Rotation matrices $R_z(\alpha)$, $R_x(\alpha)$ and $R_y(\alpha)$ describe these rotations::

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverse Homogeneous Rotation matrix R^{-1}

Let R_3 be a 3×3 rotation matrix, then:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Rightarrow R^{-1} = \begin{bmatrix} r_{11} & r_{21} & r_{31} & 0 \\ r_{12} & r_{22} & r_{32} & 0 \\ r_{13} & r_{23} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and,

$$A = T(a_x, a_y, a_z)R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & a_x \\ r_{21} & r_{22} & r_{23} & a_y \\ r_{31} & r_{32} & r_{33} & a_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

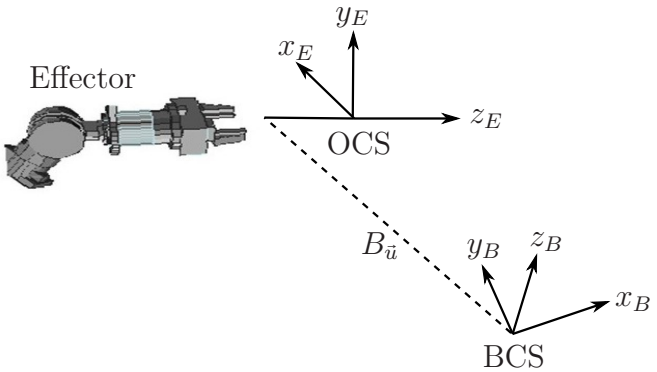
$$\Rightarrow A^{-1} = R^{-1}T^{-1} = \begin{bmatrix} r_{11} & r_{21} & r_{31} & 0 \\ r_{12} & r_{22} & r_{32} & 0 \\ r_{13} & r_{23} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -a_x \\ 0 & 1 & 0 & -a_y \\ 0 & 0 & 1 & -a_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} r_{11} & r_{21} & r_{31} & -r_{11}a_x - r_{21}a_y - r_{31}a_z \\ r_{12} & r_{22} & r_{32} & -r_{12}a_x - r_{22}a_y - r_{32}a_z \\ r_{13} & r_{23} & r_{33} & -r_{13}a_x - r_{23}a_y - r_{33}a_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Properties of Homogeneous Matrices

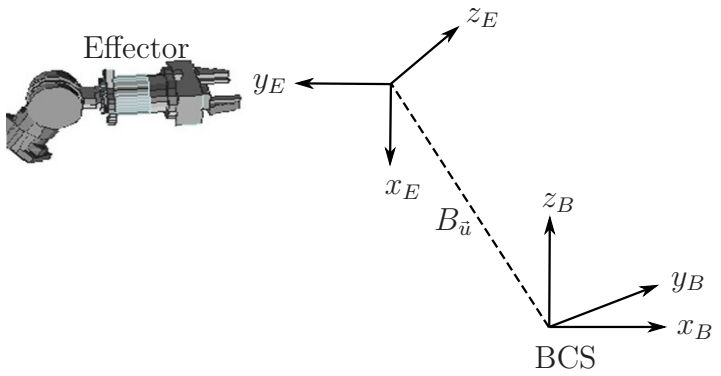
In a homogeneous 4×4 matrix there are 12 $(\vec{n}, \vec{o}, \vec{a}, \vec{u})$ nontrivial parameters, but only 6 $(x, y, z, \alpha, \beta, \gamma)$ are necessary. Also homogeneous matrices are redundant, because they are orthogonal.

Calculation of Homogeneous Transformation Matrices



$$({}^E x,\ {}^E y,\ {}^E z, R) = \left[\begin{array}{ccc|c} 0 & 0 & 1 & {}^B u_x \\ 1 & 0 & 0 & {}^B u_y \\ 0 & 1 & 0 & {}^B u_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

We have a description vector: $(-7, 0, 8, 0°, 90°, 90°)$, Roll-Pitch-Yaw, and our aim is to find a homogeneous representation of the pose as a matrix A



R can be computed by the following equation:

$$R = R_z(90°)R_y(90°)R_x(0°)$$

$$\begin{aligned}
R_z(90) &= \begin{bmatrix} \cos(90°) & -\sin(90°) & 0 \\ \sin(90°) & \cos(90°) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
R_y(90) &= \begin{bmatrix} \cos(90°) & 0 & \sin(90°) \\ 0 & 1 & 0 \\ -\sin(90°) & 0 & \cos(90°) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \\
R_x(0) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(0°) & -\sin(0°) \\ 0 & \sin(0°) & \cos(0°) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
R_s &= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix} \\
\Rightarrow A &= \begin{bmatrix} 0 & -1 & 0 & -7 \\ 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 8 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

Calculation of the Description Vector

The description vector $\vec{v} = (x, y, z, \alpha, \beta, \gamma)$ is to be determined from a given homogeneous matrix.

$$A = \begin{bmatrix} 0 & 0 & 1 & -7 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let us the position: $x = -7, \quad y = 0 \quad$ and $z = 8$.
Orientation can be calculated by matrix equation:

$$R_s = R_1 \cdot R_2 \cdot R_3$$

R_3 is 3×3-orientation part of homogeneous matrix, and product of rotation matrices corresponding to axes of rotation (Euler, roll-pitch-yaw).

Reformulating Matrix Equations

The matrix equations can be represented as 16 individual equations, where 12 are non trivial and can be solved by multiplying with the inverse matrix R_1^{-1} or R_3^{-1}:

$$R_1^{-1} \cdot R_s = R_2 \cdot R_3$$

or

$$R_s \cdot R_3^{-1} = R_1 \cdot R_2$$

Form of the Equations

After these reformulations there exist for each α, β, γ An equation of the form:

$$b \cdot \sin(\alpha) - a \cdot \cos(\alpha) = 0 \tag{3.1}$$

Or a set of equations of the form:

$$\sin(\alpha) = a \qquad \cos(\alpha) = b \qquad \text{for } \alpha, \beta, \gamma \tag{3.2}$$

Approaches to Determine the Angle:

An unknown angle should/must not be calculated with sin or cos because of its ambiguity. The approach is using arctan (arcus tangent) and projection of the angle to the correct quadrant.

With 3.1 or 3.2 one can compute α, since from 3.1, and 3.2 it follows that:

$$\alpha = \arctan(a/b)$$

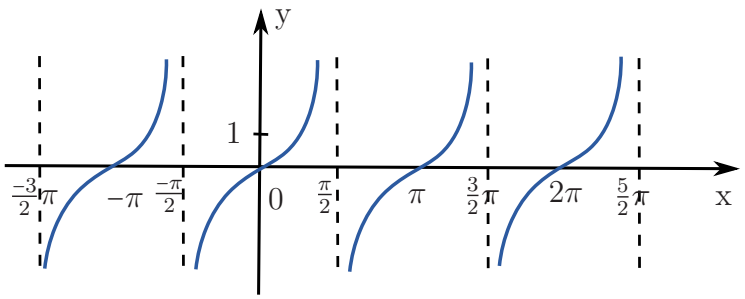


Figure 3.20: tan(x)

The Problem is that $\arctan(x)$ is ambiguous! So, we have to restrict ourselves to angles between $-\frac{\pi}{2}$ and $+\frac{\pi}{2}$.

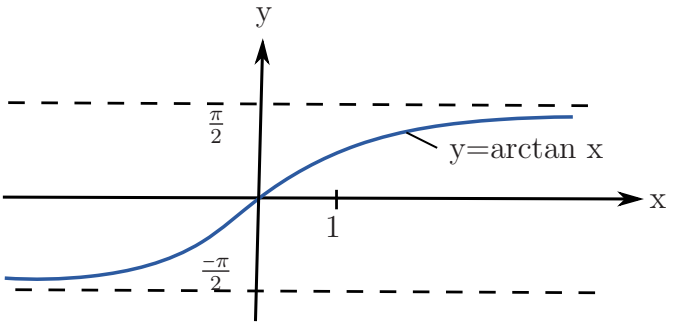


Figure 3.21: arctan(x)

3.4.3 ATAN2

To calculate α we do not use arctan, but the 2-argument arctangent ATAN2. At the starting point, equations ate of the form:

$$\sin(\alpha) = a \qquad \cos(\alpha) = b$$

These equations are only solveable for α, if $a^2+b^2 = 1$, Then $\sin^2(\alpha)+\cos^2(\alpha) = 1$. Especially, a and b can not be zero simultaneously.

$$\arcsin(a) = \alpha \qquad \text{where } \alpha \in [\frac{-\pi}{2}, \frac{\pi}{2}] \tag{3.3}$$

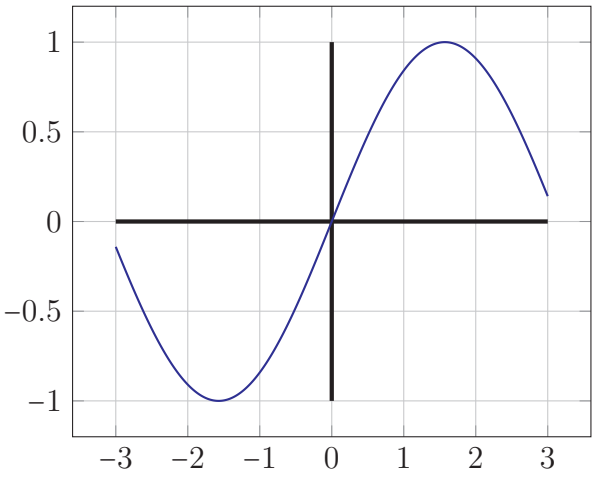


Figure 3.22: $\sin(x)$

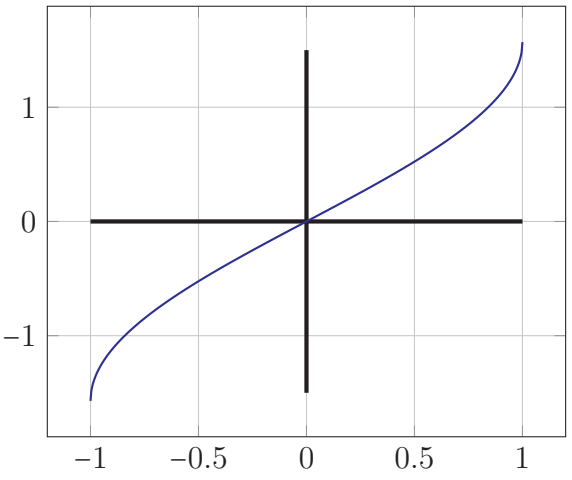


Figure 3.23: $\arcsin(x)$

$$\arccos(b) = \alpha \qquad \text{where } \alpha \in [0, \pi] \tag{3.4}$$

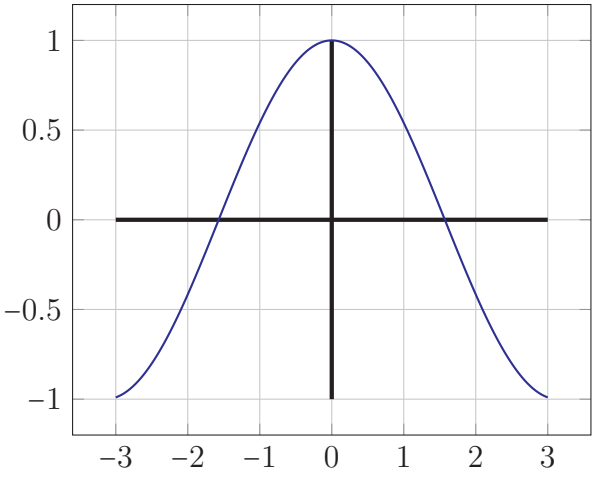


Figure 3.24: $\cos(x)$

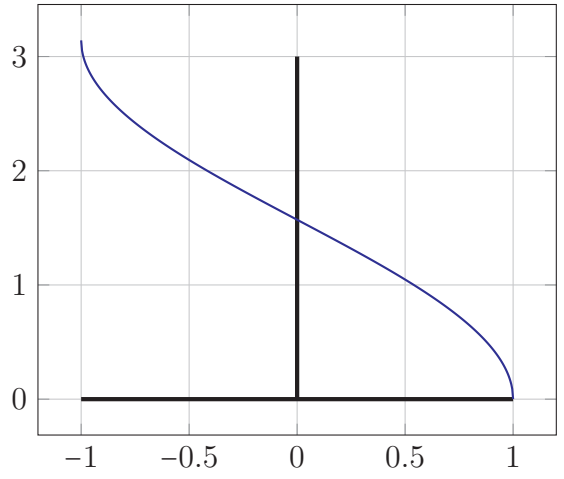


Figure 3.25: $\arccos(x)$

Definition:

The definition of the tangent is:

$$\tan(\alpha) = \frac{\sin(\alpha)}{\cos(\alpha)} \tag{3.5}$$

The definition of the $ATAN2$ function was as follows:

$ATAN2(a, b)$ where $a = \sin(\alpha)$ and $b = \cos(\alpha)$ due to 3.5

$$ATAN2(a, b) = \begin{cases} \arccos(b), & \text{if } a \geq 0 \\ -\arccos(b), & \text{if } a < 0 \end{cases}$$

due to 3.3 and 3.4.

$ATAN2$ can also be computed as:

$$ATAN2(a,b) = \tan^{-1}(a/b)$$

Choose quadrant with the signs of a and b.

Examples:
$ATAN2(-2,-2) = -135°$ (third quadrant)
$ATAN(2,2) = 45°$ (first quadrant)

Computing Angles with ATAN2

$$\begin{array}{rcl}
\sin(\theta) = a & \Rightarrow & \theta = \pm\, ATAN2(\sqrt{1-a^2}, a) \\
\cos(\theta) = b & \Rightarrow & \theta = \pm\, ATAN2(b, \pm\sqrt{1-b^2}) \\
a\cos(\theta) + b\sin(\theta) = 0 & \Rightarrow & \theta = ATAN2(a,-b) \\
\text{has two solutions} & \Rightarrow & \theta = ATAN2(-a,b) \\
a\cos(\theta) + b\sin(\theta) = c & \Rightarrow & \theta = ATAN2(b,a) \pm ATAN2(\sqrt{a^2+b^2+c^2}, c) \\
a\cos(\theta) - b\sin(\theta) = c & \text{and} & \\
a\cos(\theta) + b\sin(\theta) = d & \Rightarrow & \theta = ATAN2(ad-bc, ac+bd)
\end{array}$$

3.4.4 Computation of Euler Angles

Euler angles can be computed for a general orientation matrix as follows:

let us consider the matrix equation

$$R_s = R_z(\alpha) \cdot R_{y'}(\beta) \cdot R_{z''}(\gamma) = \begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} \tag{3.6}$$

Multiplying from the left with $R_z(\alpha)^{-1}$ holds:

$$R_z(\alpha)^{-1} \cdot R_z(\alpha) \cdot R_{y'}(\beta) \cdot R_{z''}(\gamma) = R_z(\alpha)^{-1} \cdot R_s$$

For orthogonal matrices it holds that $A^{-1} = A^T$. therefore:

$$R_{y'}(\beta) \cdot R_{z''}(\gamma) = R_z(\alpha)^{-1} \cdot R = R_z(\alpha)^T \cdot R_s$$

$$\begin{bmatrix} C\beta \cdot C\gamma & -C\beta \cdot S\gamma & S\beta \\ S\gamma & C\gamma & 0 \\ -S\beta \cdot C\gamma & S\beta \cdot S\gamma & C\beta \end{bmatrix} =$$

$$\begin{bmatrix} C\alpha n_x + S\alpha n_y & C\alpha o_x + S\alpha o_y & C\alpha a_x + S\alpha a_y \\ -S\alpha n_x + C\alpha n_y & -S\alpha o_x + C\alpha o_y & -S\alpha a_x + C\alpha a_y \\ n_z & o_z & a_z \end{bmatrix}$$

Example:

The following equations are given:

row.column

$$\begin{aligned}
1.1: \quad C\beta \cdot C\gamma &= C\alpha \cdot n_x + S\alpha \cdot n_y \\
1.2: \quad -C\beta \cdot S\gamma &= C\alpha \cdot o_x + S\alpha \cdot o_y \\
1.3: \quad S\beta &= C\alpha \cdot a_x + S\alpha \cdot a_y \\
2.1: \quad S\gamma &= -S\alpha \cdot n_x + C\alpha \cdot n_y \\
2.2: \quad C\gamma &= -S\alpha \cdot o_x + C\alpha \cdot o_y \\
2.3: \quad 0 &= -S\alpha \cdot a_x + C\alpha \cdot a_y \\
3.1: \quad -S\beta \cdot C\gamma &= n_z \\
3.2: \quad S\beta \cdot S\gamma &= o_z \\
3.3: \quad C\beta &= a_z
\end{aligned}$$

For compution Angle α, from (2.3) it follows that:

$$S\alpha \cdot a_x = C\alpha \cdot a_y \Leftrightarrow \frac{S\alpha}{C\alpha} = \tan(\alpha) = \frac{a_y}{a_x}$$

Therefor $\alpha = ATAN2(a_y, a_x)$

For computin Angle β, from (1.3), (3.3) it follows that:

$$\beta = ATAN2(C\alpha a_x + S\alpha a_y, a_z)$$

For compution Angle γ, from (2.1), (2.2) it follows that:

$$\gamma = ATAN2(C\alpha n_x + S\alpha n_y, -S\alpha o_x + C\alpha o_y)$$

Note that, α is present in the solutions of β and γ.

3.5 Concatenated Poses

Poses are most of the time not given relative to the BCS, but relative to a more suitable CS (relative definition). It is about converting coordinates to different reference systems (e.g. BCS) required. Advantages of the relative position definition in robotics are, on the one hand, the reduction of the tracking effort for object movements and, on the other hand, the individual ones only cover small distances.

Rotation/Translation of Poses

Let ${}^{BCS}_{A}H_{obj} = (4\times4)$ be the pose of an object in frame A relative to BCS. Let ${}^{A}_{B}H_{obj} = (4\times4)$ be the pose of an object in frame B relative to the OCS of A. And also let ${}^{BCS}_{B}H_{obj} = (4\times4)$ be the pose of an object in frame B relative to BCS.

It holds:

$$ {}^{BCS}_{B}H_{obj} = {}^{BCS}_{A}H_{obj} \cdot {}^{A}_{B}H_{obj} $$

Notation is more compact, compared to the cartesian notation:

$$ {}^{BCS}R + {}^{BCS}\vec{v} = {}^{BCS}_{A}R_1 \cdot ({}^{A}_{B}R_2 + {}^{A}\vec{v}_B) + {}^{BCS}\vec{v}_A = {}^{BCS}_{A}R_1 \cdot {}^{A}_{B}R_2 + ({}^{BCS}_{A}R_1 \cdot {}^{A}\vec{v}_B + {}^{BCS}\vec{v}_A) $$

Example:

Pose of object in frame 1 relative to BCS: ${}^{BCS}_{1}H_{obj}$

Pose of object in frame 2 relative to frame 1: ${}^{1}_{2}H_{obj}$

Pose of object in frame 3 relative to frame 2 : ${}^{2}_{3}H_{obj}$

Pose of object in frame 3 relative to BCS : ${}^{BCS}_{3}H_{obj}$

$$ {}^{BCS}_{3}H_{obj} = {}^{BCS}_{1}H_{obj} \cdot {}^{1}_{2}H_{obj} \cdot {}^{2}_{3}H_{obj} $$

In order to use concatenated poses each matrix must be defined relative to the frame defined by its left matrix. So:

$$ \prod_{i=1}^{n} ({}^{i-1}_{i}H) \qquad \text{where} \qquad 1 \le i \le n \quad \text{and} \quad {}^{0}H = BCS $$

System of object H_1,is generated by a transformation $((3,3,0)^T, R_z(90°))$ of an arbitrary coordinate system B: ${}^{B}_{1}H$

System of object H_2, is generated by a transformation $((-5,-5,0)^T, R_z(-180°))$ of the system object H_1: ${}^{1}_{2}H$

Example:

$$ {}^{BCS}_{1}H = \begin{bmatrix} 0 & -1 & 0 & 3 \\ 1 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{1}_{2}H = \begin{bmatrix} -1 & 0 & 0 & -5 \\ 0 & -1 & 0 & -5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{BCS}_{2}H = {}^{BCS}_{1}H \cdot {}^{1}_{2}H = \begin{bmatrix} 0 & 1 & 0 & 8 \\ -1 & 0 & 0 & -2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

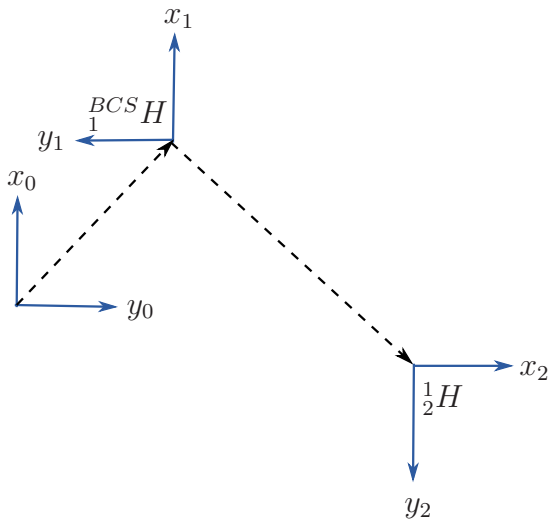


Figure 3.26: Linked Location Description: Example

vector z is the same, so is ommited.

4. Direct Kinematics

4.1 Degree of Freedom of a Robotic System

Degree of Freedom (DoF) f of an Object in E_3

In general, the *degree of freedom* is the number of independent movements in relation to the BCS (minimum number of translations and rotations to fully describe the position of the object). For objects with unconstraned movement in 3-dimensional space: $f = 6$ (3 translations and 3 rotations)

4.1.1 Kinematic Degree of Freedom of a Robot

Degree of freedom of a rotatinal joint is: $F_R \leq 3$

- Hinge joint
- Cardan joint
- Linear joint
- Spherical joint

Degree of freedom of a translational joint is: $F_T = 1$
Number of joints of a robot is: n (in general $n \geq 6$)
Kinematic degrees of freedom is: $F = \sum_{i=1}^{n} (F_{R_i} + F_{T_i})$

Relation between f and F

The relationship holds: $F \geq f$

Examples:

- The degree of freedom f of an 8-axis robot is 6, and its Kinematic degree of freedom F is 8.

- The degree of freedom f of a human hand is 6, and its kinematic degree of freedom F is 22.

- The degree of freedom f of a human arm including shoulder is 6, and its kinemaic degree of freedom F is 12.

In order to reach a DoF f=6 for a robot‘s effector, it requires at least F=6 axes of movement.

4.2 Models

Terminology

Geometry:
Mathematical description of robot form

The *geometric model* represents bodies graphically. It serves as the basis of movement calculation, is used to determine the forces and moments that act, and is the starting point for distance and collision measurement .

Kinematics:
Geometric and analytic description of mechanical systems‘ states of movement

Dynamics:
Investigates movement of objects based on the forces and moments acting upon them

4.2.1 Kinematic Models

A *kinematic model* describes the pose (position and orientation) of bodies in space with the help of the geometric model. A *kinematic chain* is formed by several bodies that are kinematically connected via joints, such as a robot arm. A distinction is made between closed kinematic chains and open kinematic chains. The purpose of the kinematic model is to determine the relationship between joint values and poses and the accessibility analysis.

4.2.2 Dynamic Model

A *Dynamic model* describes and calculates the forces and moments acting in a mechanical multi-body system. The purpose of the dynamic model is the dimensioning of the drive mechanism, the optimization of the construction (lightweight construction), the consideration of bending and stiffness and the support of the controller design.

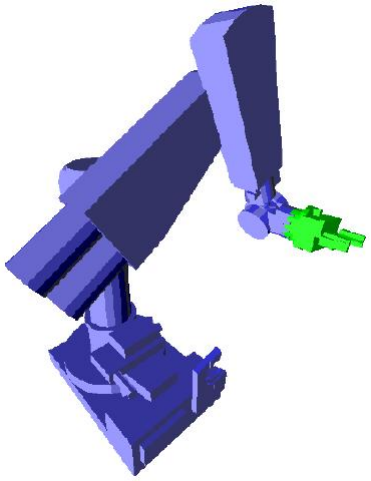


Figure 4.1: Geometric Model Classification

4.3 Geometric Model

Classification:

- 2D-model
- 2.5D-model
- 3D-model
- Edge or wire-frame models
- Surface models
- Volumetric models

Block World

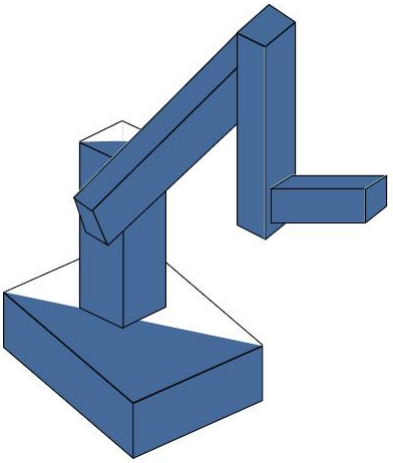


Figure 4.2: Geometric Model: Block World

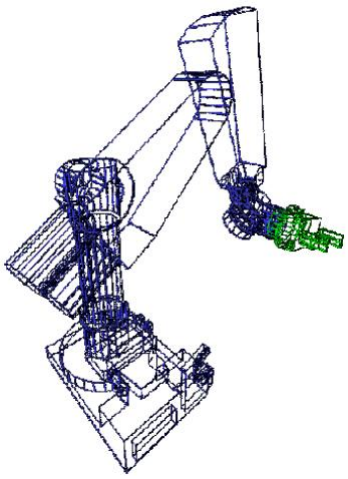


Figure 4.3: Geometic Model: Edge Model

Bodies represented by enveloping cuboids (bounding boxes). Calculation is easy with regard to collision avoidance.

Edge Model

The bodies are represented by polygons (edges). This is used for quick visualization.

Volumetric Models

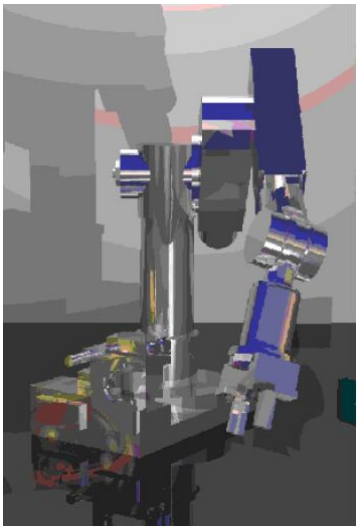


Figure 4.4: Geometric Model: Volumetric Models

This model is a precise representation of bodies and is used for the exact computation of the contact points for collision avoidance, and also is used for representation with animations.

Links and Joints

See figure 4.5

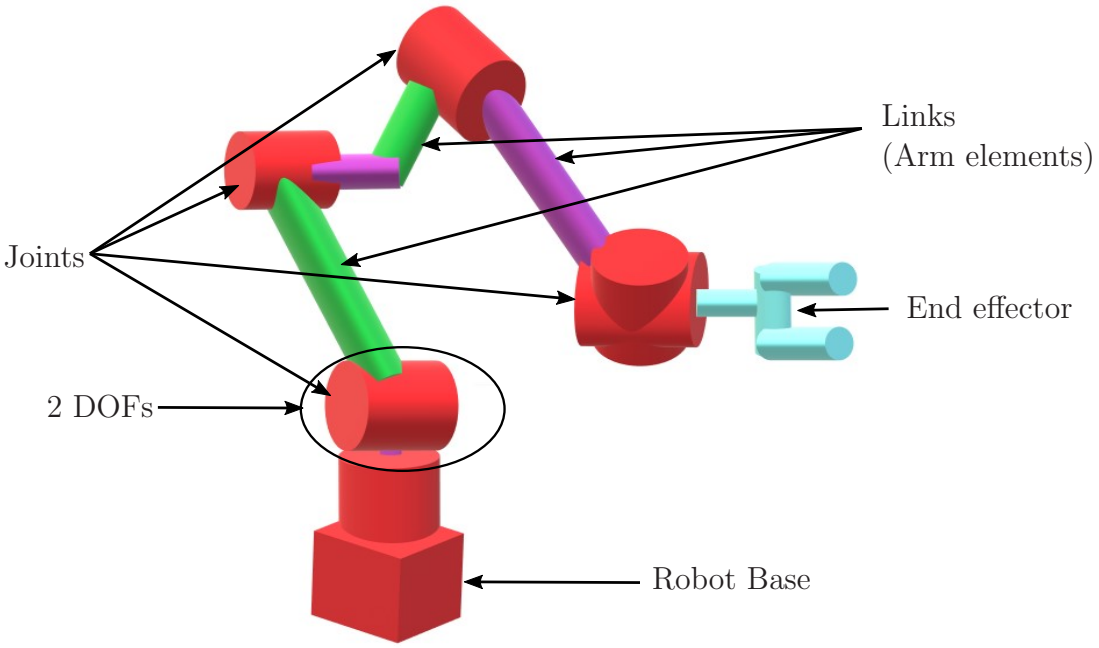


Figure 4.5: Links and Joints

Puma 260

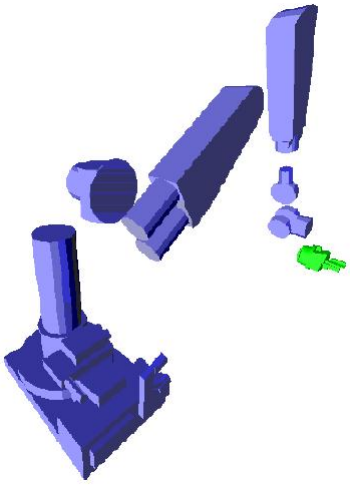


Figure 4.6: Volmetric Model Puma 260

In Volmetric model, each arm element corresponds to a rigid body and it is joined to to the next via a linear or rotational joint. Each joint also has only one degree of freedom (rot. Or transl.).
Kinematic Pair = Joint + joined arm elements.
The Puma 260 is a 6-axis robot with a basis and 6 arm elements(links), as can be seen in figure 4.6.

Coordinate Systems:

In order to describe the kinematics of a robot (kinematic chain), it is necessary to define the links´ poses in relation to a reference coordinate system. Each links received a fixed

local coordinate system. The origin of each coordinate system lies in the joint that moves the respective link. A transformation matrix must be determined for each link element, which converts the respective local coordinate system into the reference coordinate system. This conversion of the local coordinate systems into the reference coordinate system is achieved either by a description vector or a 4×4 homogeneous transformation matrix.

Link Parameters

Every link element i is connected through 2 confining joints i and $i+1$ (see figure 4.8). Let g_i and g_{i+1} be movement axes of joints which are skewed to each other. Let a_i be the normal between g_i and g_{i+1}. Then, the distance of intersections of a_{i-1} and a_i with g_i is referred to as joint offset d_i. Angle θ_i between a_{i-1} and a_i is referred to as *joint angle*, length of a_i (shortest distance between g_i and $g_i + 1$) is called *link length*, and angle α_i between g_i and g_{i+1} is called *link twist*.

Parameters of Joints

Parameter	Symbol	Rotatioal joinr	Prismatic joint
Link length	a	invariant	invariant
Link twist	α	invariant	invariant
Joint distance	d	invariant	variable
Joint angle	θ	variable	invariant

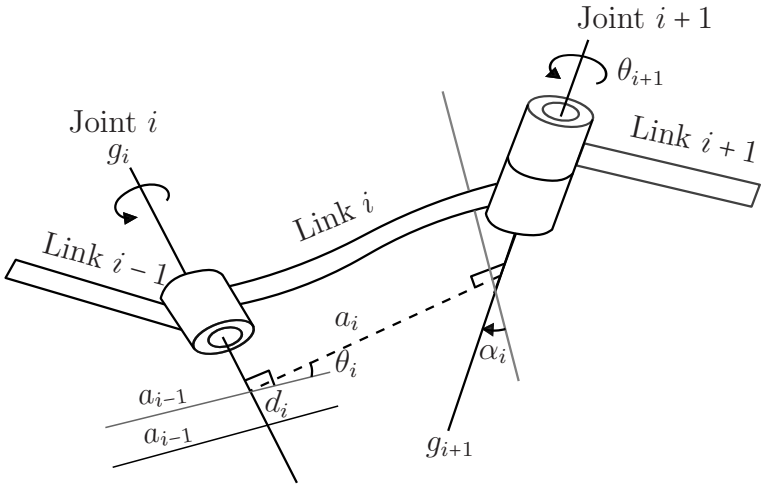


Figure 4.7: Denavit-Hartenberg-Convention

4.3.1 Denavit-Hartenberg-Convention

The transformation from the OCS of the $i-th$ link to the OCS of the $(i-1)-th$ link takes place according to the *Denavit-Hartenberg Convention*.

Coordinate systems for every joint are defined as follows: (see figure 4.7)

- Origin of CS i lies in intersection of a_i and g_{i+1}.
- Axis z_i lies along the joint axis g_{i+1}.
- Axis x_i follows as extension of normal a_i.
- Complemented by axis y_i, so that a clockwise CS follows.
- Origin of CS 0 can be freely placed along g_1.
- The last coordinate system lies in the end effector.

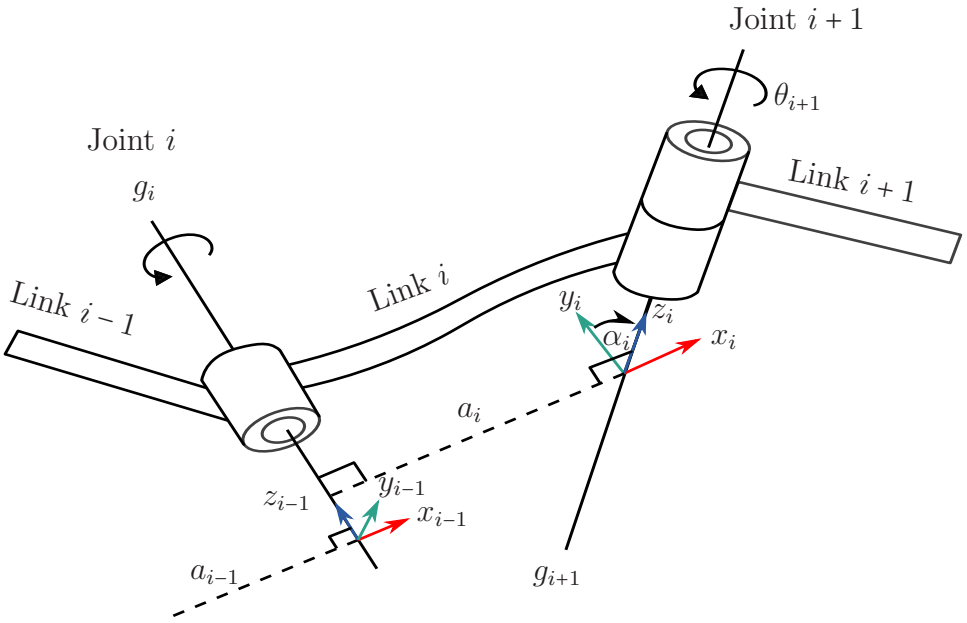


Figure 4.8: Derivation of the link element parameters

For transformation from OCS of link i to OCS of link $i-1$ via Denavit-Hartenberg-Transformation some equirements are needed:

- All CS follow the Denavit-Hartenberg-Convention
- Parameters of link i are known

4.3.2 Denavit-Hartenberg-Transformation

Having all of the requirements, OCS of link i is transformed to OCS of link $i-1$ as following:

1. Rotation θ_i around z_{i-1}-axis so that x_{i-1}-axis lies paralle to x_i-axis.

$$R_{z_{i-1}}(\theta_i) = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & 0 \\ \sin(\theta_i) & \cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Translation d_i along z_{i-1}-axis to the intersection point of z_{i-1} and x_i.

$$T_{z_{i-1}}(d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. Translation a_i along x_i-axis in order to make the origins of the coordinate systems congruent.

$$T_{x_i}(a_i) = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4. Rotation α_i around x_i-axis, to transform the z_{i-1}-axis into z_i.

$$R_{x_i}(\alpha_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In matrix notation, the Denavit-Hartenberg-Transformation (D-H-Transformation) of the coordinate system of link $i-1$ to i looks like:

$$\begin{aligned} {}^{i-1}_{i}A &= R_{z_{i-1}}(\theta_i) \cdot T_{z_{i-1}}(d_i) \cdot T_{x_i}(a_i) \cdot R_{x_i}(\alpha_i) \\ &= \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \cdot \cos(\alpha_i) & \sin(\theta_i) \cdot \sin(\alpha_i) & a_i \cdot \cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \cdot \cos(\alpha_i) & -\cos(\theta_i) \cdot \sin(\alpha_i) & a_i \cdot \sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Denavit-Hartenberg-Transformation (Inverse)

The inverse of transformation matrix ${}^{i-1}_{i}A^{-1}$ corresponds to the transformation from CS_{i-1} to CS_i:

$$\begin{aligned} {}^{i-1}_{i}A^{-1} &= {}^{i}_{i-1}A \\ &= \begin{bmatrix} \cos(\theta_i) & \sin(\theta_i) & 0 & -a_i \\ -\cos(\alpha_i) \cdot \sin(\theta_i) & \cos(\theta_i) \cdot \cos(\alpha_i) & \sin(\alpha_i) & -d_i \cdot \sin(\alpha_i) \\ \sin(\theta_i) \cdot \sin(\alpha_i) & -\sin(\alpha_i) \cdot \cos(\theta_i) & \cos(\alpha_i) & -d_i \cdot \cos(\alpha_i) \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

To determine the pose of the manipulator (Tool Center Point, TCP) in relation to the BCS, the D-H-matrices are multiplied in order of the corresponding links:

$${}^{BASIS}_{TCP}A = {}^{BASIS}_{1}A(\theta_1) \cdot {}^{1}_{2}A(\theta_2) \cdots {}^{n-2}_{n-1}A(\theta_{n-1}) \cdot {}^{n-1}_{TCP}A(\theta_n)$$

4.3.3 Approach with D-H

Step 1 (Finding the Normal a_i):

- Joint axes g_i.
- a_i points from g_i to g_{i+1}.

Step 2 (Defining the CS)

- Origin O_i in the intersection of a_i with g_{i+1}.
- x_i lies on normal a_i and points in the same direction
- z_i lies on g_{i+1}, in the direction allowing joint rotation or translation in mathematically positive sense
- y_i completes the clockwise CS; at TCP, y_i indicates width of opening.

see figure 4.9

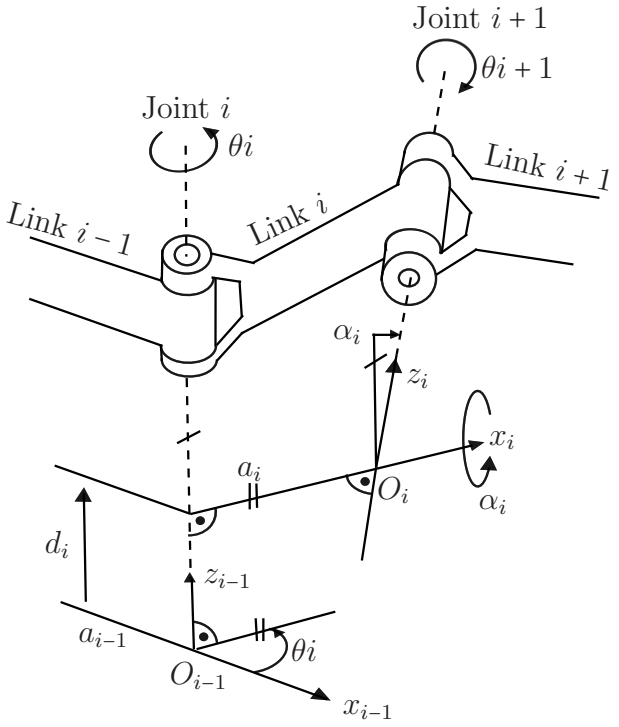


Figure 4.9: Approach with D-H

Special Cases of Steps (1),(2)

- g_i **and** g_{i+1} **intersect.**
  In this case, the irection of x_i is not defined and x_i arises from x_{i-1} through smallest possible rotation around z_{i-1}.

- g_i **and** g_{i+1} **are parallel or collinear.**
  In this case, intersection of a_i and g_{i+1} is not well defined, and normals can be determined by backstepping (recursively) with starting at next uniquely identifiable origin O_j with $j > i$. At last joint, the origin is in the center of TCP.

Step 3 (Determining the Transformation Matrix ${}^{i+1}_{i}A$):

- Rotation of CS of joint i around z_{i-1} with joint angle θ_i
  $\rightarrow x'_{i-1}$ is parallel to x_i.

- Translation by d_i along z_{i-1}
  $\rightarrow$ origin lies in intersection of z_{i-1} and x_i.

- Translation by $|a_i|$ along x_i
  $\rightarrow$ Origins are congruent.

- Rotation around x_i with twist α_i
  $\rightarrow z'_{i-1}$ is parallel to z_i.

- Combination of all these results:

$$ {}^{i-1}_{i}A = R_{z_{i-1}}(\theta_i) \cdot T_{z_{i-1}}(d_i) \cdot T_{x_i}(|a_i|) \cdot R_{x_i}(\alpha_i) $$

4.4 Examples

4.4.1 Example 1

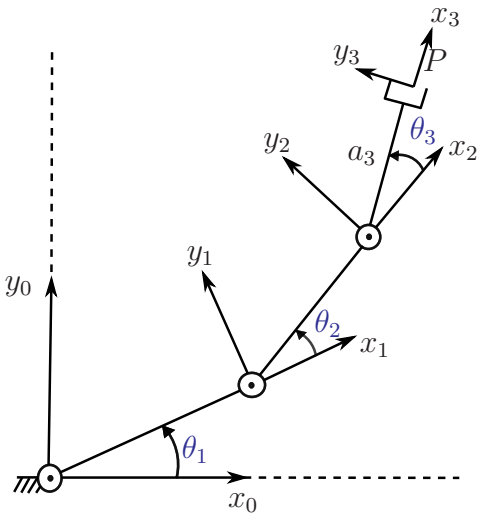


Figure 4.10: Direct kinematics:Example 1-1

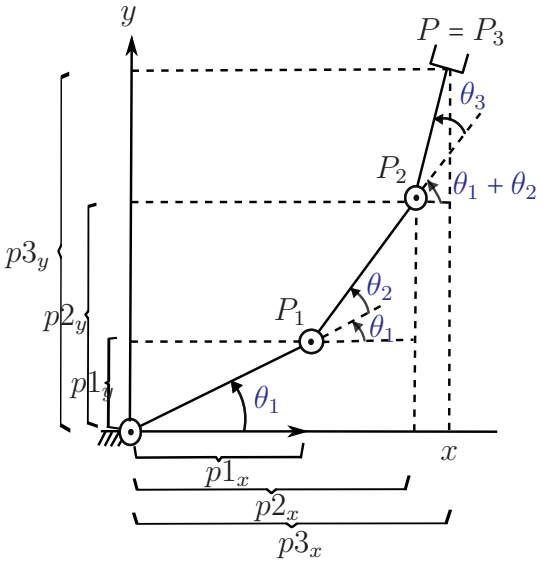


Figure 4.11: Direct kinematics: Example 1-2

Joint	a_i	α_i	d_i	θ_i
1	a_1	0	0	θ_1
2	a_2	0	0	θ_2
3	a_3	0	0	θ_3

$$\begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cdot\cos(\alpha_i) & \sin(\theta_i)\cdot\sin(\alpha_i) & a_i\cdot\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cdot\cos(\alpha_i) & -\cos(\theta_i)\cdot\sin(\alpha_i) & a_i\cdot\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Rightarrow {}^{i-1}_{i}A = \begin{bmatrix} c_i & -s_i & 0 & a_i\cdot c_i \\ s_i & c_i & 0 & a_i\cdot s_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Result:

$${}^{0}_{3}A = {}^{0}_{1}A\cdot {}^{1}_{2}A\cdot {}^{2}_{3}A = \begin{bmatrix} c_{123} & -s_{123} & 0 & a_1\cdot c_1 + a_2\cdot c_{12} + a_3\cdot c_{123} \\ s_{123} & c_{123} & 0 & a_1\cdot s_1 + a_2\cdot s_{12} + a_3\cdot s_{123} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

with:
$s_{123} = \sin(\theta_1 + \theta_2 + \theta_3)$ and
$c_{123} = \cos(\theta_1 + \theta_2 + \theta_3)$

4.4.2 Example 2

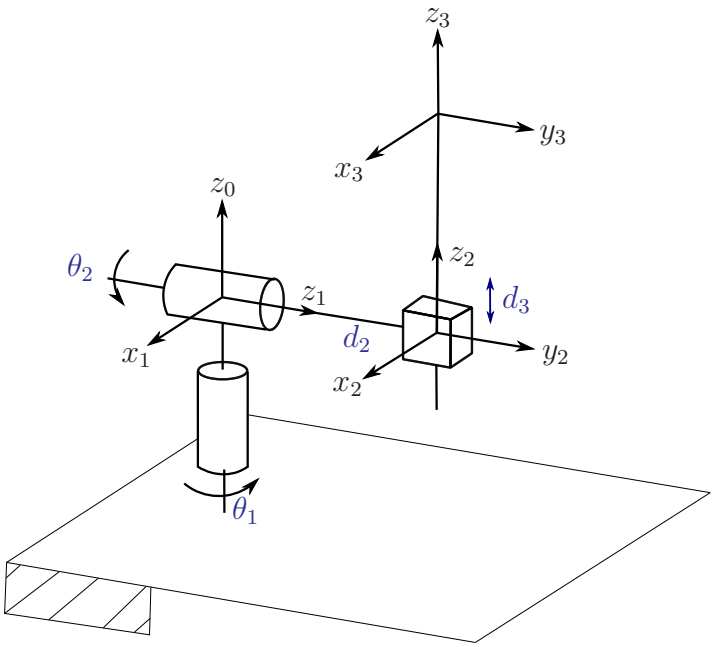


Figure 4.12: Direct kinematics: Example 2

Joint	a_i	α_i	d_i	θ_i
1	0	$-90°$	0	θ_1
2	0	$90°$	d_2	θ_2
3	0	0	d_3	0

$$
\begin{aligned}
{}^{0}_{1}A &= \begin{bmatrix} c_1 & 0 & -s_1 & 0 \\ s_1 & 0 & c_1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
{}^{1}_{2}A &= \begin{bmatrix} c_2 & 0 & s_2 & 0 \\ s_2 & 0 & -c_2 & 0 \\ 0 & 1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
{}^{2}_{3}A &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

Result:

$$
{}^{0}_{3}A = {}^{0}_{1}A \cdot {}^{1}_{2}A \cdot {}^{2}_{3}A = \begin{bmatrix} c_1 \cdot c_2 & -s_1 & c_1 \cdot s_2 & c_1 \cdot s_2 \cdot d_3 - s_1 \cdot d_2 \\ s_1 \cdot c_2 & c_1 & s_1 \cdot s_2 & s_1 \cdot s_2 \cdot d_3 + c_1 \cdot d_2 \\ -s_2 & 0 & c_2 & c_2 \cdot d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

4.4.3 Example 3

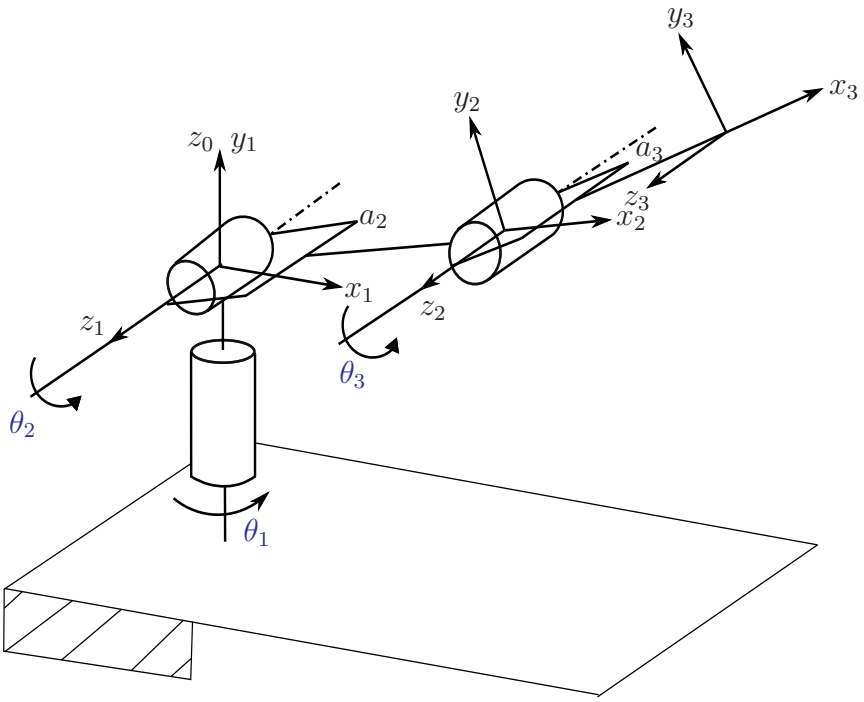


Figure 4.13: Direct Kinematics: Example 3

Joint	a_i	α_i	d_i	θ_i
1	0	90°	0	θ_1
2	a_2	0	0	θ_2
3	a_3	0	0	θ_3

4.4.4 Example 4

Joint	a_i	α_i	d_i	θ_i
1	0	90°	0	θ_1
2	a_2	0	0	θ_2
3	0	90°	0	$\theta_3 + 90°$
4	0	−90°	d_4	θ_4
5	0	90°	0	$\theta_5 - 90°$
6	0	0	d_6	$\theta_6 + 90°$

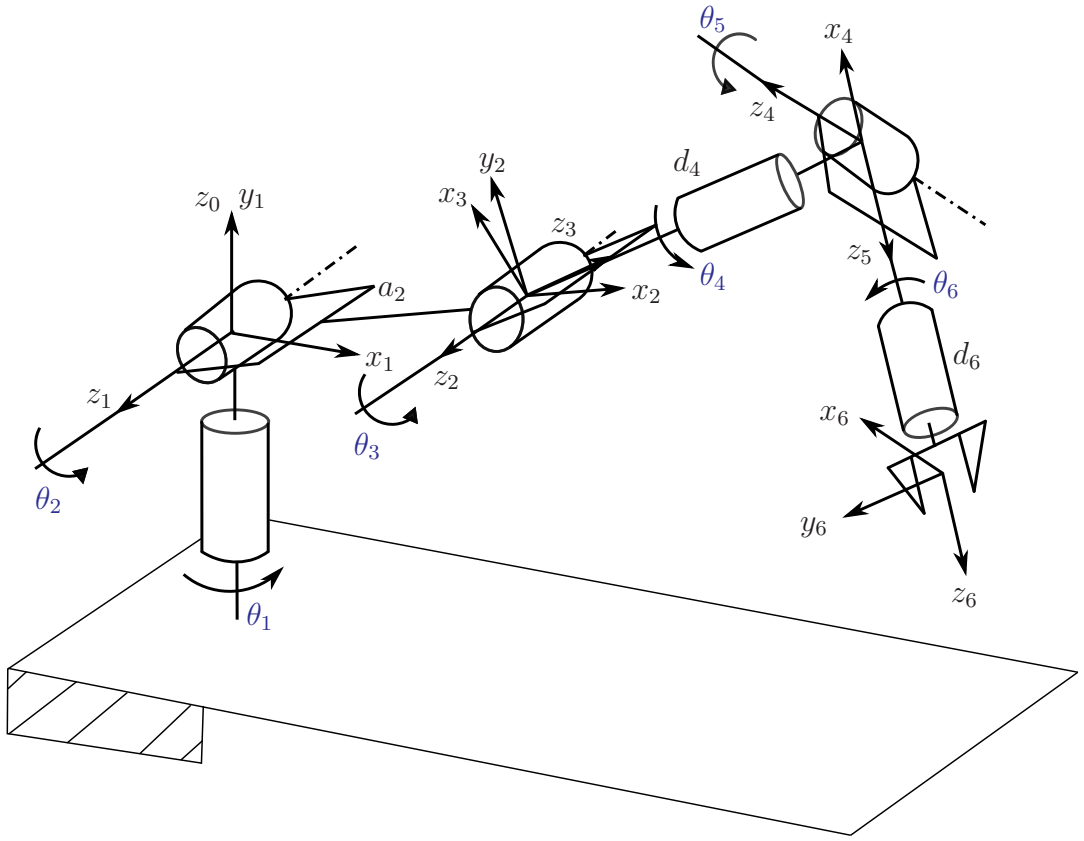


Figure 4.14: Direct Kinematics: Example 4

4.5 Robot Kinematics

The robot kinematics describes the relations between the joint angle space (robot coordinates, configuration space) and the space of the positioning and orientation of the end effector (EE) in world coordinates.

4.5.1 Direct Kinematics Problem (Forward Kinematics)

The position of the manipulator is to be determined from the D-H-parameters and the joint angles. The position of the gripper (TCP) in relation to the BCS (basis) is given by:

$$ {}^{BASIS}_{TCP}A = {}^{BASIS}_{1}A(\theta_1) \cdot {}^{1}_{2}A(\theta_2) \cdots {}^{n-2}_{n-1}A(\theta_{n-1}) \cdot {}^{n-1}_{TCP}A(\theta_n) $$

θ ist given, → Calculate the equation.

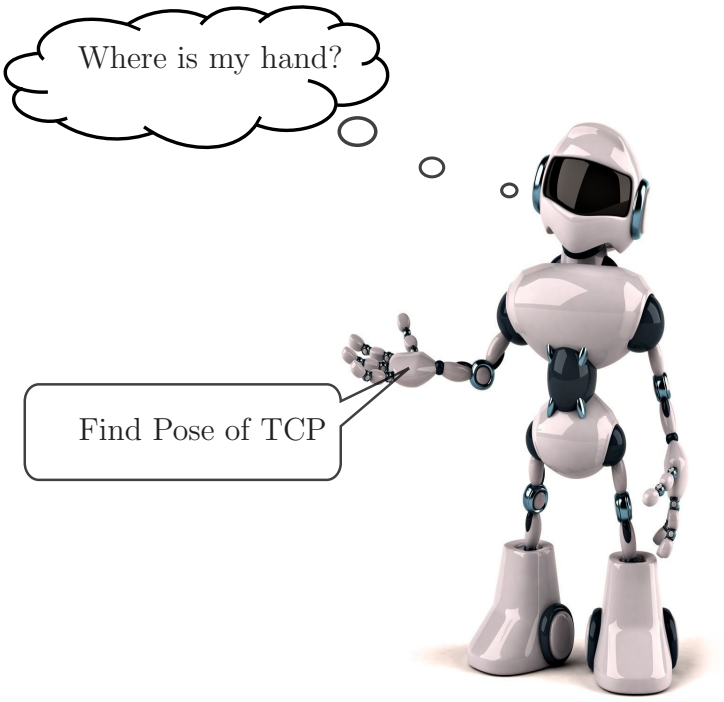


Figure 4.15: Direct Kinematics Problem

4.6 Summary

- Sketch of the manipulator
- Enumerate links: Basis =0, last link =n
- Identify and enumerate the joints
- Draw axes z_i for every joint i
- Determine parameters a_i between z_{i-1} and z_i
- Draw x_{i-1}-axes
- Determine parameter α_{i-1} (twist around x_{i-1}-axes)
- Determine parameter d_i (offset)
- Determine angle θ_i around z_i-axes
- Joint-transformation matrices $A_{i-1,1}$

Inverse kinematic problem (backward kinematics) will be discussed in chapter 7.

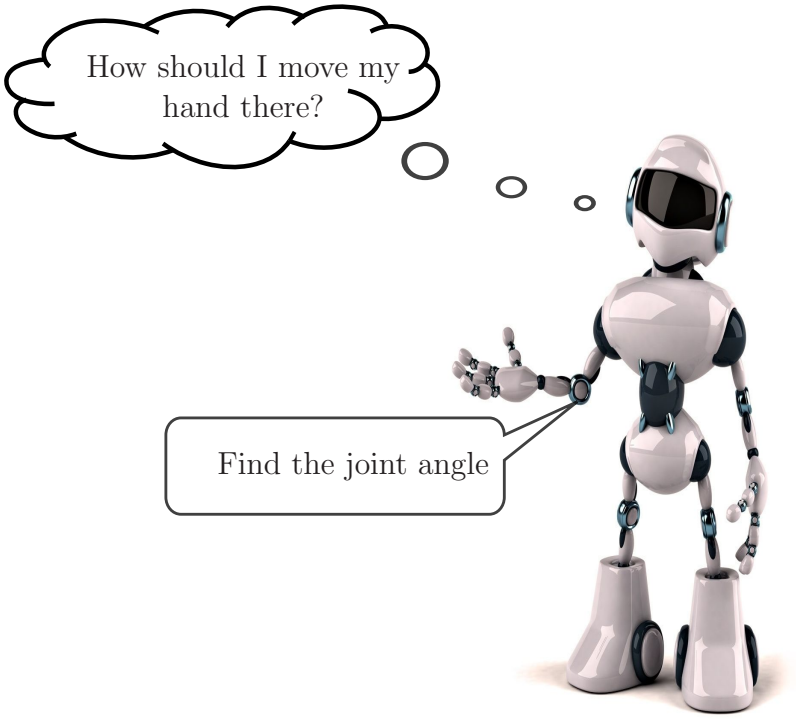


Figure 4.16: Inverse Kinematics Problem

5. Direct Kinematics - Quaternions

Problems of (homogeneous) rotation matrices are highly redundant. This is because of many arithmetic operations at concatenation and the singularities of the matrices. For the orientation of a rigid body, it is generally sufficient to specify a rotation axis (3-dimensional vector $\vec{g}$) and an angle θ. It will reduce the required computational effort sufficiently. See figure 5.1.

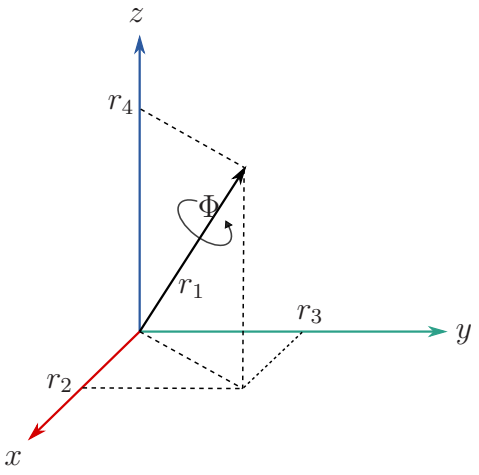


Figure 5.1: Quaternions

5.1 Real Quaternions

A *real quaternion* consists of 4 real values: $Q = (r_1, r_2, r_3, r_4)$ with $r_1, r_2, r_3, r_4 \in \mathbb{R}$ often represented in the form of a linear vector space over $\mathbb{R}$:
$Q = r_1 + i \cdot r_2 + j \cdot r_3 + k \cdot r_4$ (extension of complex numbers).
The following multiplicative link table (not commutative) applies to the basic elements $1, i, j, k$:

·	1	i	j	k
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

The scalar part is r_1 (angle of rotation) and the vector part is $i \cdot r_2 + j \cdot r_3 + k \cdot r_4$ (axis of rotation). Quaternions can be used to display all rotations in which the axis of rotation passes through the origin of the reference system.

- **Conjugated:** $\overline{Q} = r_1 - i \cdot r_2 - j \cdot r_3 - k \cdot r_4$
- **Magnitude:** $|Q| = \sqrt{Q \cdot \overline{Q}}$
- **Inverse:** $Q^{-1} = \frac{\overline{Q}}{|Q|^2} \rightarrow Q \cdot Q^{-1} = Q^{-1} \cdot Q = 1$

Examples:

Let us consider $Q_1 = (3, 2, -4, 1)$ and $Q_2 = (4, -3, 1, -5)$.
then it holds:
$Q_1 + Q_2 = (7, -1, -3, -4)$

$Q_1 \cdot Q_2 = (3+2i-4j+k)(4-3i+j-5k) = 12-9i+3j-15k+8i-6i^2+2ij-10ik-16j+12ji-4j^2+20jk+4k-3ki+kj-5k^2 = 12-9i+3j-15k+8i+6+2k+10j-16j-12k+4+20i+4k-3j-i+5 = (12+6+4+5)+(-9+8+20-1)i+(3+10-16-3)j+(-15+2-12+4)k = 27+18i-6j-21k = (27, 18, -6, -21)$

$Q_2 \cdot Q_1 = (27, -20, -20, -1)$

$Q_1^{-1} = \frac{(3-2i+4j-k)}{30}$

5.2 Rotation of Points by Means of Quaternions

We call $|Q| = 1$ *Unit quaternion* and $Q^{-1} = \overline{Q}$, then $|Q|^2 = 1 \rightarrow$ Simple forward / backward calculation.
Rotation of point $\vec{p}$ at axis $\vec{g}$ with θ (see figure 5.2) is defined as following:

1. Creat a unique quaternion from $\vec{g}$ und θ:
   - Standardization of $\vec{g}$ to 1
   - $Q = [\cos(\frac{\theta}{2}), \sin(\frac{\theta}{2})\vec{g}]$, since $\cos^2(\theta) + \sin^2(\theta) = 1$
2. Represent point $\vec{p}$ as quaternion: $P = [0, \vec{p}]$
3. Final rotation: $P' = Q \cdot P \cdot Q^{-1} = Q \cdot P \cdot \overline{Q}$

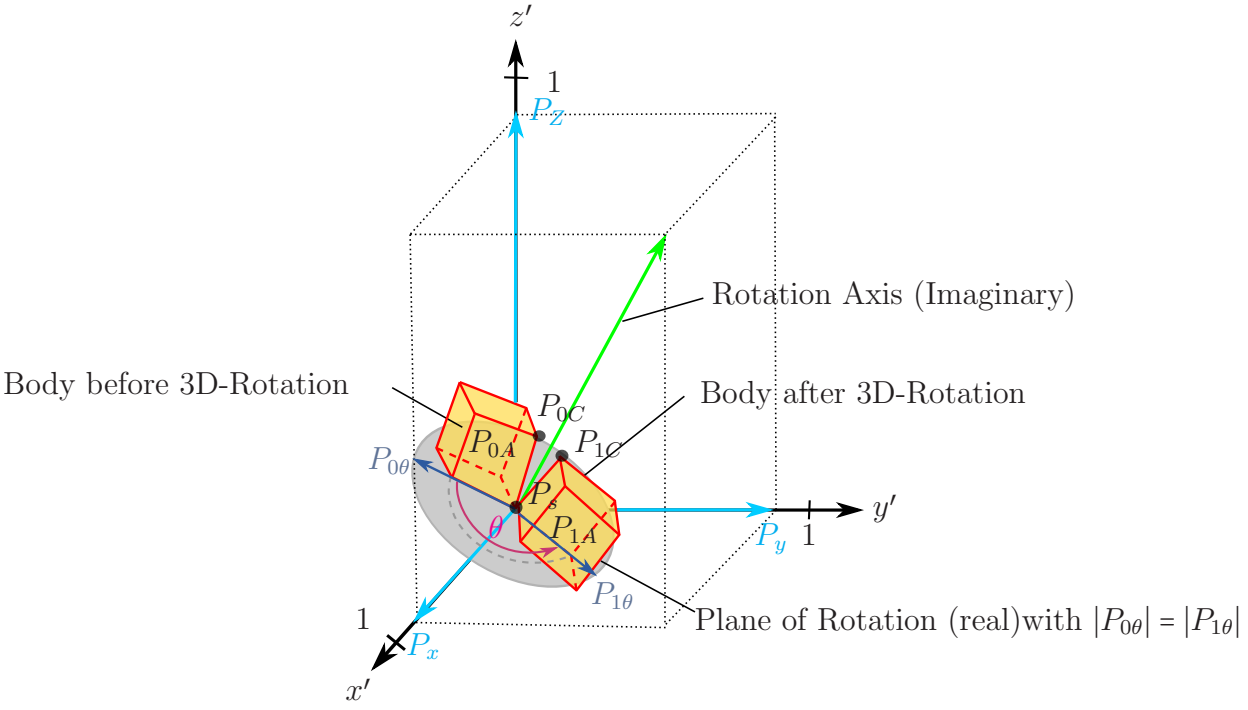


Figure 5.2: Rotation of Points by Means of Quaternions

5.3 Converting between Quaternion and Rotation matrix

Let us condider *Rotation quaternion*: $Q = (s, (x, y, z))$
From rotation by means of unit quaternion $|Q| = 1 \Rightarrow Q^{-1} = \overline{Q}$ follows the rotation matrix R:

$$R = \begin{pmatrix} 1-2(y^2+z^2) & 2xy-2sz & 2sy+2xz \\ 2xy+2sz & 1-2(x^2+z^2) & -2sx+2yz \\ -2sy+2xz & 2sx-2yz & 1-2(x^2+y^2) \end{pmatrix}$$

From the rotation matrix R with entries $r_{ij}, i, j \in \{1, 2, 3\}$ the corresponding rotation quaternion is calculated $Q = (s, (x, y, z))$ as follows:

$$s = \frac{1}{2}\sqrt{1 + r_{11} + r_{22} + r_{33}}$$
$$x = \frac{r_{32} - r_{23}}{4s}$$
$$y = \frac{r_{13} - r_{31}}{4s}$$
$$z = \frac{r_{21} - r_{12}}{4s}$$

5.4 Dual Quaternions

Real quaternions are suitable for the description of the orientation, but not the position of an object. In order to be able to express the position in a quaternion in addition to the

orientation, the 4 real values are replaced by complex numbers:

- $Q = (q_1, q_2, q_3, q_4)$
- $q_i = qr_i + \epsilon \cdot qd_i$
- $\epsilon^2 = 0$
- d_1 is the angle value and the displacement length.
- d_2, d_3, d_4 are used to describe a directed straight line in space, in which rotation and translation take place.

Denote the real part by $q_r = (q_{r_1}, q_{r_2}, q_{r_3}, q_{r_4})$, and the dual part by $q_d = (q_{d_1}, q_{d_2}, q_{d_3}, q_{d_4})$. Then,

$$Q = [q_r, q_d]$$

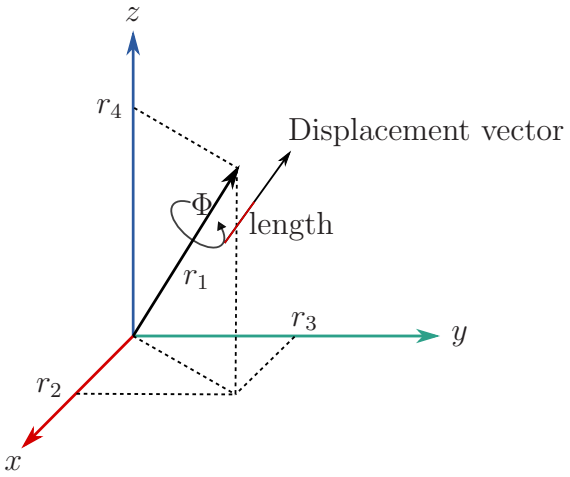


Figure 5.3: Dual Quaternionen

Properties

Dual quaternions are suitable for location describing of an object. Operations on dual quaternions also allow all needed transformations. The redundancy is low, as only 8 characteristics are required and there is no gimbal lock.

Weaknesses

There is the difficulty for the user to describe a position by specifying a dual quaternion and there are complex processing rules, for example for multiplication.

5.5 Dual-Quaternion Arithmetic Operations

The elementary arithmetic operations for use to use dual-quaternions are:

- Scaler multiplication: $sq = sq_r + sq_d\epsilon$

- Addition: $q_1 + q_2 = q_{r1} + q_{r2} + (q_{d1} + q_{d2})\epsilon$
- Multiplication: $q_1.q_2 = q_{r1}.q_{r2} + (q_{r1}.q_{d2} + q_{d1}.q_{r2})\epsilon$
- Conjugate: $q^\star = q_{r^\star} + q_{d^\star}\epsilon$
- Magnitude: $||q|| = qq^\star$
- Unit condition: $||q|| = 1, \qquad q_{r^\star}q_d + q_{d^\star}q_r = 0$

The unit dual-quaternion is our key concern as it can represent any rigid rotational and translational transformations.

Dual-Quaternions representation

For a rotation around $\vec{g}$ with angle ψ,

$$q_{rot} = [\cos(\frac{\phi}{2}), g_1 \sin(\frac{\phi}{2}), g_2 \sin(\frac{\phi}{2}), g_3 \sin(\frac{\phi}{2}), 0, 0, 0, 0]$$

For a transformation by $\vec{g}$ with no rotation,

$$q_{trans} = [1, 0, 0, 0, 0, \frac{g_1}{2}, \frac{g_2}{2}, \frac{g_3}{2}]$$

and finally,

$$q = q_{trans} \times q_{rot}$$

5.5.1 Gimbal lock

Gimbal lock is the loss of one degree of freedom in a three-dimensional, three-gimbal mechanism, that occurs when the axes of two of the three gimbals are driven into a parallel configuration, "locking" the system into rotation in a degenerate two-dimensional space.

The word lock is misleading: no gimbal is restrained. All three gimbals can still rotate freely about their respective axes of suspension. Nevertheless, because of the parallel orientation of two of the gimbals' axes there is no gimbal available to accommodate rotation about one axis.

6. Direct Kinematics - Exponential Coordinates and Screws

6.1 Exponential Coordinates

We have a Rotation axis $\vec{g}$, and a rotation angle θ. We want to Rotate a point $\vec{q}$ with a velocity of 1 around an axis $\vec{g}$:

$$\Rightarrow \dot{\vec{q}}(t) = \vec{g} \times \vec{q}(t) =: \hat{g}\vec{q}(t)$$

With:

$$\hat{g} = \begin{bmatrix} 0 & -g_3 & g_2 \\ g_3 & 0 & -g_1 \\ -g_2 & g_1 & 0 \end{bmatrix}$$

Angle θ can be rotate with integral equation:

$$\int_0^\theta g\vec{q}(t)dt = e^{g\theta}\vec{q}(0)$$

where $e^{g\theta}$ is exponential of a matrix:

$$e^{g\theta} = 1 + g\theta + \frac{(g\theta)^2}{2!} + \frac{(g\theta)^3}{3!} + ... = 1 + g\sin\theta + g^2(1 - \cos\theta)$$

6.2 Screws

6.2.1 Definitions

A *screw* $S = S(h, \theta, \vec{g}, \vec{P})$ is defined by a translation h, a twist angle θ, a normalized screw axis $\vec{g}$, and a location P.

Screw Types:

$p = \frac{h}{\theta}$ is called the *pitch* of screw $S = S(h, \theta, \vec{g}, \vec{P})$. If $p > 0$, S is called right handed, and if $p < 0$, S is called left handed.

If $\vec{P} = 0$, then we write $S = S(h, \theta, \vec{g}) = S = S(h, \theta, \vec{g}, 0)$ and S is called a *central screw*.

Chasles Theorem:

For all homogeneous Matrices

$$\begin{bmatrix} R & \vec{u} \\ 0 & 1 \end{bmatrix}$$

there exists $\vec{g}$ and θ, such that

$$\begin{bmatrix} R & \vec{u} \\ 0 & 1 \end{bmatrix}$$

can be describe as

$$\begin{bmatrix} R_{\vec{g}}(\theta) & \vec{g} \\ 0 & 1 \end{bmatrix}.$$

$\vec{g}$ is called the *Screw axis* and θ is calle the *twist angle*. See figures 6.1 and 6.2.

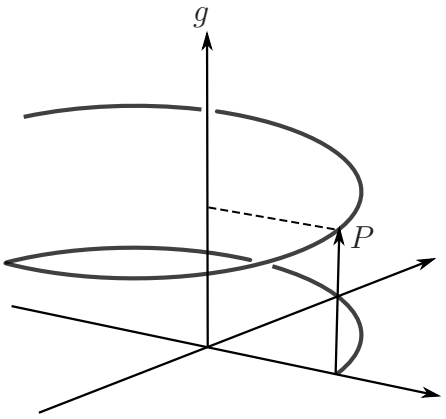


Figure 6.1: Translation along a line

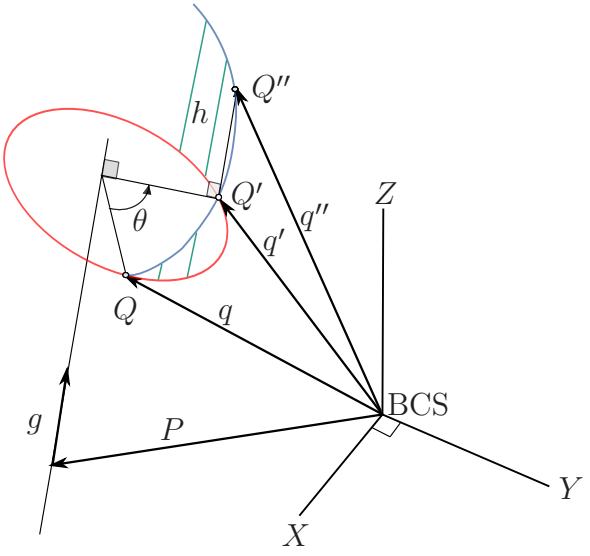


Figure 6.2: Screw motion of a rigid body

6.2.2 From central Screws to Homogenous Matrices

If we have central screw $S(h, \theta, \vec{g})$, then

$$R_{\vec{g}}(\theta) = \begin{bmatrix} g_1^2\eta\theta + C\theta & g_1g_2\eta\theta - g_3S\theta & g_1g_3\eta\theta + g_2S\theta \\ g_1g_2\eta\theta + g_3S\theta & g_2^2\eta\theta + C\theta & g_2g_3\eta\theta - g_1S\theta \\ g_1g_3\eta\theta - g_2S\theta & g_2g_3\eta\theta + g_1S\theta & g_3^2\eta\theta + C\theta \end{bmatrix}$$

and hence,

$$A_S(h,\theta,\vec{g}) = \begin{bmatrix} R_{\vec{g}}(\theta) & h\vec{g} \\ 0 & 1 \end{bmatrix}$$

In general, if we have a screw $S(h,\theta,\vec{g},\vec{P})$, then from screws to homogenous matrices formula would be:

$$A_S(h,\theta,\vec{g},\vec{P}) = \begin{bmatrix} C\theta\mathcal{I}_3 + \vec{g}\vec{g}^T\eta\theta + \hat{g}S\theta & ((\mathcal{I}_3 - \vec{g}\vec{g}^T)\eta\theta - \hat{g}S\theta)\hat{p} + h\vec{g} \\ 0 & 1 \end{bmatrix}$$

6.2.3 From Homogenous Matrices to Screws

By Rodrigues formula we get

$$\theta = \cos^{-1}(\frac{tr(R)-1}{2}) \in [0,\pi]$$

and,

$$\vec{g} = \frac{1}{2S\theta}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}$$

In this case, $\vec{P}$ is any point on the screw. Remember that h and $\vec{P}$ are still missing. So, our goal is to find $\vec{P} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix}$ and h, such that,

$$\begin{bmatrix} R & \vec{u} \\ 0 & 1 \end{bmatrix}\begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} R_{\vec{g}}(\theta) & h\vec{g} \\ 0 & 1 \end{bmatrix}\begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

$$\Leftrightarrow \begin{bmatrix} R - R_{\vec{g}}(\theta) & \vec{u} - h\vec{g} \\ 0 & 0 \end{bmatrix}\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\Leftrightarrow R - R_{\vec{g}}(\theta)\begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} + \vec{u} - h\vec{g} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\Leftrightarrow \vec{u} = h\vec{g} - (R - R_{\vec{g}}(\theta))\begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

w. l. o. g., $P_1 = 0$, since any point on the screw is fine.

$$\Rightarrow \vec{u} = \begin{bmatrix} \vec{g} & (R - R_{\vec{g}}(\theta))_2 & (R - R_{\vec{g}}(\theta))_3 \end{bmatrix} \begin{bmatrix} h \\ P_2 \\ P_3 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} h \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} \vec{g} & (R - R_{\vec{g}}(\theta))_2 & (R - R_{\vec{g}}(\theta))_3 \end{bmatrix}^{-1} \vec{u}$$

here, $(.)_i$ denotes the $i-th$ column.

6.2.4 Principle Screws

For a central Screw we have:

$$A_S(h, \phi, \vec{g}) = \begin{bmatrix} R_{\vec{g}}(\theta) & h\vec{g} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4}$$

If

$$\vec{g} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad i.e. \qquad \vec{g} = \vec{x}_{i-1}$$

then,

$$A(h, \phi, \vec{x}_{i-1}) = \begin{bmatrix} 1 & 0 & 0 & h \\ 0 & C\theta & -S\theta & 0 \\ 0 & S\theta & C\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Compound Screws
Similare for homogeneous matrices, screws can be combined:

$${}^{k-2}_{k}S(h, \phi, \vec{g}, \vec{P}) = {}^{k-2}_{k-1}S(h_1, \phi_1, \vec{g}_1, \vec{P}_1)\,{}^{k-1}_{k}S(h, 0, 0, 0)$$

General soloution is lengthy, but is not needed.

6.2.5 D-H with Screws

We have θ_i, d_i ,a_i and α_i from DH for ${}^{i-1}_{i}A$. Then,

$${}^{i-1}_{i}A = S(d_i, \theta_i, z_{i-1})S(a_i, \alpha_i, x_i) = \begin{bmatrix} C\theta_i & -S\theta_i C\theta_i & S\theta_i S\alpha_i & \alpha_i C\theta_i \\ S\theta_i & C\theta_i C\alpha_i & -C\theta_i S\theta_i & \alpha_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6.2.6 D-H with Screws Compound

From this, we can compute the compound $S(h, \phi, \vec{g}, \vec{P})$ such that:

$$\cos\phi = \frac{1}{2}(tr({}^{G}R_{B}) - 1) = \frac{1}{2}(\cos\theta_i + \cos\theta_i\cos\alpha_i + \cos\alpha_i - 1)$$

and,

$$\vec{g} = \frac{1}{2S\phi}\begin{bmatrix}\cos\alpha_i + \cos\theta_i\sin\alpha_i \\ \sin\theta_i\sin\alpha_i \\ \sin\theta_i + \cos\alpha_i\sin\theta_i\end{bmatrix}$$

Then screws with $x = 0$, $y = 0$ and $z = 0$ are defined as following:

Screw with x=0:

$$\begin{bmatrix}h \\ y \\ z\end{bmatrix} = \begin{bmatrix}g_1 & -r_{12} & -r_{13} \\ g_2 & 1 - r_{22} & -r_{23} \\ g_3 & -r_{32} & 1 - r_{33}\end{bmatrix}^{-1}\begin{bmatrix}r_{14} \\ r_{24} \\ r_{34}\end{bmatrix}$$
$$= \frac{1}{2S\phi}\begin{bmatrix}S\alpha_i + C\theta_iS\alpha_i & -S\theta_iC\alpha_i & S\theta_iS\alpha_i \\ S\theta_iS\alpha_i & 1 - C\theta_iC\alpha_i & -C\theta_iS\alpha_i \\ S\theta_i + C\alpha_iS\theta_i & S\alpha_i & C\alpha_i\end{bmatrix}^{-1}\begin{bmatrix}\alpha_iC\theta_i \\ \alpha_iS\theta_i \\ d\end{bmatrix}$$

Screw with y=0:

$$\begin{bmatrix}h \\ x \\ z\end{bmatrix} = \begin{bmatrix}g_1 & 1 - r_{11} & -r_{13} \\ g_2 & -r_{21} & -r_{23} \\ g_3 & -r_{31} & 1 - r_{33}\end{bmatrix}^{-1}\begin{bmatrix}r_{14} \\ r_{24} \\ r_{34}\end{bmatrix}$$
$$= \frac{1}{2S\phi}\begin{bmatrix}S\alpha_i + C\theta_iS\alpha_i & 1 - C\theta_i & S\theta_iS\alpha_i \\ S\theta_iS\alpha_i & S\theta_i & -C\theta_iS\alpha_i \\ S\theta_i + C\alpha_iS\theta_i & 0 & C\alpha_i\end{bmatrix}^{-1}\begin{bmatrix}\alpha_iC\theta_i \\ \alpha_iS\theta_i \\ d\end{bmatrix}$$

Screw with z=0:

$$\begin{bmatrix}h \\ x \\ y\end{bmatrix} = \begin{bmatrix}g_1 & 1 - r_{11} & -r_{12} \\ g_2 & -r_{21} & 1 - r_{22} \\ g_3 & -r_{31} & 1 - r_{32}\end{bmatrix}^{-1}\begin{bmatrix}r_{14} \\ r_{24} \\ r_{34}\end{bmatrix}$$
$$= \frac{1}{2S\phi}\begin{bmatrix}S\alpha_i + C\theta_iS\alpha_i & 1 - C\theta_i & -S\theta_iS\alpha_i \\ S\theta_iS\alpha_i & S\theta_i & 1 - C\theta_iS\alpha_i \\ S\theta_i + C\alpha_iS\theta_i & 0 & C\alpha_i\end{bmatrix}^{-1}\begin{bmatrix}\alpha_iC\theta_i \\ \alpha_iS\theta_i \\ d\end{bmatrix}$$

Direct Kinematics: Example 1

- $\phi_i = \theta_i$
- $\hat{g}_i = (0,0,1)^T$
- $h_i = 0$
- $\vec{P}_i = (\frac{a(2C\theta-1)}{4S\theta}, \frac{a(2C\theta-1)}{4C\theta-1}, 0)^T$

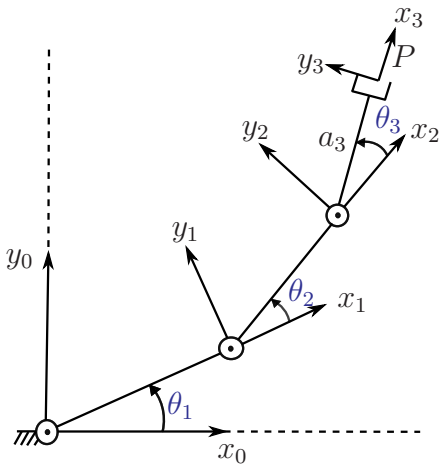


Figure 6.3: Direct Kinematics: Example 1

Joint	a_i	α_i	d_i	θ_i
1	a_1	0	0	θ_1
2	a_2	0	0	θ_2
3	a_3	0	0	θ_3

6.2.7 Exponential Representation of a Screw

Similar to Exponential coordinates For a rotation, a transformation can be rotated. See figure 6.2.

As in image 6.2,

$$Q'' = P + e^{\phi\hat{g}(g-P)} + h\vec{g} =: [T]$$

where,

$$[T] = \begin{bmatrix} e^{\phi\vec{g}} & (1-e^{\phi\vec{g}})P + h\vec{g} \\ 0 & 1 \end{bmatrix}$$

Exponential Representation of Central Screws

For central screws the same holds. And,

$$[T] = \begin{bmatrix} e^{\phi\hat{\vec{g}}} & h\vec{g} \\ 0 & 1 \end{bmatrix}$$

Note that,

$$e^{\begin{bmatrix} \phi\hat{g} & 0 \\ 0 & 0 \end{bmatrix}} = \begin{bmatrix} e^{\phi\hat{g}} & 0 \\ 0 & 1 \end{bmatrix}$$

and,

$$e^{\begin{bmatrix} 0 & h\vec{g} \\ 0 & 0 \end{bmatrix}} = \begin{bmatrix} \mathcal{I}_3 & h\vec{g} \\ 0 & 1 \end{bmatrix}$$

Therefore, exponential representation of D-H would be:

$$[T] = e^{\begin{bmatrix} \phi\hat{g} & h\vec{g} \\ 0 & 0 \end{bmatrix}} = e^{\xi(h,\phi,\vec{g})}$$

for D-H this means:

$$\begin{matrix} i-1 \\ i \end{matrix} A = S(d_i, \theta, z_{i-1}) S(a_i, \alpha_i, x_{i-1})$$

$$= e^{\xi(d_i,\theta_i,z_{i-1})} . e^{\xi(a_i,\alpha_i,x_i)}$$

Exponential Representation of Example 1:

$$A = e^{\xi(0,\theta_i,z_{i-1})} . e^{\xi(a_i,0,x_i)}$$

$$= e^{\begin{bmatrix} \theta_i\hat{z}_{i-1} & 0 \\ 0 & 0 \end{bmatrix}} \begin{bmatrix} 0 & a_i x_i \\ 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} e^{\theta_i\hat{z}_{i-1}} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathcal{I}_3 & a_i x_i \\ 0 & 1 \end{bmatrix}$$

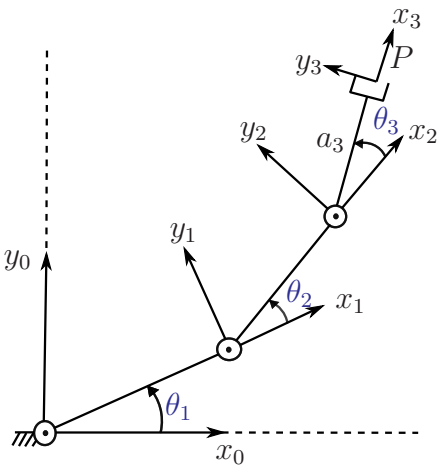


Figure 6.4: Direct Kinematics: Example 1-1

Joint	a_i	α_i	d_i	θ_i
1	a_1	0	0	θ_1
2	a_2	0	0	θ_2
3	a_3	0	0	θ_3

7. Inverse Kinematics

7.1 Inverse Kinematic Problem (IK)

From D-H-parameters and the positon of the gripper, one should calculate the joint angles → solve equation for θ

$$ {}^{BASIS}_{TCP}A = {}^{Basis}_{1}A(\theta_1) \cdot {}^{l}_{2}A(\theta_2) \cdots {}^{n-2}_{n-1}A(\theta_{n-1}) \cdot {}^{n-1}_{n}A(\theta_n) $$

This yields 12 equations with n unknowns. For Puma 260 there are 12 equations with 6 unknowns.

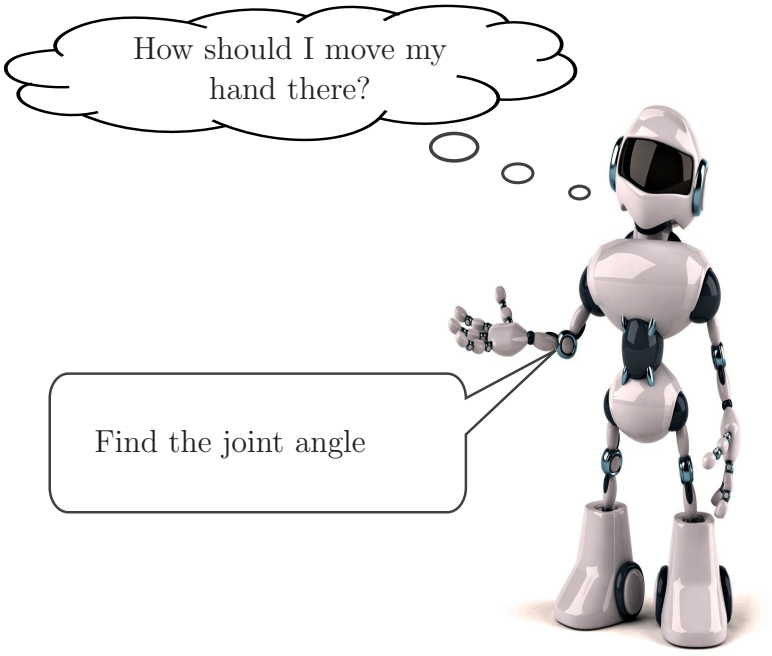


Figure 7.1: Inverse Kinematic Problem

7.1.1 IK-Problem

Acceptable Configurations: Not all mathematical solutions can be reached mechanically (Limitation of joint angles, Singular configuration, Endpoint does not belong to workspace, etc.)

Uniqueness: Multiple configurations (combinations of joint angles) result in the same position of the end effector, as in Figure 7.2. So the question is: How to choose a suitable solution?

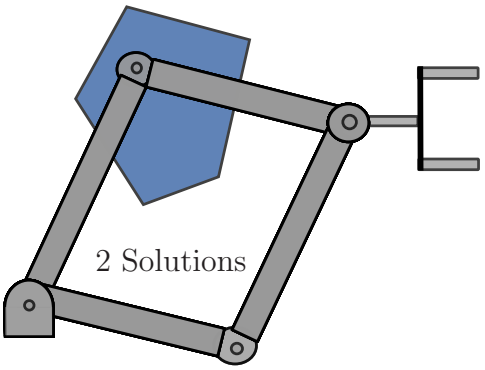


Figure 7.2: Uniqueness problem

Singular Configuration

Specifying constraints, e.g.:

$$min \quad f(\theta_{i-1}, \theta_{i+1}) = a({}^{t-1}\theta_{i-1} - {}^{t}\theta_{i-1})^2 + b({}^{t-1}\theta_{i+1} - {}^{t}\theta_{i+1})^2$$

with weights a and b. See figure 7.3.

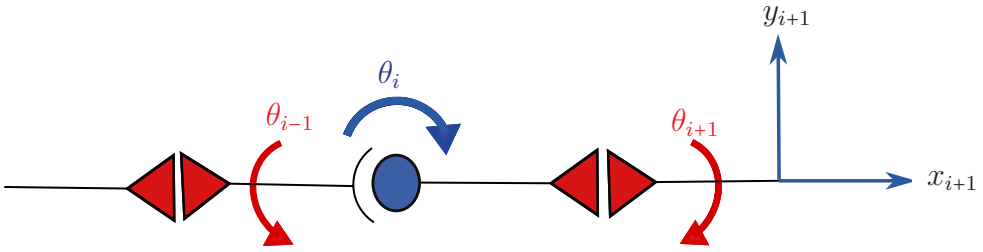


Figure 7.3: IK problems: Singular position

There is no generally applicable approach. In order to improve the velocity, the calculation of velocities must be fast. The methods for this are solutions in closed form, i.e. algebraic and geometric methods, and also numerical methods.

Example: Planar 2-Link Robot Arm

$$\begin{aligned} x &= l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \\ y &= l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \\ \phi &= \theta_1 + \theta_2 \end{aligned}$$

Abbreviations:

$$c_{12} = \cos(\theta_1 + \theta_2)$$
$$s_{12} = \sin(\theta_1 + \theta_2)$$

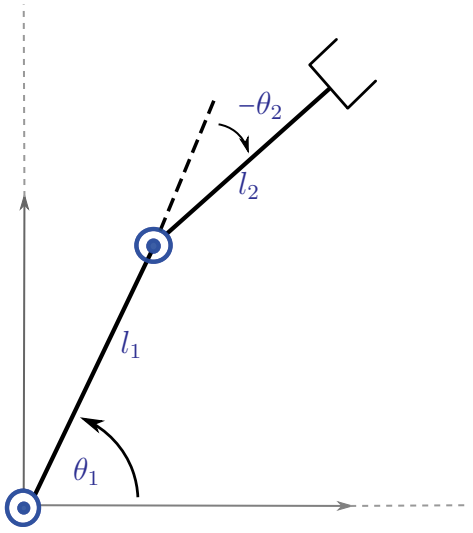


Figure 7.4: Example:Planar 2-Link Robot Arm

7.1.2 Algebraic Solution

Forward kinematics gives:

$${}^{0}_{2}T = \begin{bmatrix} c_{12} & -s_{12} & 0 & l_1 c_1 + l_2 c_{12} \\ s_{12} & c_{12} & 0 & l_1 s_1 + l_2 s_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Positioning and orientation of end effector is given by:

$${}^{BASIS}_{TCP}T = \begin{bmatrix} c\phi & -s\phi & 0 & x \\ s\phi & c\phi & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Comparison of coefficients gives:

$$\begin{aligned} c\phi &= c_{12} \\ s\phi &= s_{12} \\ x &= l_1 c_1 + l_2 c_{12} \\ y &= l_1 s_1 + l_2 s_{12} \end{aligned}$$

Sum of squares for the last two equations

$$x^2 + y^2 = l_1^2 + l_2^2 + 2l_1 l_2 c_2$$

therefore

$$c_2 = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1 l_2}$$

When does a solution exist and why two solutions are possible for θ_2?

Calculation of θ_1 with $k_1 = l_1 + l_2 c_2$ and $k_2 = l_2 s_2$:

$$\begin{aligned}
\cos(\theta_1 + \theta_2) = c_{12} &= c_1 c_2 - s_1 s_2 \\
\sin(\theta_1 + \theta_2) = s_{12} &= c_1 s_2 + c_2 s_1 \\
x &= k_1 c_1 - k_2 s_1 \\
y &= k_1 s_1 + k_2 c_1 \\
T_{0,2} &= \begin{bmatrix} c_{12} & -s_{12} & 0 & l_1 c_1 + l_2 c_{12} \\ s_{12} & c_{12} & 0 & l_1 s_1 + l_2 s_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

Let $r = +\sqrt{k_1^2 + k_2^2}$ and $\gamma = \text{ATAN2}(k_2, k_1)$. Constraint for angles: $\phi = \theta_1 + \theta_2$

$$\begin{aligned}
k_1 &= r\cos(\gamma) \\
k_2 &= r\sin(\gamma) \\
x/r &= \cos(\gamma)\cos(\theta_1) - \sin(\gamma)\sin(\theta_1) \\
y/r &= \cos(\gamma)\sin(\theta_1) + \sin(\gamma)\cos(\theta_1) \quad \text{or} \\
\cos(\gamma + \theta_1) &= x/r \qquad \text{and} \qquad \sin(\gamma + \theta_1) = y/r \\
\gamma + \theta_1 &= \text{ATAN2}(y/r, x/r) = \text{ATAN2}(y, x) \\
\rightarrow \theta_1 &= \text{ATAN2}(y, x) - \text{ATAN2}(k_2, k_1)
\end{aligned}$$

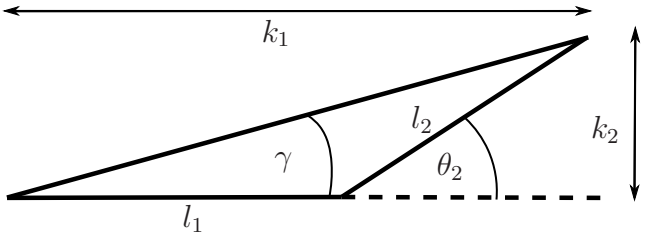


Figure 7.5: Example: Algebraic Solution

7.1.3 Geometric Solution

Law of cosine:

$$\begin{aligned} x^2 + y^2 &= l_1^2 + l_2^2 - 2l_1 l_2 \cos(180 - \theta_2) \\ \rightarrow \cos(\theta_2) &= \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1 l_2} \end{aligned}$$

$$\begin{aligned} l_2^2 &= x^2 + y^2 + l_1^2 - 2l_1\sqrt{(x^2 + y^2)}\cos(\psi) \\ \cos(\psi) &= (x^2 + y^2 + l_1^2 - l_2^2)/2l_1\sqrt{(x^2 + y^2)} \\ \theta_1 &= \beta \pm \psi \qquad \text{with} \qquad 0 \le \psi \le 180 \end{aligned}$$

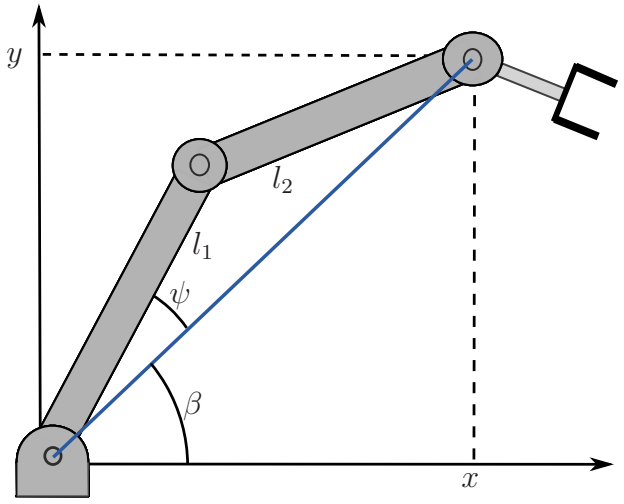


Figure 7.6: Geometric Solution

IK-Problem

Calculation of T_{TCP} by multiplying the homogeneous transformation matrices.

$$\overset{BASIS}{TCP}A = \overset{\text{BASIS}}{1}A(\theta_1) \cdot \overset{1}{2}A(\theta_2) \cdots \overset{n-2}{n-1}A(\theta_{n-1}) \cdot \overset{n-1}{n}A(\theta_n)$$

T_{tcp} is a homogeneous 4×4 matrix describing the desired position and orientation of the end effector.

$$\overset{BASIS}{TCP}T = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

7.1.4 Algorithms to Solve IK-Problems

Given: Transformation matrix (where e.g. $n = 6$)

Wanted:: Joint angles θ_1 to θ_n

1. Invert ${}^{0}_{1}A(\theta_i)$ and multiply (1) by ${}^{0}_{1}A^{-1}$ from both sides.
2. Find in the newly formed equation system one equation which has only one unknown and solve it.
3. Find equations in the equation system, which can be solved by substituting the angle we found in the last step
4. If there is no suitable equation invert the matrix ${}^{i}_{i+1}A(\theta_{i+1})$
5. Repeat steps 1–4 until all joint angles are found

Closed Form Solutions

By inverting transformation matrices and multiplying from the left or the right one creates new matrix equations which might give a closed form solution for some angles.

$$
\begin{aligned}
{}^{BASIS}_{1}A^{-1} \cdot {}^{BASIS}_{TCP}T &= {}^{1}_{2}A \cdot {}^{2}_{3}A \cdot {}^{3}_{4}A \cdot {}^{4}_{5}A \cdot {}^{5}_{6}A \\
{}^{1}_{2}A^{-1} \cdot {}^{BASIS}_{1}A^{-1} \cdot {}^{BASIS}_{TCP}T &= {}^{2}_{3}A \cdot {}^{3}_{4}A \cdot {}^{4}_{5}A \cdot {}^{5}_{6}A \\
{}^{2}_{3}A^{-1} \cdot {}^{1}_{2}A^{-1} \cdot {}^{BASIS}_{1}A^{-1} \cdot {}^{BASIS}_{TCP}T &= {}^{3}_{4}A \cdot {}^{4}_{5}A \cdot {}^{5}_{6}A \\
{}^{3}_{4}A^{-1} \cdot {}^{2}_{3}A^{-1} \cdot {}^{1}_{2}A^{-1} \cdot {}^{BASIS}_{1}A^{-1} \cdot {}^{BASIS}_{TCP}T &= {}^{4}_{5}A \cdot {}^{5}_{6}A \\
{}^{4}_{5}A^{-1} \cdot {}^{3}_{4}A^{-1} \cdot {}^{2}_{3}A^{-1} \cdot {}^{1}_{2}A^{-1} \cdot {}^{BASIS}_{1}A^{-1} \cdot {}^{Basis}_{TCP}T &= {}^{5}_{6}A
\end{aligned}
$$

$$
\begin{aligned}
{}^{BASIS}_{TCP}T \cdot {}^{5}_{6}A^{-1} &= {}^{BASIS}_{1}A \cdot {}^{1}_{2}A \cdot {}^{2}_{3}A \cdot {}^{3}_{4}A \cdot {}^{4}_{5}A \\
{}^{BASIS}_{TCP}T \cdot {}^{5}_{6}A^{-1} \cdot {}^{4}_{5}A^{-1} &= {}^{BASIS}_{1}A \cdot {}^{1}_{2}A \cdot {}^{2}_{3}A \cdot {}^{3}_{4}A \\
{}^{BASIS}_{TCP}T \cdot {}^{5}_{6}A^{-1} \cdot {}^{4}_{5}A^{-1} \cdot {}^{3}_{4}A^{-1} &= {}^{BASIS}_{1}A \cdot {}^{1}_{2}A \cdot {}^{2}_{3}A \\
{}^{B}_{TCP}T \cdot {}^{5}_{6}A^{-1} \cdot {}^{4}_{5}A^{-1} \cdot {}^{3}_{4}A^{-1} \cdot {}^{2}_{3}A^{-1} &= {}^{BASIS}_{1}A \cdot {}^{1}_{2}A \\
{}^{BASIS}_{TCP}T \cdot {}^{5}_{6}A^{-1} \cdot {}^{4}_{5}A^{-1} \cdot {}^{3}_{4}A^{-1} \cdot {}^{2}_{3}A^{-1} \cdot {}^{1}_{2}A^{-1} &= {}^{BASIS}_{1}A
\end{aligned}
$$

7.1.5 Numeric Methods

$$\vec{x}(t) = f(\vec{\theta}(t)) \Longrightarrow \frac{d\vec{x}(t)}{dt} = \dot{\vec{x}}(t) = J(\vec{\theta}).\dot{\vec{\theta}}(t)$$

$$J(\vec{\theta}) \in R^{n \times m} \qquad J_{ij} = \frac{df_i}{d\theta_j} \qquad 1 \le i \le m, 1 \le j \le n$$

m is cartesian degrees of freedom, and n is number of joints

Translation and angle velocities of the TCP (e.g. differential temporal change of the Euler-angles).

$$\dot{x}(t) = (\dot{p}_x, \dot{p}_y, \dot{p}_z, \dot{\alpha}, \dot{\beta}, \dot{\gamma})^T$$

Difference quotient instead of differential quotient:

$$\Delta\theta = J(\theta)^{-1}\Delta x$$

Idea:

- Calculate change in the description vector $\Delta\vec{x}$
- Calculate the necessary changes in the joint angles $\Delta\vec{\theta}$ using the inverse Jacobi matrix.
- Approximate solution, since when $\Delta\vec{x}$ changes to a Jacobi matrix is assumed for the corresponding $\Delta\vec{\theta}$ changes.
- After the calculation of $\Delta\vec{\theta}$ the Jacobi matrix is updated.
- The resulting error is iteratively reduced.
- If $n = m$ the manipulator is not redundant. ⇒ Jacobi matrix is square and can be inverted.
- If $n > m$ (underdetermined system) the manipulator is redundant.
  ⇒ Inverse of the Jacobi matrix does not exist.
  ⇒ Generalized inverse of the Jacobi matrix "Pseudoinverse".
- If $n < m$ (over-determined system) there is often no solution or only subspace.
  ⇒ The inverse of the Jacobian matrix does not exist.
  ⇒ Generalized inverse of the Jacobi matrix "Pseudoinverse".

In principle, these approaches can be used for all types of robots with arbitrary degrees of freedom. Further problems are the Susceptible for singularities, long runtimes, or an arbitrary solution will be found.

Example: Planar 2-Link Robot Arm

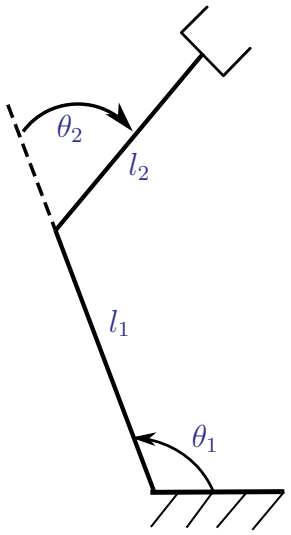


Figure 7.7: Example: Planar 2-Link Robot Arm 2

$$
\begin{aligned}
x &= l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2) \\
y &= l_1\sin(\theta_1) + l_2\sin(\theta_1+\theta_2) \\
\begin{pmatrix}\dot{x}\\ \dot{y}\end{pmatrix} &= J(\vec{\theta})\dot{\vec{\theta}} = J(\vec{\theta})\cdot\begin{pmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{pmatrix} \\
&= \begin{pmatrix}-l_1\sin(\theta_1) - l_2\sin(\theta_1+\theta_2) & -l_2\sin(\theta_1+\theta_2)\\ l_1\cos(\theta_1) + l_2\cos(\theta_1+\theta_2) & l_2\cos(\theta_1+\theta_2)\end{pmatrix}\cdot\begin{pmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{pmatrix}
\end{aligned}
$$

The Jacobi-matrix needs to be inverted.

$$
\begin{pmatrix}\Delta\theta_1\\ \Delta\theta_2\end{pmatrix} = \underbrace{\frac{1}{l_1 l_2\sin(\theta_2)}\cdot\begin{pmatrix}l_2 c_{12} & l_2 s_{12}\\ -l_1 c_{12} - l_1 c_1 & -l_1 s_{12} - l_1 s_1\end{pmatrix}}_{J(\theta)^{-1}}\cdot\begin{pmatrix}\Delta x\\ \Delta y\end{pmatrix}
$$

If $\theta_2 = \theta, \pm 180$, then $J(\theta)$ is singular!
→ All singular configurations are at the boundaries of the working space.
Abbreviations:

$$
\begin{aligned}
c_{12} &= \cos(\theta_1+\theta_2) \\
s_{12} &= \sin(\theta_1+\theta_2) \\
c_i &= \cos(\theta_i) \\
s_i &= \sin(\theta_i)
\end{aligned}
$$

Optimization

1. Overdetermined system ($n < m$):
   - Not enough degrees of freedom to reach pose

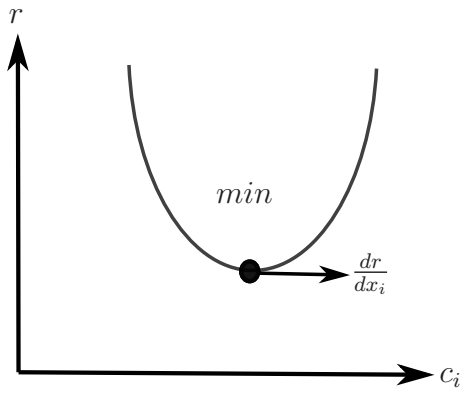
- Approximate solutions yield $min \quad \|J\Delta\theta - \Delta x\|^2$
- Via pseudo-inverse $J^+ := (J^T J)^{-1} J^T$ one gets $\Delta\theta = J^+ \Delta x$
- With difference vector $\Delta\vec{x} = \vec{x}_{target} - \vec{x}_{current} = (\Delta p_x, \Delta p_y, \Delta p_z, \Delta\alpha, \Delta\beta, \Delta\gamma)^T$

Minimization of Mean Squared Error (Gauß)

- Overdetermined system $J \cdot \Delta x = \Delta\theta$ where $i = 1, \ldots, m > n$.

$$\sum_{k=1}^{n} j_{ik} \Delta x_k = \theta_i$$

- Minimize $\quad min \quad r = \sum_{i=1}^{m} (\sum_{k=1}^{n} j_{ik} \Delta x_k - \Delta\theta_i)^2$.
- (Necessary) Condition is $\nabla r = 0$, i.e. $\frac{dr}{dx_i} \stackrel{!}{=} 0 \qquad i = 1, \ldots, n$.



The last necessary condition implies:

$$\frac{dr}{dx_i} \stackrel{!}{=} 0 \qquad i = 1, \ldots, n \tag{7.1}$$

$$\frac{dr}{dx_i} = \sum_{l=1}^{m} \frac{d}{dx_i} (\sum_{k=1}^{n} j_{lk} \Delta x_k - \Delta\theta_l)^2 \qquad \text{Chain rule} \tag{7.2}$$

$$= \sum_{l=1}^{m} 2 (\sum_{k=1}^{n} j_{lk} \Delta x_k - \Delta\theta_l) \frac{d}{dx_i} \sum_{k=1}^{n} j_{lk} \Delta x_k \tag{7.3}$$

$$= 2 \sum_{l=1}^{m} (\sum_{k=1}^{n} j_{lk} \Delta x_k - \Delta\theta_l) j_{li} \tag{7.4}$$

$$= 2 \sum_{l=1}^{m} \sum_{k=1}^{n} j_{li} j_{lk} \Delta x_k - 2 \sum_{l=1}^{m} j_{li} \Delta\theta_1 \stackrel{!}{=} 0. \tag{7.5}$$

System can now be written as $J^T \cdot J \cdot \Delta\vec{x} = J^T \cdot \Delta\theta$ or $\vec{x} = (J^T \cdot J)^{-1} \cdot J^T \cdot \theta$

2. Underdetermined system $(n > m)$

   Usually there are too many degrees of freedom. There are additional conditions (e.g. most natural joint positions $\Delta\theta_i'$). The Constrained optimization problem is:

$$min \qquad h := \sum_{i=1}^{n} w_i (\Delta\theta_i - \Delta\theta_i')^2$$

under the secondary condition:

$$J\Delta\vec{\theta} = \Delta\vec{x}$$

Solution via Lagrange multiplier:

$$min \qquad r := h + \lambda^T(J\Delta\vec{\theta} - \Delta\vec{x})$$

Lagrange Multiplier

Idea:

Find extrema of a function $h(x_1, \ldots, x_n)$ under the constrained to $g(x_1, \ldots, x_n) = c$.
Solution:

$$g = c \wedge \nabla h = \lambda \nabla g$$

See figure 7.8.

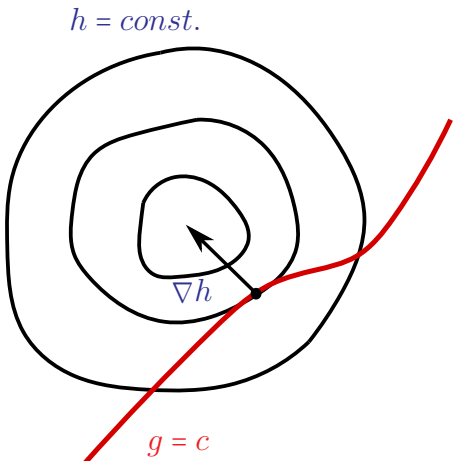


Figure 7.8: Lagrange Multiplier

Given:
m *strict* constraints (end effector, comparison $g = c$) of the form

$$\begin{array}{rcl} J \cdot \Delta\vec{x} & = & \Delta\vec{\theta} \\ m\overbrace{\boxed{\text{J}}}^{n}|_n\, \Delta x & = & |_m\, \Delta\theta \end{array}$$

l *soft* constraints (natural joints, $h \to min$)

$$\begin{array}{rcl} A \cdot \Delta\vec{x} & = & \vec{q} \\ l\overbrace{\boxed{\text{A}}}^{n}|_n\, \Delta x & = & |_l\, q \end{array}$$

With unknowns x_i where $i = 1, \ldots, n$ and $n > m$.

Approach:
The aim is solving optimization problem:

$$min \quad r = \sum_{k=1}^{l}(\sum_{r=1}^{n} a_{kr}\Delta x_r - q_k)^2 + \sum_{k=1}^{m}\lambda_k(\sum_{r=1}^{n} j_{kr}\Delta x_r - \Delta\theta_k)$$

with r depends on x_i, $i = 1, \ldots, n$ and λ_k, $k = 1 \ldots, m$.

$$\frac{dr}{dx_i} = 2\sum_{k=1}^{l}(\sum_{r=1}^{n} a_{kr}\Delta x_r - q_k)a_{ki} + \sum_{k=1}^{m}\lambda_k j_{ki} \stackrel{!}{=} 0 \qquad \text{(Soft constraits, } \nabla h = \lambda \cdot \nabla g\text{)}$$

$$\frac{dr}{d\lambda_i} = \sum_{r=1}^{n} j_{ir}\Delta x_r - \Delta\theta_i \stackrel{!}{=} 0 \qquad \text{(Strict constraint } g = c\text{)}$$

In matrix form:

$$\begin{pmatrix} 2A^TA & J^T \\ J & 0 \end{pmatrix}\begin{pmatrix} \Delta\vec{x} \\ \lambda \end{pmatrix} = \begin{pmatrix} 2A^T\vec{q} \\ \Delta\theta \end{pmatrix}$$

Note that the solution only consists of x_i and Lagrange multipliers λ_k are just used in order to find the solution.

Algorithm

1. Initial values for Θ and translation v
2. Determine $J(\Theta)$
3. Eliminate redundant parameters as necessary (singularities)
4. Solve for $\Delta\Theta$ (optimization if necessary with constraints)
5. Update Θ and v
6. If solution not yet reached go back to (2)

Problems

Singular situations need to be detected and $\Delta\Theta$ is restricted. See figures 7.9 and 7.10.

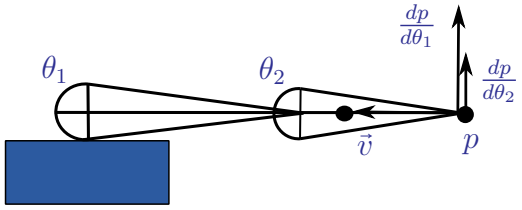


Figure 7.9: Blocked Manipulator

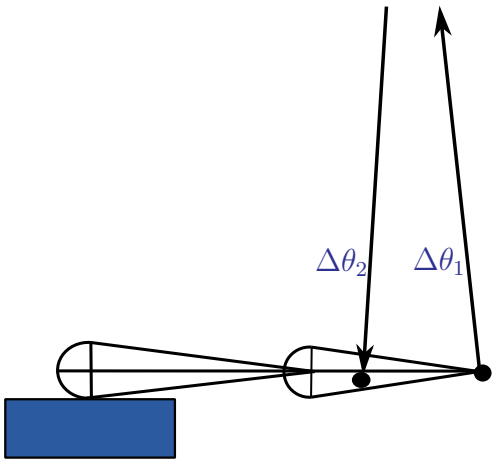


Figure 7.10: Surrounding of Singularity

7.2 Direct and Invers Kinematic

Direct Kinematic

$$f : R^n \to R^m \qquad \vec{x} = f(\vec{\theta})$$

Invers Kinematic

$$f^{-1} R^m \to R^n \qquad \vec{\theta} = f^{-1}(\vec{x})$$

- There exists a unique solution
- There exists a finite set of solutions
- There exists an infinite set of solutions
- There exists no solution

7.2.1 IK-Problems

	General Approaches	**Special Approaches**
Procedure	Iteration, general solving approach for equation systems	based on trigonometric relations
Pros	General	Fast
Cons	Computationally expensive, slow	Only for special robot setups

8. Velocity

This chapter contains speed analysis of the parts of a manipulator when the manipulated variable of the joints is changed. Then the conversion of speeds to other coordinate systems will be discussed, when coordinate systems are often referred to as frames. Calculation of velocity of a part/component as a superposition of translational and rotational velocity is the next part of the chapter. Chapter will end with relationship between joint and Cartesian speed of the end effector (Jacobian matrix), and investigation of forces and moments with a rigid kinematic chain.

8.1 Velocity Vector

Velocity Vectors are free vector, i.e. no starting point, only magnitude and direction are important. It means only rotation is considered. The velocity vector is the derivative of a position vector with respect to time:

$$^{B}\vec{v}_q = \frac{d}{dt}\ ^{B}\vec{q}$$

The conversion of a speed vector into a rotated coordinate system works as follows:

$$^{A}\vec{v}_q = \ {}^{A}_{B}R \cdot\ ^{B}\vec{v}_q$$

Remember that only the rotation matrix is considered here, and not the displacement vector.

8.1.1 Linear Velocity

The origin OB of the system B moves with a linear velocity $^{A}\vec{v}_{OB}$ relative to system A. Point $^{B}\vec{q}$ represented in the system B moves with a linear velocity $^{B}\vec{v}_q$. System B was created from system A by rotation ${}^{A}_{B}R$. Linear velocity of the point $^{B}\vec{q}$ related to system A is:

$$^{A}\vec{v}_q = ^{A}\ \vec{v}_{OB} + {}^{A}_{B}\ R \cdot ^{B}\ \vec{v}_q \tag{8.1}$$

8.1.2 Rotational Velocity

System A and system B share a common origin. The linear velocity between the two systems related to the origin is 0:

$$^{A}\vec{v}_{OB} = 0$$

$^{B}\vec{q}$ is represented in system B, and system B rotates around an axis through the common origin of system A and B at a rotation speed $^{A}\vec{\omega}_{B}$. See figure 8.1

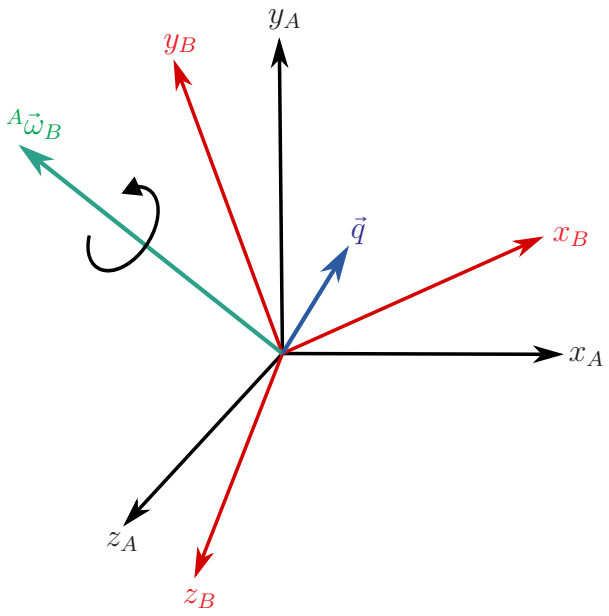


Figure 8.1: Rotational Velocity

The speed of the point $\vec{q}$ is:

$$^{A}\vec{v}_{q} = ^{A}\vec{\omega}_{B} \times ^{A}\vec{q}$$

Considering the linear velocity, the formula changes to:

$$^{A}\vec{v}_{q} = ^{A}_{B}R \cdot ^{B}\vec{v}_{q} + ^{A}\vec{\omega}_{B} \times ^{A}_{B}R \cdot ^{B}\vec{q} \tag{8.2}$$

So, linear and rotational velocity is:

$$^{A}\vec{v}_{q} = ^{A}\vec{v}_{OB} + ^{A}_{B}R \cdot ^{B}\vec{v}_{q} + ^{A}\vec{\omega}_{B} \times ^{A}_{B}R \cdot ^{B}\vec{q}$$

See figure 8.2.

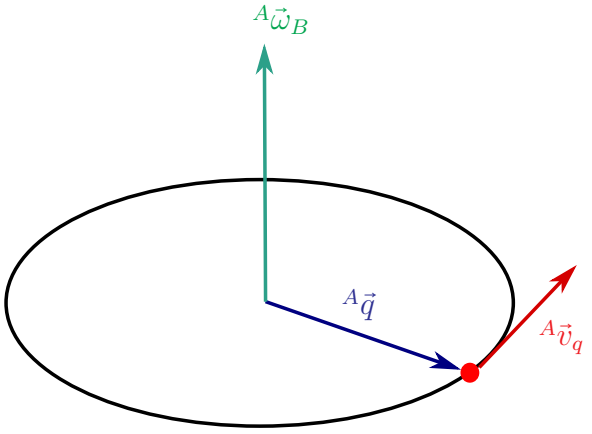


Figure 8.2: Rotational Velocity: Figure 2

8.1.3 Point Velocity in Another Reference System

$$^A\vec{v}_q = {}^A\vec{v}_{OB} + {}^A_B R \cdot {}^B\vec{v}_q + {}^A\vec{\omega}_B \times {}^A_B R \cdot {}^B\vec{q}$$

When:

$^A\vec{v}_{OB}$	Translational velocity of origin OB in system A
${}^A_B R \cdot {}^B\vec{v}_q$	Translational velocity of the point $^B\vec{q}$ in the system B transformed to the reference system A
${}^A\vec{\omega}_B \times {}^A_B R \cdot {}^B\vec{q}$	Translational point velocity due to the rotation of the system B compared to system A

8.2 Velocity of the Robot Parts

Velocity of the end effector of a robot with n joints is calculated from the kinematic structure and all the members involved in the movement. Velocity of a part consists of the velocity of its fixed coordinate system and rotational and translational velocity of the part. Velocity of the end effector in the base coordinate system is determined by successive calculation of the velocities of the parts from the base. Velocity of the part $i+1$ is the sum of the velocity of member i and the component resulting from relative motion between i and $i+1$. Pay attention that both summands must be in the same coordinate system. See figure 8.3.

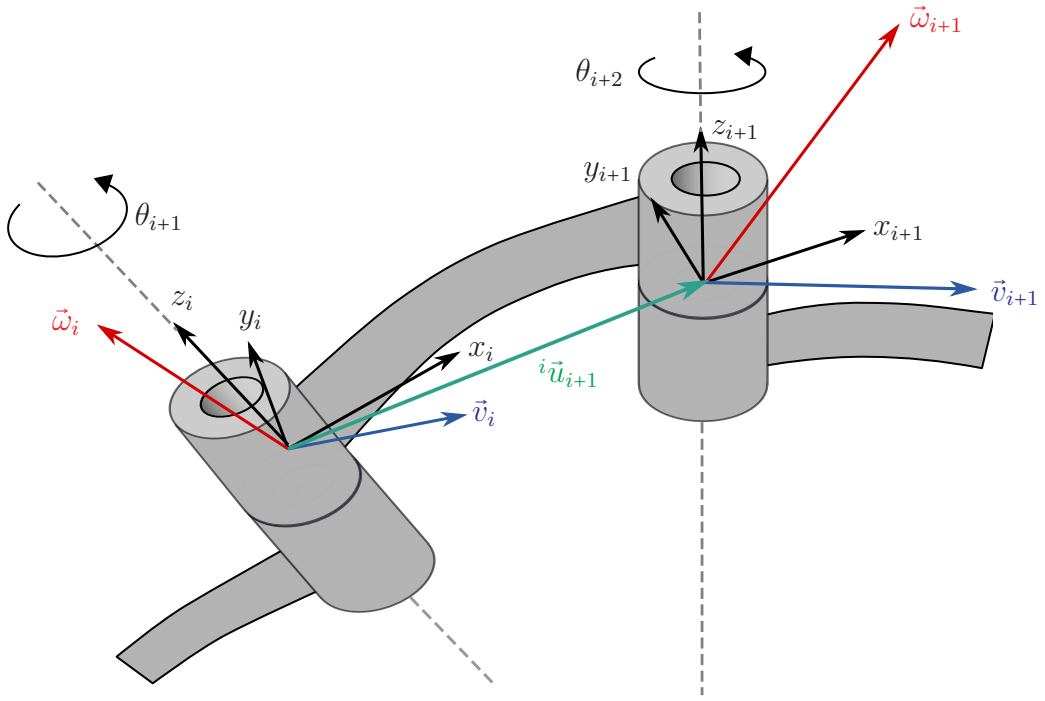


Figure 8.3: Coordinate System Identifier

8.2.1 Rotational Velocity at Rotational Joints

Let joint $i+1$ be a rotational joint with degree of freedom θ_{i+1}. Furthermore, let ${}^i\vec{\omega}_i$ be the rotational velocity of the part i. Then, the rotational velocity of the link $i+1$ is the rotational velocity of the part i, plus the component by rotation of the joint $i+1$:

$$ {}^i\vec{\omega}_{i+1} = {}^i\vec{\omega}_i + \dot{\theta}_{i+1} \cdot {}^i\vec{e}_{z_i} $$

with: $\dot{\theta}_{i+1} \cdot {}^i\vec{e}_{z_i} = (0 \quad 0 \quad \dot{\theta}_{i+1})^T$.
Transformation of ${}^i\vec{\omega}_{i+1}$ in the system $i+1$ by multiplying with ${}^{i+1}_{i}R$:

$$ {}^{i+1}\vec{\omega}_{i+1} = {}^{i+1}_{i}R \cdot ({}^i\vec{\omega}_i + \dot{\theta}_{i+1} \cdot {}^i\vec{e}_{z_i}) \tag{8.3} $$

For the translational speed of the origin of coordinate system $i+1$ represented in system i it holds:

$$ \vec{v}_{i+1} = \vec{v}_i + \vec{\omega}_{i+1} \times \vec{u}_{i+1} $$

and is represented in system $i+1$ with:

$$ {}^{i+1}\vec{v}_{i+1} = {}^{i+1}_{i}R({}^i\vec{v}_i + {}^i\vec{\omega}_{i+1} \times {}^i\vec{u}_{i+1}) $$

Velocity of Linear Joints

Let joint i be a translational joint with degree of freedom d_i. Then the followings hold:

- For rotational velocity:

$$ {}^{i+1}\vec{\omega}_{i+1} = {}^{i+1}_{i}R \cdot {}^i\vec{\omega}_i $$

- For translational velocity:

$$ {}^{i+1}\vec{v}_{i+1} = {}^{i+1}_{i}R \cdot ({}^{i}\vec{v}_i + {}^{i}\vec{\omega}_i \times {}^{i}\vec{u}_{i+1} + \dot{d}_{i+1} {}^{i}\vec{e}_{z_i}) \tag{8.4} $$

8.2.2 Example: Planar Robot Arm

Calculation of the rotation matrices required for the velocity ${}^{i+1}_{i}R = {}^{i}_{i+1}R^T$.
Rotations and translations separated.

$$ \begin{aligned} {}^{0}_{1}A &= T_{z_0}(0) \cdot R_{z_0}(\theta_1) \cdot R_{x_1}(0°) \cdot T_{x_1}(a_1) \\ &= \begin{bmatrix} c_1 & -s_1 & 0 & c_1 a_1 \\ s_1 & c_1 & 0 & s_1 a_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ {}^{1}_{2}A &= T_{z_1}(0) \cdot R_{z_1}(\theta_2) \cdot R_{x_2}(0°) \cdot T_{x_2}(a_2) \\ &= \begin{bmatrix} c_2 & -s_2 & 0 & c_2 a_2 \\ s_2 & c_2 & 0 & s_2 a_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} $$

Specification ${}^{0}v_0 = \vec{0}$, ${}^{0}\omega_0 = 0$ and derive by equations 8.3 and 8.4 and application of the derivation:

$$ \begin{aligned} {}^{0}\vec{\omega}_1 &= {}^{0}\vec{\omega}_0 + \dot{\theta}_1 {}^{0}\vec{e}_{z_0} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} \\ {}^{1}\vec{\omega}_1 &= {}^{1}_{0}R \cdot {}^{0}\vec{\omega}_1 = \begin{bmatrix} c_1 & s_1 & 0 \\ -s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} \end{aligned} $$

$$ \begin{aligned} {}^{1}\vec{v}_1 &= {}^{1}_{0}R({}^{0}\vec{v}_0 + {}^{0}\vec{\omega}_1 \times {}^{0}\vec{u}_1) \\ &= \begin{bmatrix} c_1 & s_1 & 0 \\ -s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \left[\begin{pmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{pmatrix} \times \begin{pmatrix} c_1 a_1 \\ s_1 a_1 \\ 0 \end{pmatrix} \right] \\ &= \begin{bmatrix} 0 \\ a_1 \dot{\theta}_1 \\ 0 \end{bmatrix} \end{aligned} $$

$$
\begin{aligned}
{}^1\vec{\omega}_2 &= {}^1\vec{\omega}_1 + \dot{\theta}_2\ {}^1\vec{e}_{z_1} \\
&= \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 \end{bmatrix} \\
{}^2\vec{\omega}_2 &= {}^2_1R\ {}^1\vec{\omega}_2 = \begin{bmatrix} c_2 & s_2 & 0 \\ -s_2 & c_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
{}^2\vec{v}_2 &= {}^2_1R({}^1\vec{v}_1 + {}^1\omega_2 \times {}^1\vec{u}_2) \\
&= {}^2_1R \cdot \left[\begin{pmatrix} 0 \\ a_1\dot{\theta}_1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 \end{pmatrix} \times \begin{pmatrix} c_2a_2 \\ s_2a_2 \\ 0 \end{pmatrix} \right] \\
&= \begin{bmatrix} c_2 & s_2 & 0 \\ -s_2 & c_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -s_2a_2(\dot{\theta}_1 + \dot{\theta}_2) \\ a_1\dot{\theta}_1 + c_2a_2(\dot{\theta}_1 + \dot{\theta}_2) \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} s_2a_1\dot{\theta}_1 \\ a_1c_2\dot{\theta}_1 + a_2(\dot{\theta}_1 + \dot{\theta}_2) \\ 0 \end{bmatrix}
\end{aligned}
$$

TCP linear velocity with respect to the base coordinate system:

$$
\begin{aligned}
{}^0_2R &= {}^0_1R \cdot {}^1_2R = \begin{bmatrix} c_{12} & -s_{12} & 0 \\ s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
{}^0\vec{v}_2 &= {}^0_2R\ {}^2\vec{v}_2 = \begin{bmatrix} -s_1a_1\dot{\theta}_1 - s_{12}a_2(\dot{\theta}_1 + \dot{\theta}_2) \\ c_1a_1\dot{\theta}_1 + c_{12}a_2(\dot{\theta}_1 + \dot{\theta}_2) \\ 0 \end{bmatrix} \\
{}^0\vec{\omega}_2 &= {}^0_2R\ {}^2\vec{\omega}_2 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 \end{bmatrix}
\end{aligned}
$$

See figure 8.4.

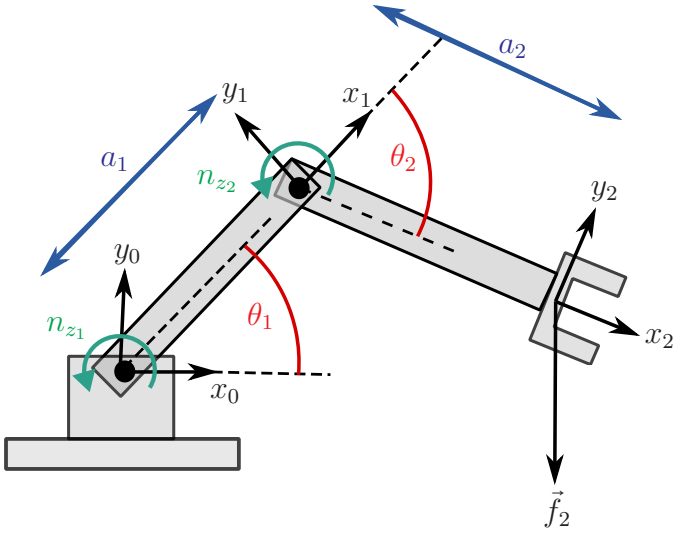


Figure 8.4: Example: Planar Robot Arm

8.3 Application of Jacobian Matrix

Let $\vec{y} = f(\vec{x}) \quad \vec{x} \in \mathbb{R}^m, \vec{y} \in \mathbb{R}^n$. Then, total differential of y:

$$\begin{aligned} y_1 &= f_1(x_1, x_2, \ldots, x_m) \\ y_2 &= f_2(x_1, x_2, \ldots, x_m) \\ &\vdots \\ y_n &= f_n(x_1, x_2, \ldots, x_m) \end{aligned} \qquad \begin{aligned} dy_1 &= \tfrac{df_1}{dx_1}dx_1 + \tfrac{df_1}{dx_2}dx_2 + \cdots + \tfrac{df_1}{dx_n}dx_m \\ dy_2 &= \tfrac{df_2}{dx_1}dx_1 + \tfrac{df_2}{dx_2}dx_2 + \cdots + \tfrac{df_2}{dx_n}dx_m \\ &\vdots \\ dy_n &= \tfrac{df_n}{dx_1}dx_1 + \tfrac{df_n}{dx_2}dx_2 + \cdots + \tfrac{df_n}{dx_n}dx_m \end{aligned}$$

In vector notation:

$$\begin{bmatrix} dy_1 \\ dy_2 \\ \vdots \\ dy_n \end{bmatrix} = \begin{bmatrix} \frac{df_1}{dx_1} & \frac{df_1}{dx_2} & \cdots & \frac{df_1}{dx_m} \\ \frac{df_2}{dx_1} & \frac{df_2}{dx_2} & \cdots & \frac{df_2}{dx_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{df_n}{dx_1} & \frac{df_n}{dx_2} & \cdots & \frac{df_n}{dx_m} \end{bmatrix} \begin{bmatrix} dx_1 \\ dx_2 \\ \vdots \\ dx_m \end{bmatrix} = J(\vec{x})d\vec{x}$$

Or $d\vec{y} = df(\vec{x}) = \frac{df(\vec{x})}{d\vec{x}}d\vec{x} = J(\vec{x})d\vec{x}$ with Jacobian Matrix $J(\vec{x}) = \frac{df(\vec{x})}{d\vec{x}}$

Derivation of the function $f(x)$ w.r.t. time yields:

$$\frac{d\vec{y}}{dt} = \frac{df(\vec{x})}{dt} = J(\vec{x})\frac{d\vec{x}}{dt}$$

or

$$\dot{\vec{y}} = J(\vec{x})\dot{\vec{x}}$$

In robotics, the Jacobian matrix describes the relationship between the effector velocity $\dot{\vec{y}}$ and the joint velocities $\dot{\vec{\theta}}$.

$\dot{\vec{y}} = J(\theta)\dot{\theta}$ with vector notation $\dot{\vec{y}} = (\dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\gamma})^T$

The number of columns m is equal to the degrees of freedom of movement/joint, and the number of rows n is equal to the degrees of freedom in cartesian space.

The transformation of a square 6×6 Jacobian matrix into another coordinate system looks like this:

$$ {}^0J(\vec{\theta}) = \underbrace{\begin{bmatrix} {}^0_1R & 0 \\ 0 & {}^0_1R \end{bmatrix}}_{6\times 6} \cdot {}^1J(\vec{\theta}) $$

Rest of the procedure:

- Determine ${}^m\vec{v}_m$ and ${}^m\vec{\omega}_m$ as shown
- Transform with the above equation in ${}^0\vec{v}_m$ and ${}^0\vec{\omega}_m$

Using ${}^0\vec{v}_2$ from the example above:

$$ \begin{aligned} \dot{\vec{y}} = {}^0\vec{v}_2 &= \begin{bmatrix} -s_1a_1\dot{\theta}_1 - s_{12}a_2(\dot{\theta}_1+\dot{\theta}_2) \\ c_1a_1\dot{\theta}_1 + c_{12}a_2(\dot{\theta}_1+\dot{\theta}_2) \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} -s_1a_1 - s_{12}a_2 & -s_{12}a_2 \\ c_1a_1 + c_{12}a_2 & c_{12}a_2 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_1+\dot{\theta}_2 \end{bmatrix} \\ \text{with} \quad J(\vec{\theta}) &= \begin{bmatrix} -s_1a_1 - s_{12}a_2 & -s_{12}a_2 \\ c_1a_1 + c_{12}a_2 & c_{12}a_2 \\ 0 & 0 \end{bmatrix} \end{aligned} $$

Considering the angular velocity:

$$ \dot{\vec{y}} = \begin{bmatrix} {}^0\vec{v}_2 \\ {}^0\vec{\omega}_2 \end{bmatrix} = \begin{bmatrix} {}^0v_{2_x} \\ {}^0v_{2_y} \\ {}^0v_{2_z} \\ {}^0\omega_{2_x} \\ {}^0\omega_{2_y} \\ {}^0\omega_{2_z} \end{bmatrix} = \begin{bmatrix} -s_1a_1 - s_{12}a_2 & -s_{12}a_2 \\ c_1a_1 + c_{12}a_2 & c_{12}a_2 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} $$

Further possibility for the calculation of the Jacobian matrix is the derivation of the forward kinematics.

8.3.1 Inverse Jacobian Matrix

Calculation of the joint angular velocities from cartesian velocities with the inverse Jacobian matrix:

$$\dot{\vec{\theta}} = J(\vec{\theta})^{-1}\dot{\vec{y}} \quad \text{Solution, if} \quad \det(J) \neq 0$$

If this is not square, the Cartesian degrees of freedom are greater than the joint degrees of freedom. Then the following applies:

1. Elimination of linear dependent lines in J
   $\rightarrow$ Invertible matrix
2. Least-square-method as an approximation
   $\dot{\vec{\theta}} = (J^T J)^{-1} J^T \dot{\vec{y}}$

If it is not square then Joint degrees of freedom are greater than cartesian degrees of freedom. So, there are a lot of solutions:

1. Block degrees of freedom of movement so that J square
2. Introduce constraints (collision avoidance)

8.3.2 Singularities

Robot configuration often with singular Jacobian matrices, thus losing cartesian degrees of freedom. There are two types of singularities:

1. At the edge of the working space
2. Inside the workspace

For example, the typical industrial robot where $\theta_5 = 0$, θ_4 and θ_6 act in the same direction, i.e. one degree of freedom is lost. See figure 8.5.

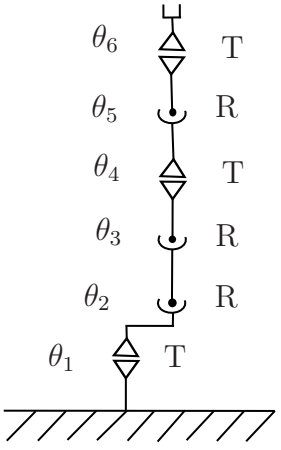


Figure 8.5: Singularities

Attention: In the vicinity of singularities very big joint velocities can result from small Cartesian velocities.

Example: Planar Robot

Singular position of the planar robot

Jacobian matrix is:	$J(\theta_1, \theta_2)$	$=$	$\begin{bmatrix} -s_1a_1 - s_{12}a_2 & -s_{12}a_2 \\ c_1a_1 + c_{12}a_2 & c_{12}a_2 \end{bmatrix}$
Associated determinant:	$\det(J)$	$=$	$a_1a_2\sin(\theta_2)$
Singularity (det = 0):	$a_1a_2\sin(\theta_2)$	$=$	0
	$\rightarrow \theta_2$	$=$	0 and $\theta_2 = \pi$
Relevant for practice:	θ_2	$=$	0
i.e. robotic arm fully extended			
(singularity at the edge of the workspace)			

Inverse Jacobian matrix is:

$$J^{-1}(\vec{\theta}) = \frac{1}{a_1a_2s_2}\begin{bmatrix} a_2c_{12} & a_2s_{12} \\ -a_1c_1 - a_2c_{12} & -a_1s_1 - a_2s_{12} \end{bmatrix}$$

For $\theta_2 = 0$, $\sin(\theta_2) = 0$. So, $\dot{\theta}_1 \quad \text{and} \quad \dot{\theta}_2 \rightarrow \infty$
See figure 8.6

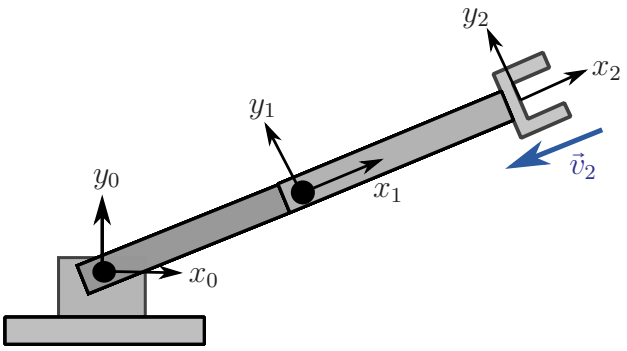


Figure 8.6: Singularities: Example Planar Robot

8.4 Velocity in Wheel-driven Robotic Systems

In the following, the stepwise calculation of the linear and the rotational velocity will be applied to wheeled vehicles operating in a 2D environment. The question to be answered is how the rotational velocity of each wheel can be determined, as the kinematic center is moved with a linear velocity $\dot{x}$ relative to the x-axis, $\dot{y}$ relative to the y-axis and $\dot{\theta}$ the rotational velocity around the z-axis. Also the inverse of this problem should be determined. Therefore, the basic wheel types shown in figure 8.7 will be considered.

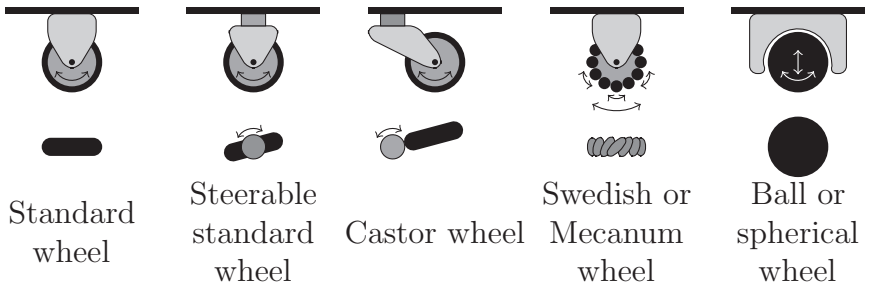


Figure 8.7: The Basic Wheel Types

It is obvious that for the standard and steerable standard wheel there should be no sliding orthogonal to the wheel plane (the velocity must be zero). The linear velocity of each of these wheels in rolling direction can be calculated with $r\dot{\psi}$ (r is the radius of the wheel and $\dot{\psi}$ is the rotational velocity of the wheel). This formula could also be applied for the rolling velocity of the castor wheel and the spherical wheel. If d is the offset between the wheel axis and the vertical axis of rotation, the linear velocity orthogonal to the wheel plane is the rotational velocity around the vertical axis times the length of the offset ($-d_c\dot{\beta}$).

In case of the Swedish or Mecanum wheel, in which passive rollers are mounted in an angle γ (γ is normally 45° or 90°) on the perimeter of the main wheel, the linear velocity in rolling direction is $r\dot{\psi}\cos\gamma$. Because the Swedish or Mecanum wheel is able to move in an omnidirectional way, the velocity orthogonal to the wheel plane can be calculated as $r\dot{\psi}\sin\gamma + r_{pr}\dot{\psi}_{pr}$ (with r_{pr} the radius and $\dot{\psi}_{pr}$ the rotational velocity of the roller).

To determine the velocity of a wheel due to the velocity vector $\vec{v} = (\dot{x}, \dot{y}, \dot{\theta})^T$ of the kinematic center, one must first define the robot coordinate frame, which has its origin in the kinematic center of the vehicle (see figure 8.8). By definition, $\alpha = 0$ if the normal vector of the wheel plane is located on the x-axis and has the same orientation. β determines the angle between the straight line through the kinematic center and the fixing point of the wheel and the y-axis of the wheel frame. Parameter d is the distance from the kinematic center to the fixing point of the wheel on the chassis. The wheel coordinate system has its x-axis in the rolling direction and the y-axis as the normal to the wheel plane. The linear velocity of the wheel is in the direction of the x-axis of the wheel frame. This parameter definition can be used for all wheel types. In case of the Swedish or Mecanum wheel an additional parameter γ has to be introduced, which describes the angle between the x-axis of the wheel and rolling axis of the rollers.

Supposing the velocity vector $\vec{v} = (\dot{x}, \dot{y}, \dot{\theta})^T$ of the kinematic center is given, equation 8.2 and equation 8.1 should be applied to calculate the linear velocity of a standard wheel. Thus, we receive ${}^1\omega_1 = (0, 0, \dot{\theta})^T$ and ${}^1(\vec{v})_1 = (\dot{x}, \dot{y}, 0)^T$. Because there is no additional rotational velocity, all ${}^i\omega_i = (0, 0, \dot{\theta})^T$. See equation 8.2.

A stepwise application of equation 8.1 will deliver the following ${}^i(\vec{v})_i$. After the rotation around the z-axis with angle α, ${}^2\vec{v}_2$ is calculated as:

$$ {}^2\vec{v}_2 = \begin{bmatrix} c\alpha\dot{x} + s\alpha\dot{y} \\ -s\alpha\dot{x} + c\alpha\dot{y} \\ 0 \end{bmatrix} \tag{8.5} $$

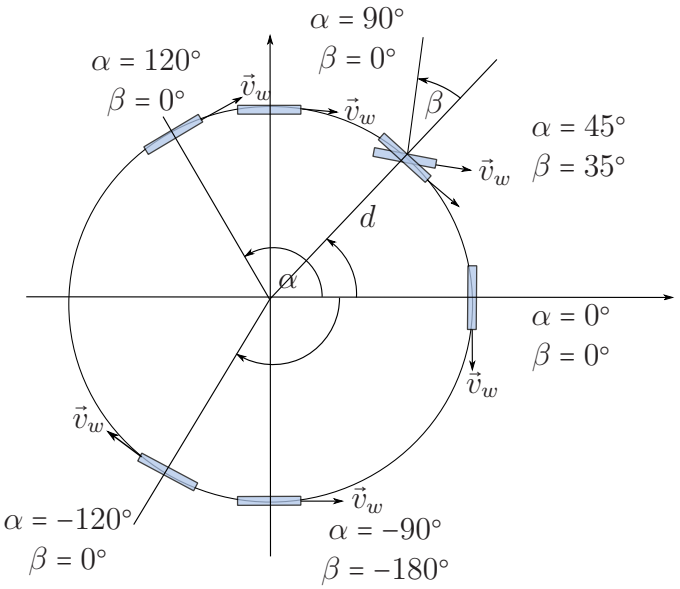


Figure 8.8: The parameters for solving the kinematic problem for different wheel positions

Due to the translation d, ${}^3\vec{v}_3$ is:

$$ {}^3\vec{v}_3 = \begin{bmatrix} c\alpha\dot{x} + s\alpha\dot{y} \\ -s\alpha\dot{x} + c\alpha\dot{y} + d\dot{\theta} \\ 0 \end{bmatrix} \tag{8.6} $$

The last rotation around the z-axis with angle $\beta - 90°$ transfers the x-axis of the last frame to the rolling direction of the wheel. For the calculation of ${}^4\vec{v}_4$ in the next equation, $\sin(\beta - 90°) = -\cos(\beta)$ and $\cos(\beta - 90°) = \sin(\beta)$ is used.

$$ {}^4\vec{v}_4 = \begin{bmatrix} s(\alpha+\beta)\dot{x} - c(\alpha+\beta)\dot{y} - c\beta d\dot{\theta} \\ c(\alpha+\beta)\dot{x} + s(\alpha+\beta)\dot{y} + s\beta d\dot{\theta} \\ 0 \end{bmatrix} \tag{8.7} $$

The last step is to equalize the linear velocity vector of the standard wheel to that of equation 8.7.

$$ \begin{bmatrix} s(\alpha+\beta) & -c(\alpha+\beta) & -c\beta d \\ c(\alpha+\beta) & s(\alpha+\beta) & s\beta d \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} r\dot{\psi} \\ 0 \\ 0 \end{bmatrix} \tag{8.8} $$

In case of the steerable standard wheel, equation 8.8 can be used in the same way, if the fixed angle β is replaced by a function $\beta(t)$. This equation could also be applied to the spherical wheel (because of the forces which affect the wheel and change $\beta(t)$, only a linear velocity in the rolling direction exists). If the wheel is a castor wheel, the y-component of the velocity vector is depending on the angular velocity $\dot{\beta}$ and the length of the rod (see equation 8.9).

$$ \begin{bmatrix} s(\alpha+\beta) & -c(\alpha+\beta) & -c\beta d \\ c(\alpha+\beta) & s(\alpha+\beta) & s\beta d \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} r\dot{\psi} \\ -d_c\dot{\beta} \\ 0 \end{bmatrix} \tag{8.9} $$

The Swedish or Mecanum wheel is able to move in an omnidirectional way. Therefore, lateral movement of the wheel should be possible and can be calculated with equation 8.10.

$$\begin{bmatrix} s(\alpha+\beta+\gamma) & -c(\alpha+\beta+\gamma) & -c(\beta+\gamma)d \\ c(\alpha+\beta+\gamma) & s(\alpha+\beta+\gamma) & s(\beta+\gamma)d \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} r\dot{\psi}\cos\gamma \\ r\dot{\psi}\sin\gamma + r_{pr}\dot{\psi}_{pr} \\ 0 \end{bmatrix} \tag{8.10}$$

8.4.1 Kinematics of a Differential Vehicle

To calculate the kinematics of a differential drive vehicle, first the wheel types used have to be determined. This type of robot has two fixed standard wheels which are mounted on one axis. The kinematic center is located in the middle of the axis; the distance between the kinematic center and each wheel should be d. To solve the kinematics problem the coordinate system to define the parameters must be specified. The origin of this frame lies on the kinematic center. One solution for modelling the wheel configuration is to place the wheels on the y-axis of the coordinate frame. As shown in figure 8.8, the parameters are $\alpha_l = 90°, \beta_l = 0°, \alpha_r = -90°, \beta_r = 180°$. An example of a differential drive robot is Marvin, the mobile vehicle of the University of Kaiserslautern. See figure 8.9.

Figure 8.9: The autonomous vehicle Marvin of the University of Kaiserslautern

Based on equation 8.8 one obtains for the left and the right wheel:

$$\begin{aligned} s(\alpha_l + \beta_l)\dot{x} - c(\alpha_l + \beta_l)\dot{y} - c\beta_l d\dot{\theta} &= r_l\dot{\psi}_l \\ c(\alpha_l + \beta_l)\dot{x} + s(\alpha_l + \beta_l)\dot{y} + s\beta_l d\dot{\theta} &= 0 \\ s(\alpha_r + \beta_r)\dot{x} - c(\alpha_r + \beta_r)\dot{y} - c\beta_r d\dot{\theta} &= r_r\dot{\psi}_r \\ c(\alpha_r + \alpha_r)\dot{x} + s(\alpha_r + \beta_r)\dot{y} + s\beta_r d\dot{\theta} &= 0 \end{aligned} \tag{8.11}$$

If the above mentioned parameters are inserted, the following equation will result:

$$\begin{aligned} \dot{x} - d\dot{\theta} &= r_l\dot{\psi}_l \\ \dot{y} &= 0 \\ \dot{x} + d\dot{\theta} &= r_r\dot{\psi}_r \\ \dot{y} &= 0 \end{aligned} \tag{8.12}$$

After solving the equation system on receives:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(r_l\dot{\psi}_l + r_r\dot{\psi}_r) \\ 0 \\ \frac{1}{2d}(-r_l\dot{\psi}_l + r_r\dot{\psi}_r) \end{bmatrix} \tag{8.13}$$

8.4.2 Kinematics of Omnidirectional Vehicles

To increase the mobility of a vehicle, an omnidirectional drive can be used. The climbing robot Cromsci. See figure 8.10) of the University of Kaiserslautern.

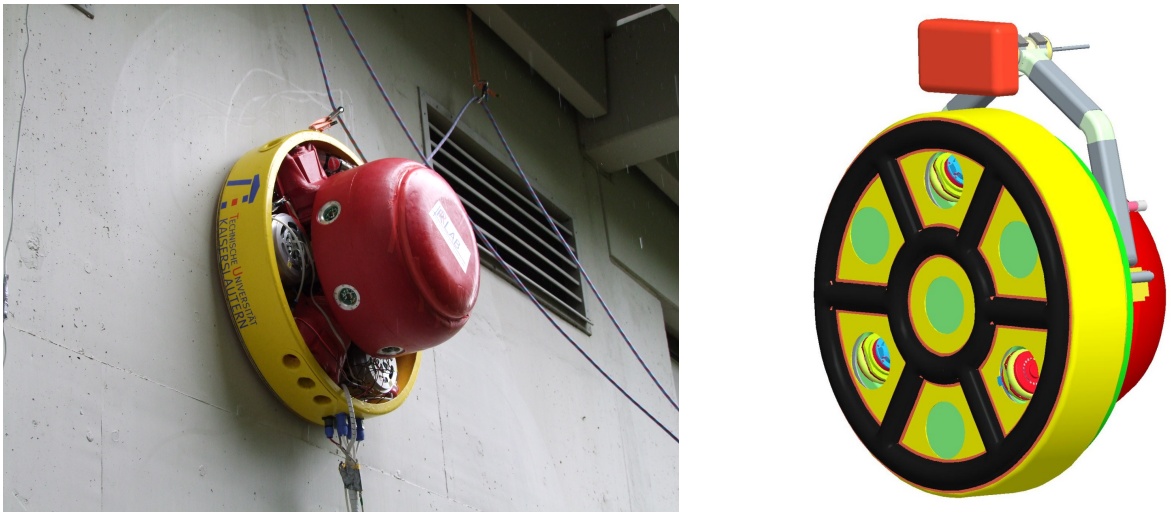


Figure 8.10: The climbing robot Cromsci (left) and the wheel settings (right)

For example, is equipped with such a drive, in which 3 steerable standard wheels are mounted with an angle displacement of 120° between them. See figure 8.11.

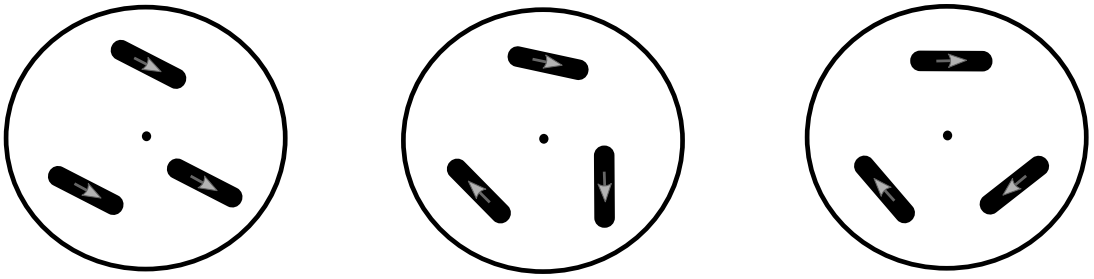


Figure 8.11: Typical orientations of the 3 steerable wheels of an omnidirectional vehicle

To set up the kinematics equations one can model the wheel configuration as shown in figure 8.8 with the kinematic center in the middle of the robot. The front wheel is located on the x-axis, the two rear wheels have a displacement to the front wheel of ±120°. As shown in figure 8.8 $\alpha_1 = 0°, \alpha_2 = 120°, \alpha_3 = -120°$. $\beta_{1,2,3}$ are the control parameters to determine the direction of the vehicle movements. The parameter d describes the distance between the wheel's contact point and the robot center.

For the navigation of Cromsci it is necessary to calculate based on the desired linear and rotational velocities of the kinematic center $(\dot{x}, \dot{y}, \dot{\theta})^T$, the single wheel velocities and the orientations $(r_1\dot{\psi}_1, r_2\dot{\psi}_2, r_3\dot{\psi}_3, \beta_1, \beta_2, \beta_3)$.

Applying equation 8.8 for each wheel leads to the following equation systems:

$$\begin{aligned}
s(\alpha_1+\beta_1)\dot{x} - c(\alpha_1+\beta_1)\dot{y} - d\cdot c(\beta_1)\dot{\theta} &= r_1\dot{\psi}_1\\
s(\alpha_2+\beta_2)\dot{x} - c(\alpha_2+\beta_2)\dot{y} - d\cdot c(\beta_2)\dot{\theta} &= r_2\dot{\psi}_2\\
s(\alpha_3+\beta_3)\dot{x} - c(\alpha_3+\beta_3)\dot{y} - d\cdot c(\beta_3)\dot{\theta} &= r_3\dot{\psi}_3\\
c(\alpha_1+\beta_1)\dot{x} + s(\alpha_1+\beta_1)\dot{y} + d\cdot s(\beta_1)\dot{\theta} &= 0\\
c(\alpha_2+\beta_2)\dot{x} + s(\alpha_2+\beta_2)\dot{y} + d\cdot s(\beta_2)\dot{\theta} &= 0\\
c(\alpha_3+\beta_3)\dot{x} + s(\alpha_3+\beta_3)\dot{y} + d\cdot s(\beta_3)\dot{\theta} &= 0
\end{aligned} \tag{8.14}$$

Based on the last 3 equations of 8.14, the steering angles $\beta_i, i = 1,2,3$ are determined:

$$\begin{aligned}
& c(\alpha_i+\beta_i)\cdot\dot{x} + s(\alpha_i+\beta_i)\cdot\dot{y} + d\cdot s(\beta_i)\cdot\dot{\theta} = 0\\
\Rightarrow & c(\alpha_i)\cdot c(\beta_i)\cdot\dot{x} - s(\alpha_i)\cdot s(\beta_i)\cdot\dot{x}\\
& + s(\alpha_i)\cdot c(\beta_i)\cdot\dot{y} + c(\alpha_i)\cdot s(\beta_i)\cdot\dot{y} + d\cdot s(\beta_i)\cdot\dot{\theta} = 0\\
\Rightarrow & c(\beta_i)\cdot\left(c(\alpha_i)\cdot\dot{x} + s(\alpha_i)\cdot\dot{y}\right) = s(\beta_i)\cdot\left(s(\alpha_i)\cdot\dot{x} - c(\alpha_i)\cdot\dot{y} - d\cdot\dot{\theta}\right)\\
\Rightarrow & \tan(\beta_i) = \frac{s(\beta_i)}{c(\beta_i)} = \frac{c(\alpha_i)\cdot\dot{x} + s(\alpha_i)\cdot\dot{y}}{s(\alpha_i)\cdot\dot{x} - c(\alpha_i)\cdot\dot{y} - d\cdot\dot{\theta}}\\
\Rightarrow & \beta_i = \operatorname{atan2}\left(\left(c(\alpha_i)\cdot\dot{x} + s(\alpha_i)\cdot\dot{y}\right), \left(s(\alpha_i)\cdot\dot{x} - c(\alpha_i)\cdot\dot{y} - d\cdot\dot{\theta}\right)\right)
\end{aligned} \tag{8.15}$$

From equation 8.14, the angular velocity of the wheel $\dot{\psi}_i$ can be calculated using β_i:

$$\dot{\psi}_i = \frac{1}{r_i}\left(s(\alpha_i+\beta_i)\dot{x} - c(\alpha_i+\beta_i)\dot{y} - d\cdot c(\beta_i)\dot{\theta}\right) \tag{8.16}$$

8.4.3 Kinematics of a Vehicle with Mecanum Wheels

Another drive system suited for an omnidirectional vehicle are Mecanum wheels. Those are convex cylinders arranged in a 45° angle relative to the wheel plane. Two pairs of independently driven Mecanum wheels are sufficient to enable omnidirectional movement. The orientation of the rollers of the wheels lying on a common diagonal axis is equal. The rollers of the other two wheels are oriented in the opposite direction. In figure 8.12 (right side) 4 typical movements of the vehicle are shown. If all wheels move with the same velocity in the same direction, the robot drives straight ahead. The machine will turn if the right and left wheels move in opposite direction with the same velocity. A sideward motion is possible if the neighboring wheels move in opposite direction with the same velocity. A diagonal motion results if the two wheels on the diagonal move in the same direction with the same velocity. This type of drive was applied for the vehicles PRIAMOS of Prof. Dillmann's research group at the University of Karlsruhe. See figure 8.13.

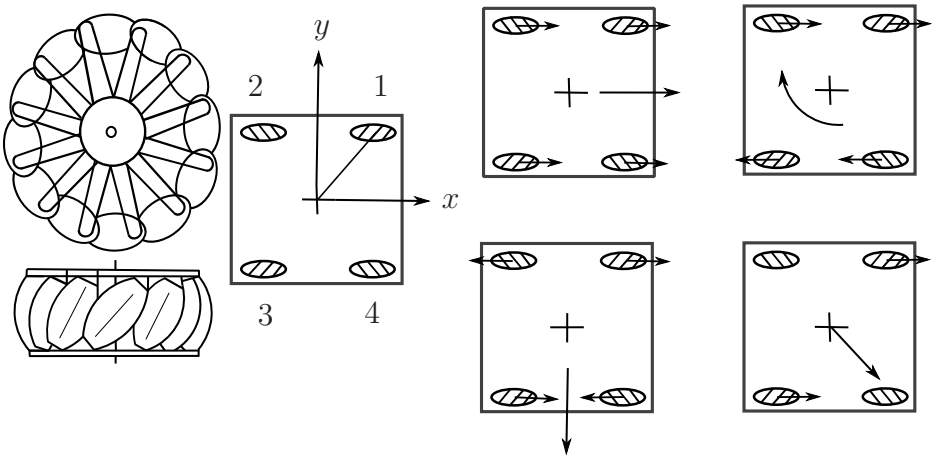


Figure 8.12: Schematic configuration of a Mecanum wheel

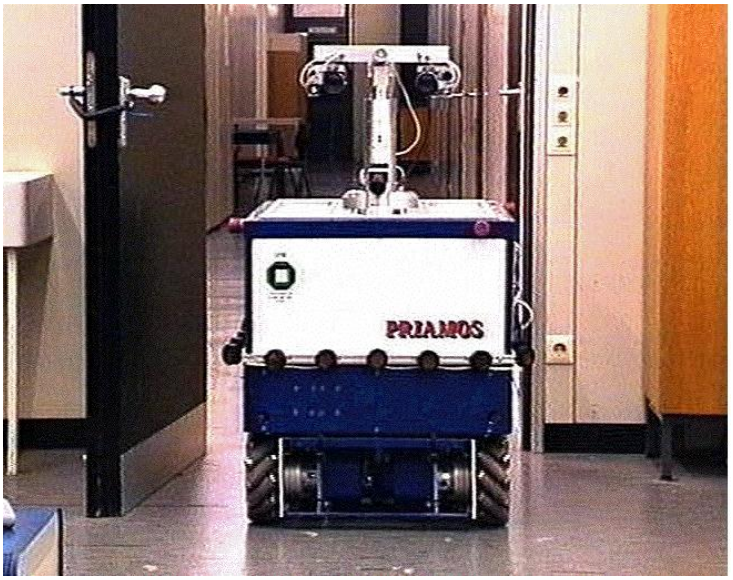


Figure 8.13: The mobile robot PRIAMOS of the University of Karlsruhe driven by Mecanum wheels [Dillmann 95] (courtesy of Prof. Dillmann, TH Karlsruhe)

To set up the kinematic equation, the parameters (α, β, γ) for each wheel must be determined. The order of the wheels is shown in figure 8.12. The parameters for four wheels are:

$$\begin{aligned}
&\alpha_1 = 45°, && \beta_1 = 45°, && \gamma_1 = -45°\\
&\alpha_2 = 135°, && \beta_2 = -45°, && \gamma_2 = 45°\\
&\alpha_3 = -135°, && \beta_3 = 225°, && \gamma_3 = -45°\\
&\alpha_4 = -45°, && \beta_4 = 135°, && \gamma_4 = 45°
\end{aligned}$$

Using equation 8.10, and supposing all driven wheels have the same radius r, the same distance d from the kinematic center and the above mentioned parameters for α, β, γ are inserted, we receive:

$$s(45°)\dot{x} - c(45°)\dot{y} - d\dot{\theta} = r \cdot c(-45°)\dot{\psi}_1 \tag{8.17}$$
$$s(135°)\dot{x} - c(135°)\dot{y} - d\dot{\theta} = r \cdot c(45°)\dot{\psi}_2 \tag{8.18}$$
$$s(45°)\dot{x} - c(45°)\dot{y} - d\dot{\theta} = r \cdot c(-45°)\dot{\psi}_3 \tag{8.19}$$
$$s(135°)\dot{x} - c(135°)\dot{y} - d\dot{\theta} = r \cdot c(45°)\dot{\psi}_4 \tag{8.20}$$

Based on this equation system, the velocities $\dot{x}, \dot{y}, \dot{\theta}$ of the kinematic center can be calculated:

$$\begin{aligned}
\dot{x} &= \frac{r}{4}(\dot{\psi}_1 + \dot{\psi}_2 + \dot{\psi}_3 + \dot{\psi}_4)\\
\dot{y} &= \frac{r}{4}(-\dot{\psi}_1 + \dot{\psi}_2 - \dot{\psi}_3 + \dot{\psi}_4)\\
\dot{\theta} &= \frac{r}{d\sqrt{2}}(\dot{\psi}_1 - \dot{\psi}_2 - \dot{\psi}_3 + \dot{\psi}_4)
\end{aligned} \tag{8.21}$$

The velocity vector of the kinematic center can be determined as:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \frac{r_{\text{wheel}}}{4} \cdot \begin{bmatrix}1 & 1 & 1 & 1\\ -1 & 1 & -1 & 1\\ C & -C & -C & C\end{bmatrix} \cdot \begin{bmatrix}\dot{\psi}_1\\ \dot{\psi}_2\\ \dot{\psi}_3\\ \dot{\psi}_4\end{bmatrix} \tag{8.22}$$

with $C = \frac{2\sqrt{2}}{d}$.

In order to assume the absence of slip the following must hold: $\dot{\psi}_4 = \dot{\psi}_1 + \dot{\psi}_2 - \dot{\psi}_3$

8.4.4 Poses Calculation Based on Velocity

Using equation 8.23 and introducing the time interval Δt, the incremental paths can be determined for 2D navigation as:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \cdot \Delta t \tag{8.23}$$

Assuming the velocity $v = (v_x(t), v_y(t))_A$ given in a robot fixed coordinate frame (x_A, y_A), angular velocity $\omega(t)$ and the robot pose (x, y, θ) in the world coordinate system given, one can compute the trajectory as:

$$\theta(t) = \int_0^t \omega(\tau) d\tau + \theta_0 \tag{8.24}$$

$$x(t) = \int_0^t \dot{x}(\tau) d\tau + x_0 \tag{8.25}$$

$$y(t) = \int_0^t \dot{y}(\tau) d\tau + y_0 \tag{8.26}$$

Vehicle velocities are $\dot{x}$ due to the x-axis, $\dot{y}$ due to the y-axis and $\omega = \dot{\theta}$ around z-axis.

8.5 Static Forces/Moments

8.5.1 Static Forces

Static forces is the calculation of the forces/torques without considering movements. For example how high do the torques have to be, in order to keep an object of mass m in a certain position with TCP?
The solution idea for this is:

- Propagate powers and moments from link to link
- Calculate a force/moment balance for each member
- Start with the TCP

Let $\vec{f}_i$ be the force that attacks on link i through link $i-1$, and then $\vec{n}_i$ is the torque (moment) that attacks link i through link $i-1$.
Forces/moments equation (influence of the next higher link) is:

$$\begin{aligned} {}^i\vec{f}_i &= {}^i\vec{f}_{i+1} \\ {}^i\vec{n}_i &= {}^i\vec{n}_{i+1} + {}^i\vec{u}_{i+1} \times {}^i\vec{f}_{i+1} \end{aligned}$$

8.5.2 Propagation

Static propagation of the forces /moments from link to link

Calculation of the forces in link i:	${}^{i}\vec{f}_i$	$=$	${}^{i}_{i+1}R \cdot {}^{i+1}\vec{f}_{i+1}$
Calculation of the moment in link i:	${}^{i}\vec{n}_i$	$=$	${}^{i}_{i+1}R \cdot {}^{i+1}\vec{n}_{i+1} + {}^{i}\vec{u}_{i+1} \times {}^{i}\vec{f}_{i+1}$
Required moment in rotational joints	τ_{i+1}	$=$	${}^{i}\vec{n}_i^T \cdot {}^{i}\vec{e}_{z_i}$
Required force in linear joints	τ_{i+1}	$=$	${}^{i}\vec{f}_i^T \cdot {}^{i}\vec{e}_{z_i}$

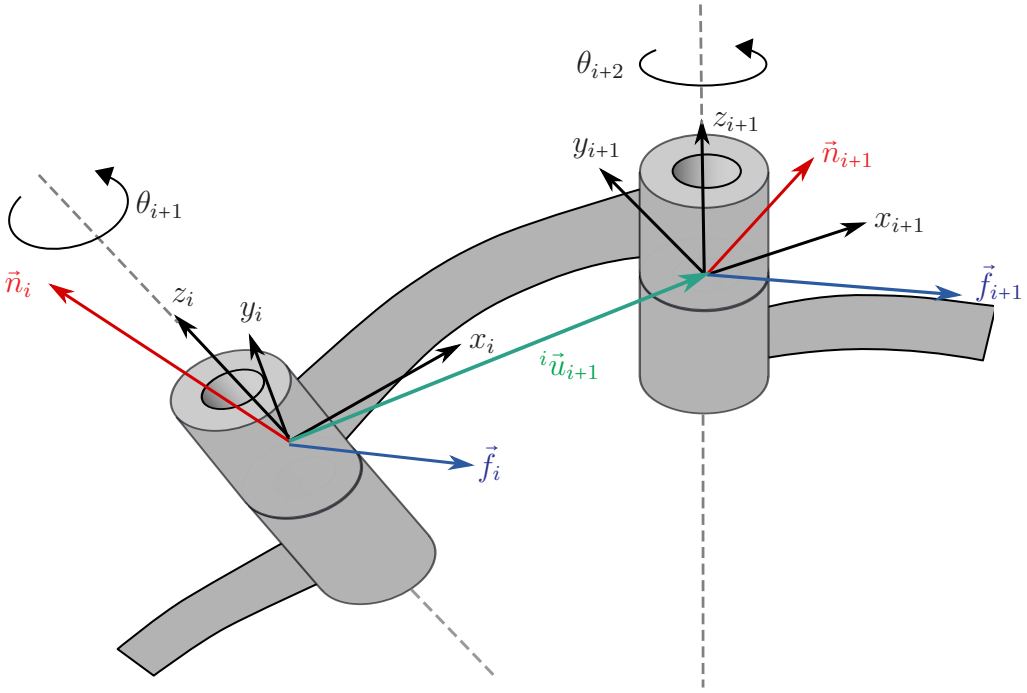


Figure 8.14: Static Forces/Moments: Propagation

Example:

Given: Forces f, applied at the TCP.
Desired: Torques in the joints.

$$
\begin{aligned}
{}^2\vec{f}_2 &= \begin{bmatrix} {}^2f_{2_x} \\ {}^2f_{2_y} \\ 0 \end{bmatrix} \\
{}^2\vec{n}_2 &= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\
{}^1\vec{n}_1 &= {}^1\vec{n}_2 + {}^1\vec{u}_2 \times {}^1\vec{f}_1 \\
&= {}^1\vec{u}_2 \times ({}^1_2R \cdot {}^2\vec{f}_2) \\
&= \begin{bmatrix} a_2c_2 \\ a_2s_2 \\ 0 \end{bmatrix} \times \begin{bmatrix} c_2 \cdot {}^2f_{2_x} - s_2 \cdot {}^2f_{s_y} \\ s_2 \cdot {}^2f_{2_x} + c_2 \cdot {}^2f_{s_y} \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 0 \\ a_2 \cdot {}^2f_{2_y} \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
{}^0\vec{n}_0 &= {}^0_1R \cdot {}^1\vec{n}_1 + {}^0\vec{u}_1 \times {}^0\vec{f}_0 \\
&= \begin{bmatrix} 0 \\ 0 \\ a_2 \cdot {}^2f_{2_y} \end{bmatrix} + \begin{bmatrix} a_1c_1 \\ a_1s_1 \\ 0 \end{bmatrix} \times \begin{bmatrix} c_{12} \cdot {}^2f_{2_x} & -s_{12} \cdot {}^2f_{s_y} \\ s_{12} \cdot {}^2f_{2_x} & +c_{12} \cdot {}^2f_{s_y} \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 0 \\ a_2 \cdot {}^2f_{2_y} + s_2a_1 \cdot {}^2f_{2_x} + c_2a_1 \cdot {}^2f_{2_y} \end{bmatrix} \\
\tau_1 &= la_2 \cdot {}^2f_{2_y} + s_2a_1 \cdot {}^2f_{2_x} + c_2a_1 \cdot {}^2f_{2_y} \\
\tau_2 &= a_2 \cdot {}^2f_{2_y}
\end{aligned}
$$

See figure 8.15.

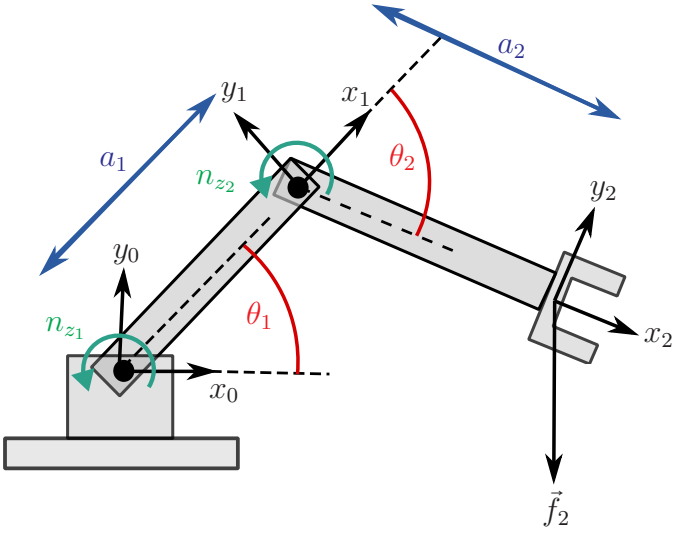


Figure 8.15: Example: Static forces

Transformation of Forces: Application Example

TCP grips tool → Load cell measures forces and moments in the wrist.
Desired: Forces and torques at the end of the tool. See figure 8.16.

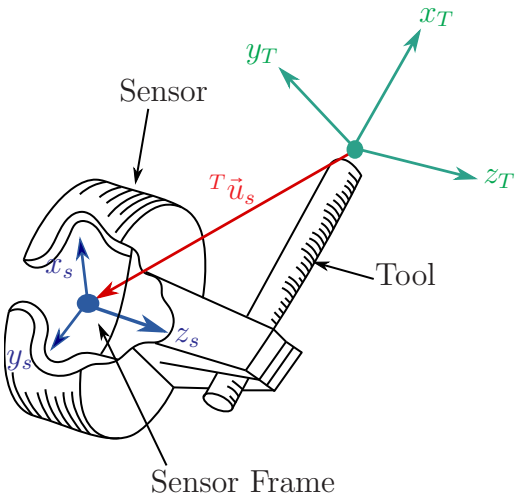


Figure 8.16: Transformation of forces: Example

8.5.3 Force/Moment Calculation with Jacobian Matrix

Contemplation of the *virtual work* in the Cartesian space - and in the configuration space is:
Work, which is caused by the forces and moments $\vec{\eta}$ acting on the TCP, must be equal to the work that is applied in the joints by adjusting forces and setting moments $\vec{\tau}$.

$$\vec{\eta}^T \cdot \dot{\vec{y}} = \vec{\tau}^T \cdot \dot{\vec{\theta}} \tag{8.27}$$

with:

$\vec{\eta} = \begin{bmatrix} \vec{f}_{TCP} \\ \vec{n}_{TCP} \end{bmatrix}$:6×1, cartesian force-/moment vector at TCP
$\dot{\vec{y}}$:6×1, infinitesimal offset vector of TCP
$\vec{\tau}$:6×1, force-/moment vector in joints
$\dot{\vec{\theta}}$:6×1, change of joint positions

By inserting the relationship $\dot{\vec{y}} = J(\vec{\theta}) \cdot \dot{\vec{\theta}}$, 8.27 can be transformed into:

$$\vec{\eta}^T \cdot J(\vec{\theta}) \cdot \dot{\vec{\theta}} = \vec{\tau}^T \cdot \dot{\vec{\theta}}$$

Thus:

$$\vec{\eta}^T \cdot J(\vec{\theta}) = \vec{\tau}^T \quad \text{und} \quad \vec{\tau} = J^T(\vec{\theta}) \cdot \vec{\eta}$$

9. Dynamics Modelling

This chapter strats with utilization of the dynamic model, and linear and angular accelerations. Then, moment of inertia will be discussed, following by calculation of dynamics via Newton-Euler and calculation of dynamics via Lagrange. It will end with comparison of approaches.

9.1 Dynamics Modelling

A *Dynamic model* calculates the relationship between forces, moments and movements that occur in a mechanical multi-body system.
The purposes of the dynamic models are:

- Analysis of dynamics
- Synthesis of mechanical structures
- Modelling of elastic structures
- Controller design

Application:

Phases of robot development and operation is shown in figure 9.1

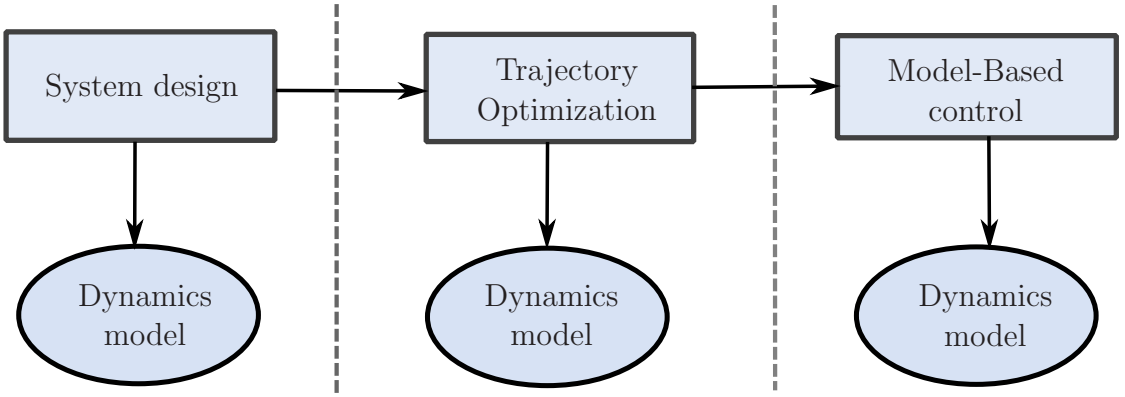


Figure 9.1: The phases of robot development and operation

The modeling is divided into different phases. It takes a long time and high effort, and also errors and inconsistencies are likely. The reusability of dynamic model code is difficult if the structure changes (kinematic structure, joint types, actuators).

Equations of Movement

In the dynamic model, the relationships between forces/torque and the positions, velocities and accelerations of the n links are represented by:

$$\vec{\tau} = M(\vec{\theta}) \cdot \ddot{\vec{\theta}} + n(\dot{\vec{\theta}}, \vec{\theta}) + g(\vec{\theta}) + R \cdot \dot{\vec{\theta}} \tag{9.1}$$

In which:

$\vec{\tau}$	:n×1 vector of general actuating forces and torques
$M(\vec{\theta})$	:n×n moment of inertia matrix
$n(\dot{\vec{\theta}}, \vec{\theta})$	:n×1 vector with centrifugal and Coriolis components
$g(\vec{\theta})$	:n×1 vector with gravitational components
R	:n×n diagonal matrix describing friction forces
$\vec{\theta}$	:n×1 manipulator variables

9.1.1 Direct Dynamic Problem

The resulting difference of motion is calculated from the mass, external forces and torques, as well as pose, initial velocity and accelerations. See figure 9.2.

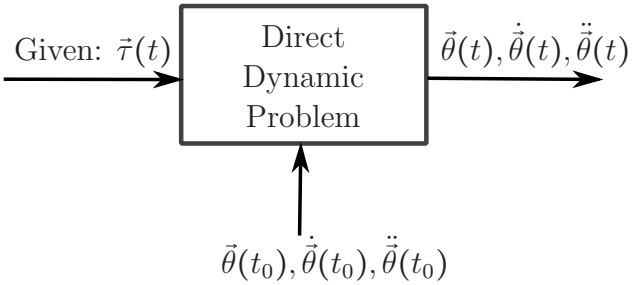


Figure 9.2: Direct Dynamic Problem

Solve Equation 9.1 for: $\vec{\theta}(t), \dot{\vec{\theta}}(t), \ddot{\vec{\theta}}(t)$

9.1.2 Inverse Dynamic Problem

Determine the required actuating forces and torques from the desired parameters of motion and kinematics. See figure 9.3.

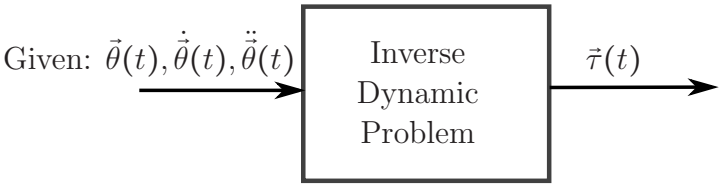


Figure 9.3: Inverse Dynamic Problem

Solve the equation 9.1.

9.2 Acceleration of Rigid Bodies

Let q be any point in any coordinate system. Then,

Linear Acceleration:

$$ {}^{B}\dot{\vec{v}}_q = \frac{d}{dt}\,{}^{B}\vec{v}_q = \lim_{\Delta t \to 0} \frac{{}^{B}\vec{v}_q(t+\Delta t) - {}^{B}\vec{v}_q(t)}{\Delta t} $$

Angular Acceleration:

$$ {}^{A}\dot{\vec{\omega}}_B = \frac{d}{dt}\,{}^{A}\vec{\omega}_B = \lim_{\Delta t \to 0} \frac{{}^{A}\vec{\omega}_B(t+\Delta t) - {}^{A}\vec{\omega}_B(t)}{\Delta t} $$

See figure 9.4

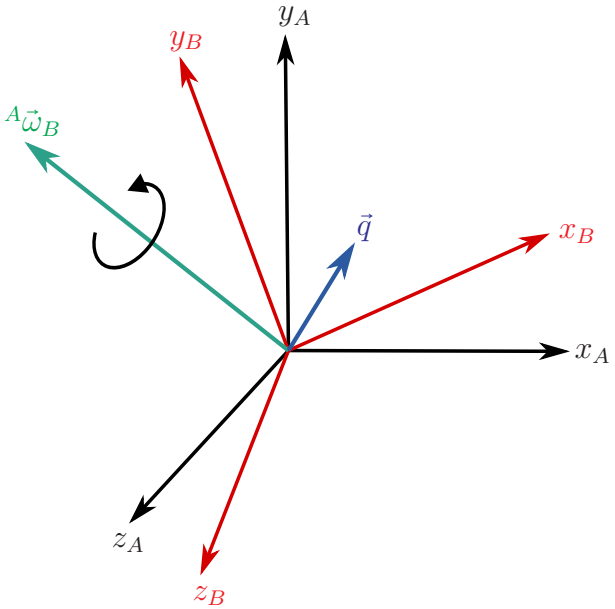


Figure 9.4: Acceleration of Rigid Bodies

9.2.1 Linear Acceleration

Calculation of the linear acceleration based on velocity:

$$ {}^{A}\vec{v}_q = {}^{A}_{B}R \cdot {}^{A}\vec{v}_q + {}^{A}\vec{\omega}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{q} \quad \text{(Compare 8.2)} \tag{9.2} $$

Because the origins of frames A and B coincide, it follows:

$$ \frac{d}{dt}({}^{A}_{B}R \cdot {}^{B}\vec{q}) = {}^{A}_{B}R \cdot {}^{A}\vec{v}_q + {}^{A}\vec{\omega}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{q} \tag{9.3} $$

Derivative of velocity results in linear acceleration:

$$ {}^{A}\dot{\vec{v}}_q = \frac{d}{dt}({}^{A}_{B}R \cdot {}^{B}\vec{v}_q) + {}^{A}\dot{\vec{\omega}}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{q} + {}^{A}\vec{\omega}_B \times \frac{d}{dt}({}^{A}_{B}R \cdot {}^{B}\vec{q}) \tag{9.4} $$

Substituting 9.3 into 9.4 holds:

$$ {}^{A}\dot{\vec{v}}_q = {}^{A}_{B}R \cdot {}^{B}\dot{\vec{v}}_q + {}^{A}\vec{\omega}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{v}_q + {}^{A}\dot{\vec{\omega}}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{q} + {}^{A}\vec{\omega}_B \times ({}^{A}_{B}R \cdot {}^{B}\vec{v}_q + {}^{A}\vec{\omega}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{q}) $$

After simplification:

$$ {}^{A}\dot{\vec{v}}_q = {}^{A}_{B}R \cdot {}^{B}\dot{\vec{v}}_q + 2 \cdot ({}^{A}\vec{\omega}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{v}_q) + {}^{A}\dot{\vec{\omega}}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{q} + {}^{A}\vec{\omega}_B \times ({}^{A}\vec{\omega}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{q}) $$

In general case, with frames A and B without common origin:

$$ {}^{A}\dot{\vec{v}}_q = {}^{A}\dot{\vec{v}}_{O_B} + {}^{A}_{B}R \cdot {}^{B}\dot{\vec{v}}_q + 2 \cdot ({}^{A}\vec{\omega}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{v}_q) + {}^{A}\dot{\vec{\omega}}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{q} + {}^{A}\vec{\omega}_B \times ({}^{A}\vec{\omega}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{q}) $$

Considering that ${}^{B}\vec{q}$ does not move, the following applies:

$$ \begin{aligned} {}^{B}\vec{v}_q &= {}^{B}\dot{\vec{v}}_q = \vec{0} \\ \Rightarrow {}^{A}\dot{\vec{v}}_q &= {}^{A}\dot{\vec{v}}_{U_B} + {}^{A}\vec{\omega}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{q} + {}^{A}\vec{\omega}_B \times ({}^{A}\vec{\omega}_B \times {}^{A}_{B}R \cdot {}^{B}\vec{q}) \end{aligned} $$

9.3 Distribution of Mass

Geometric Pre-Examination:

Let us define some Geometric Pre-Examinations which are used in this section:

dm	Mass particle
C_B	Center of mass of body K_B
$\vec{u}_{C_B}$	Vector to center of mass
$\vec{r}$	Vector to mass particle

See figure 9.5.

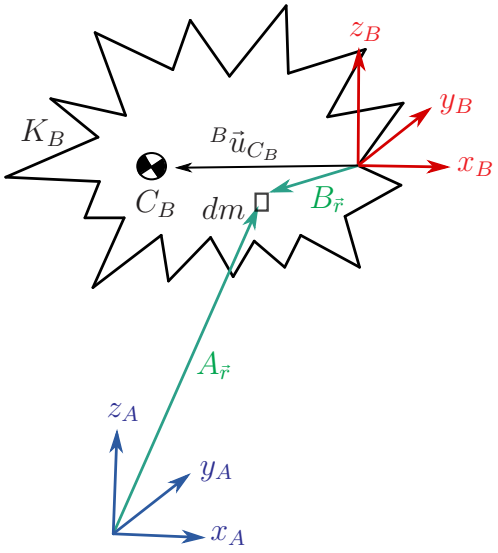


Figure 9.5: Mass distribution: geometric preview

9.3.1 Inertia Tensor

Inertia tensor in reference to frame A, specifying the body´s inertia regarding rotatin:

$$ {}^A I = \begin{bmatrix} {}^A i_{xx} & -{}^A i_{xy} & -{}^A i_{xz} \\ -{}^A i_{xy} & {}^A i_{yy} & -{}^A i_{yz} \\ -{}^A i_{xz} & -{}^A i_{yz} & {}^A i_{zz} \end{bmatrix} $$

Scalar elements of the mass inertia tensor (calculation through integration over mass distribution M):

- Axial moments of inertia:

$$ {}^A i_{xx} = \iiint_M (\, y_A^2 + \, z_A^2) dm \qquad {}^A i_{yy} = \iiint_M (\, x_A^2 + \, z_A^2) dm \qquad {}^A i_{zz} = \iiint_M (\, x_A^2 + \, y_A^2) dm $$

- Inertia products:

$$ {}^A i_{xy} = \iiint_M \, x_A y_A dm \qquad {}^A i_{xz} = \iiint_M \, x_A z_A dm \qquad {}^A i_{yz} = \iiint_M \, y_A z_A dm $$

For a point mass the tensor becomes a zero matrix.

9.3.2 Example Cuboid

Calculation of inertia tensor for cuboid (see figure 9.6) with uniform density ρ:

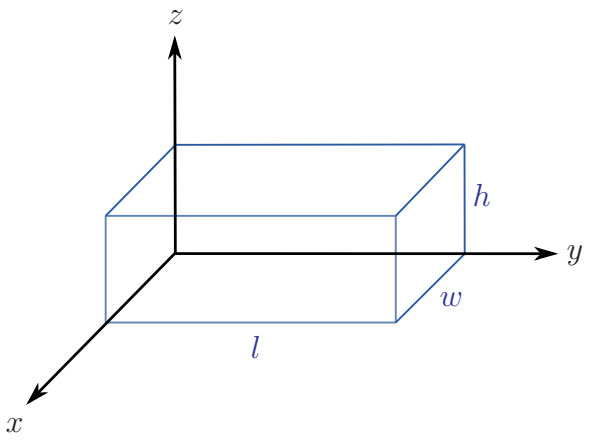


Figure 9.6: Mass Distribution: Example Cuboid

- With $dm = \rho dx dy dz$ it follows:

$$\begin{aligned}
{}^{A}i_{xx} &= \int_0^h \int_0^l \int_0^w (y_A^2 + z_A^2)\rho dx_A \, dy_A \, dz_A \\
&= \int_0^h \int_0^l (y_A^2 + z_A^2) w\rho \, dy_A \, dz_A \\
&= \int_0^h (\frac{l^3}{3} + z_A^2 l) w\rho \, dz_A \\
&= (\frac{hl^3 w}{3} + \frac{h^3 lw}{3})\rho \\
&= \frac{m}{3}(l^2 + h^2) \qquad \text{(with total mass } m\text{)}
\end{aligned}$$

- For ${}^{A}i_{yy}$ and ${}^{A}i_{zz}$ it follows analogously:

$${}^{A}i_{yy} = \frac{m}{3}(w^2 + h^2) \qquad {}^{A}i_{zz} = \frac{m}{3}(l^2 + w^2)$$

Calculation of:

$${}^{A}i_{xy} = \int_0^h \int_0^l \int_0^w x_A y_A \rho dx_A \, dy_A \, dz_A = \int_0^h \int_0^l \frac{w^2}{2} y_A \rho \, dy_A \, dz_A = \int_0^h \frac{w^2 l^2}{4} \rho \, dz_A = \frac{m}{4} wl$$

Analogous computation of:

$${}^{A}i_{xz} = \frac{m}{4} hw \qquad {}^{A}i_{yz} = \frac{m}{4} hl$$

Inertia tensor

$${}^{A}I = \begin{bmatrix} \frac{m}{3}(l^2 + h^2) & -\frac{m}{4} wl & -\frac{m}{4} hw \\ -\frac{m}{4} wl & \frac{m}{3}(w^2 + h^2) & -\frac{m}{4} hl \\ -\frac{m}{4} hw & -\frac{m}{4} hl & \frac{m}{3}(l^2 + w^2) \end{bmatrix}$$

Steiner's Theorem (For parallel axes through the center of mass):

For arbitrary frame A and frame C with origin in center of mass and axes parallel to frame A, the following holds:

$$ {}^{A}i_{zz} = {}^{C}i_{zz} + m \cdot ({}^{A}u_{C_x}^2 + {}^{A}u_{C_y}^2) \qquad {}^{A}i_{xy} = {}^{C}i_{xy} - m \cdot {}^{A}u_{C_x} \cdot {}^{A}u_{C_y} $$

With position vector ${}^{A}\vec{u}_C = ({}^{A}u_{C'_x}, {}^{A}u_{C'_y}, {}^{A}u_{C_z})^T$.

Remaining scalars follow analogously.

Steiner's theorem in matrix notation gives:

$$ {}^{A}I = {}^{C}I + m \cdot [{}^{A}\vec{u}_C^T \cdot {}^{A}\vec{u}_C \cdot \mathcal{I}_3 - {}^{A}\vec{u}_C^T \cdot {}^{A}\vec{u}_C] \qquad \text{with } \mathcal{I}_3 = 3 \times 3 \text{ identity matrix} $$

Applied to the cuboid example, it follows:

$$ {}^{A}\vec{u}_C = \begin{bmatrix} {}^{A}u_{C_x} \\ {}^{A}u_{C_y} \\ {}^{A}u_{C_z} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} w \\ l \\ h \end{bmatrix} \qquad {}^{C}i_{zz} = \frac{m}{12} \cdot (w^2 + l^2) \qquad {}^{C}i_{xy} = 0 $$

The remaining elements follow from symmetry considerations. Then resulting inertia tensor:

$$ {}^{C}I = \begin{bmatrix} \frac{m}{12} \cdot (h^2 + l^2) & 0 & 0 \\ 0 & \frac{m}{12} \cdot (w^2 + h^2) & 0 \\ 0 & 0 & \frac{m}{12} \cdot (l^2 + w^2) \end{bmatrix} $$

9.4 Geometric Description of Neighboring Arm Elements

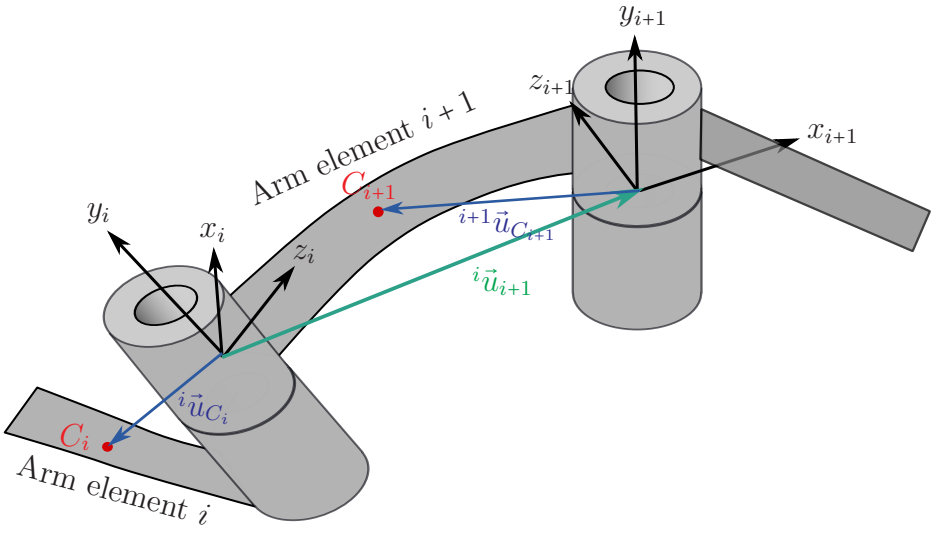


Figure 9.7: Geometric Description of Neighboring Arm Elements with $i \in 0, \ldots, n$

with:

C_i	Center of mass of link i
${}^i\vec{u}_{C_i}$	Vector to center of mass of link i in Coordinate system i
${}^i\vec{u}_{i+1}$	Vector from origin i to $i+1$ in coordinate system i

Derivation of Equations of Motion

In principle there are two possible solutions:

Synthetic method (Newton-Euler): Free body diagram

- Conservation of (angular) momentum
- Elimination of constraining forces results in equations of motion

Analytic methods (Lagrange): Application of extermal principle

- Work and energy considerations
- Formal derivation yields equations of movement

9.5 Newton-Euler Method

Fundamental Equations

Newton equation is: $\vec{f}_C = m \cdot \dot{\vec{v}}_C$
With:

m	=	Total mass of body
$\dot{\vec{v}}_C$	=	Acceleration in center of mass C
$\vec{f}_C$	=	Force acting on the center

See figure 9.8.

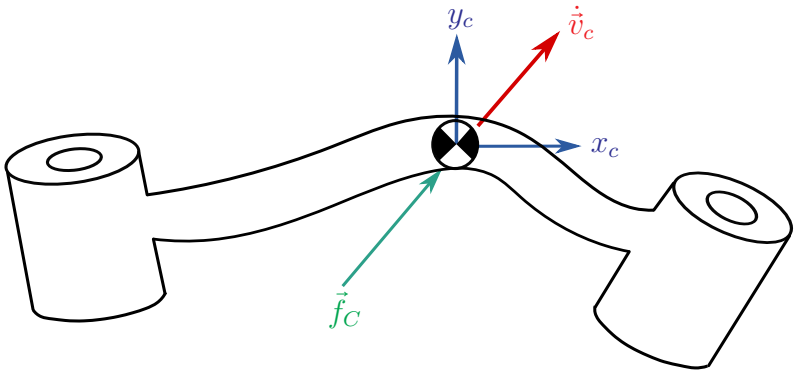


Figure 9.8: Newton-Euler Method: Newton Equation

Euler equation is: $\vec{n}_C = {}^C I\dot{\vec{\omega}}_C + \vec{\omega}_C \times {}^C I\vec{\omega}_C$
With:

$\vec{\omega}_C$ = Body‘s angular velocity
${}^C I$ = Inertia tensor in frame C (center of mass) C
$\vec{n}_C$ = Torque in center, causing the rotation

See figure 9.9.

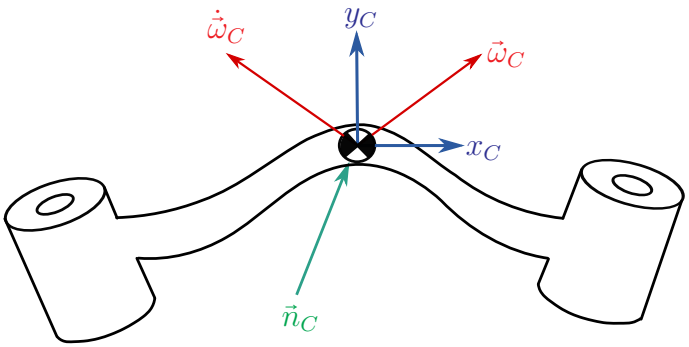


Figure 9.9: Newton-Euler Method: Euler Equation

9.5.1 Newton-Euler Method

Determination of velocities and acceleration in order to calculate the segments´ mass forces is carried out iteratively. Rotational velocity of element $i+1$ is:

$$\begin{aligned} {}^{i+1}\vec{\omega}_{i+1} &= {}^{i+1}_{i}R \cdot ({}^{i}\vec{\omega}_i + \dot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i}) \\ {}^{i}_{i+1}R \cdot {}^{i+1}\vec{\omega}_{i+1} &= {}^{i}\vec{\omega}_i + \dot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i} \end{aligned}$$

For rotational acceleration the following applies: (Rotation matrix ${}^{i}_{i+1}R$ dependent on $\vec{\theta}$ and thus time dependent.)

$$\begin{aligned} \frac{d}{dt}({}^{i}_{i+1}R \cdot {}^{i+1}\vec{\omega}_{i+1}) &= \frac{d}{dt}({}^{i}\vec{\omega}_i + \dot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i}) \\ (\frac{d}{dt}\,{}^{i}_{i+1}R) \cdot {}^{i+1}\vec{\omega}_{i+1} + {}^{i}_{i+1}R \cdot {}^{i+1}\dot{\vec{\omega}}_{i+1} &= {}^{i}\dot{\vec{\omega}}_i + \ddot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i} \quad \text{9.3 follows:} \\ \dot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i} \times {}^{i}_{i+1}R \cdot {}^{i+1}\vec{\omega}_{i+1} + {}^{i}_{i+1}R \cdot {}^{i+1}\dot{\vec{\omega}}_{i+1} &= {}^{i}\dot{\vec{\omega}}_i + \ddot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i} \\ {}^{i}_{i+1}R \cdot {}^{i+1}\dot{\vec{\omega}}_{i+1} &= {}^{i}\dot{\vec{\omega}}_i + \ddot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i} - \dot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i} \times {}^{i}_{i+1}R \cdot {}^{i+1}\vec{\omega}_{i+1} \\ {}^{i}_{i+1}R \cdot {}^{i+1}\dot{\vec{\omega}}_{i+1} &= {}^{i}\dot{\vec{\omega}}_i + \ddot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i} + {}^{i}\vec{\omega}_{i+1} \times \dot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i} \\ {}^{i+1}\dot{\vec{\omega}}_{i+1} &= {}^{i+1}_{i}R \cdot ({}^{i}\dot{\vec{\omega}}_i + \ddot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i} ({}^{i}\vec{\omega}_i + \dot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i}) \times \dot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i}) \end{aligned}$$

$$\boxed{{}^{i+1}\dot{\vec{\omega}}_{i+1} = {}^{i+1}_{i}R \cdot ({}^{i}\dot{\vec{\omega}}_i + \ddot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i} + {}^{i}\vec{\omega}_i \times \dot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i})}$$

Simplification for linear joints holds:

$$ {}^{i+1}\dot{\vec{\omega}}_{i+1} = {}^{i+1}_{i}R \cdot {}^{i}\dot{\vec{\omega}}_{i} $$

For linear velocity of element $i+1$:

$$ {}^{i+1}\vec{v}_{i+1} = {}^{i+1}_{i}R \cdot \left({}^{i}\vec{v}_{i} + {}^{i}\vec{\omega}_{i+1} \times {}^{i}\vec{u}_{i+1} + \dot{d}_{i+1} \cdot {}^{i}\vec{e}_{z_i} \right) $$

For linear acceleration in link origin:

$$ {}^{i+1}\dot{\vec{v}}_{i+1} = {}^{i+1}_{i}R \cdot \left({}^{i}\dot{\vec{v}}_{i} + \ddot{d}_{i+1} \cdot {}^{i}\vec{e}_{z_i} + {}^{i}\dot{\vec{\omega}}_{i+1} \times {}^{i}\vec{u}_{i+1} + {}^{i}\vec{\omega}_{i+1} \times \left({}^{i}\vec{\omega}_{i+1} \times {}^{i}\vec{u}_{i+1} \right) + 2 \cdot {}^{i}\vec{\omega}_{i+1} \times \left(\dot{d}_{i+1} \, {}^{i}\vec{e}_{z_i} \right) \right) $$

Simplification for revolute joint follows:

$$ {}^{i+1}\dot{\vec{v}}_{i+1} = {}^{i+1}_{i}R \cdot \left({}^{i}\dot{\vec{v}}_{i} + {}^{i}\dot{\vec{\omega}}_{i+1} \times {}^{i}\vec{u}_{i+1} + {}^{i}\vec{\omega}_{i+1} \times \left({}^{i}\vec{\omega}_{i+1} \times {}^{i}\vec{u}_{i+1} \right) \right) $$

Linear acceleration in center of mass is:

$$ {}^{i}\dot{\vec{v}}_{C_i} = {}^{i}\dot{\vec{v}}_{i} + {}^{i}\dot{\vec{\omega}}_{i} \times {}^{i}\vec{u}_{C_i} + {}^{i}\vec{\omega}_{i} \times \left({}^{i}\vec{\omega}_{i} \times {}^{i}\vec{u}_{C_i} \right) $$

Calculation of first link follows:

$$ {}^{0}\vec{\omega}_{0} = {}^{0}\dot{\vec{\omega}}_{0} = \vec{0} $$

With linear and angular accelerations in the centers of mass the following forces and torques result:

$$ \begin{aligned} \vec{f}_{C_i} &= m_i \cdot \dot{\vec{v}}_{C_i} \\ \vec{n}_{C_i} &= {}^{C_i}I \cdot \dot{\vec{\omega}}_{i} + \vec{\omega}_{i} \times {}^{C_i}I \cdot \vec{\omega}_{i} \end{aligned} $$

The calculation of forces and torques equilibrium for each link is done by considering its own mass force and inertia and taking into account the forces and torques enacted by neighboring segments.

$$ \begin{aligned} \vec{f}_{i} &= \text{Force enacted upon link } i \text{ by link } i+1 \\ \vec{n}_{i} &= \text{Torque enacted upon link } i \text{ by link } i+1 \end{aligned} $$

See figure 9.10.

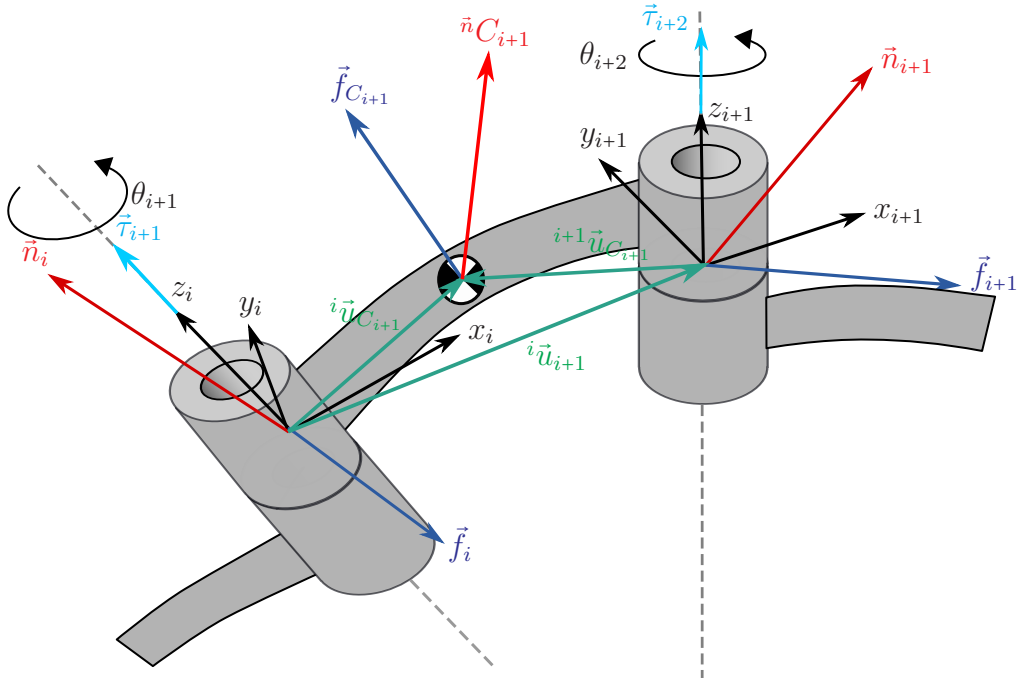


Figure 9.10: Coordinate Systems and Designators
Caution:
${}^{i+1}_{i}R \cdot {}^{i}\vec{u}_{C_{i+1}} \neq {}^{i+1}\vec{u}_{C_{i+1}}$
because they are position vectors

Force equilibrium in joint i results in:

$$ {}^{i}\vec{f}_i = {}^{i}\vec{f}_{C_{i+1}} + {}^{i}_{i+1}R \cdot {}^{i+1}\vec{f}_{i+1} $$

The following applies to Torque equilibrium:

$$ {}^{i}\vec{n}_i = {}^{i}\vec{n}_{C_{i+1}} + {}^{i}_{i+1}R \cdot {}^{i+1}\vec{n}_{i+1} + {}^{i}\vec{u}_{C_{i+1}} \times {}^{i}\vec{f}_{C_{i+1}} + {}^{i}\vec{u}_{i+1} \times {}^{i}_{i+1}R \cdot {}^{i+1}\vec{f}_{i+1} $$

The calculation proceeds therefore from the last joint to the base (backwards).

For calculation of the forces required in joint i, only the z component is used:

$$ \vec{\tau}_{i+1} = {}^{i}\vec{n}_i^T \cdot {}^{i}\vec{e}_{z_i}. $$

Linear force for linear joints:

$$ \vec{\tau}_{i+1} = {}^{i}\vec{f}_i^T \cdot {}^{i}\vec{e}_{z_i}. $$

In free space the initial forces and torques are set to 0:

$$ \vec{f}_N = \vec{n}_N = \vec{0}. $$

(If contact with environment or existing non-zero load)

9.5.2 Algorithm for Calculation of Torques

1. Iterative calculation of velocities and accelerations starting from first link (outer iteration)

$$
\begin{aligned}
{}^{i+1}\vec{\omega}_{i+1} &= {}^{i+1}_{i}R \cdot ({}^{i}\vec{\omega}_i + \dot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i}) \\
{}^{i+1}\dot{\vec{\omega}}_{i+1} &= {}^{i+1}_{i}R \cdot ({}^{i}\dot{\vec{\omega}}_i + \ddot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i} + {}^{i}\vec{\omega}_i \times \dot{\theta}_{i+1} \cdot {}^{i}\vec{e}_{z_i}) \\
{}^{i+1}\dot{\vec{v}}_{i+1} &= {}^{i+1}_{i}R \cdot ({}^{i}\dot{\vec{v}}_i + \ddot{d}_{i+1} \cdot {}^{i}\vec{e}_{z_i} + {}^{i}\dot{\vec{\omega}}_{i+1} \times {}^{i}\vec{u}_{i+1} \\
&\quad + {}^{i}\vec{\omega}_{i+1} \times ({}^{i}\vec{\omega}_{i+1} \times {}^{i}\vec{u}_{i+1}) + 2 \cdot {}^{i}\vec{\omega}_{i+1} \times (\dot{d}_{i+1} \cdot {}^{i}\vec{e}_{z_i}) \\
{}^{i}\dot{\vec{v}}_{C_i} &= {}^{i}\dot{\vec{v}}_i + {}^{i}\dot{\vec{\omega}}_i \times {}^{i}\vec{u}_{C_i} + {}^{i}\vec{\omega}_i \times ({}^{i}\vec{\omega}_i \times {}^{i}\vec{u}_{C_i}) \\
{}^{i}\vec{f}_{C_i} &= m_i \cdot {}^{i}\dot{\vec{v}}_{C_i} \\
{}^{i}\vec{n}_{C_i} &= {}^{C_i}I \cdot {}^{i}\dot{\vec{\omega}}_i + {}^{i}\vec{\omega}_i \times {}^{C_i}I \cdot {}^{i}\vec{\omega}_i
\end{aligned}
$$

   If gravity is considered, then the following applies: ${}^{0}\dot{\vec{v}}_0 = \vec{g}'$, where $\vec{g}'$ (the acceleration in the direction of the gravitational axis) is opposite to the gravitational vector. This corresponds to an acceleration of the robot base with 1g ($9.81 m/s^2$) upwards.

2. Backward calculation of forces and torques starting from last link and ending in robot base (inner iteration):

$$
\begin{aligned}
{}^{i}\vec{f}_i &= {}^{i}\vec{f}_{C_{i+1}} + {}^{i}_{i+1}R \cdot {}^{i+1}\vec{f}_{i+1} \\
{}^{i}\vec{n}_i &= {}^{i}\vec{n}_{C_{i+1}} + {}^{i}_{i+1}R \cdot {}^{i+1}\vec{n}_{i+1} + {}^{i}\vec{u}_{C_{i+1}} \times {}^{i}\vec{f}_{C_{i+1}} + {}^{i}\vec{u}_{i+1} \times {}^{i}_{i+1}R \cdot {}^{i+1}\vec{f}_{i+1} \\
\tau_{i+1} &= {}^{i}\vec{n}_i^{T} \cdot {}^{i}\vec{e}_{z_i} \quad \text{or} \quad \tau_{i+1} = {}^{i}\vec{f}_i^{T} \cdot {}^{i}\vec{e}_{z_i}
\end{aligned}
$$

Example:

The two-jointed robot is an example of a closed-form solution (see figure 9.11). For simplification point masses m_1 and m_2 are at the joint centers.

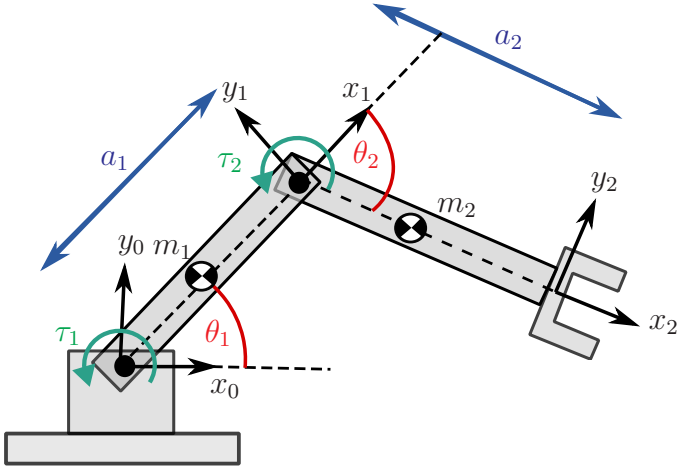


Figure 9.11: Newton Euler Method: Example

Procedure:

1. Determining known values
2. Determining rotation matrices between links
3. Outer iteration (velocity, acceleration)
   - For joint 1 and 2
4. Inner iteration (forces, torques)
   - For joint 2 and 1

To Determining known values:

- Vectors to centers of mass are:

$$\begin{aligned} {}^1\vec{u}_{C_1} &= -\frac{a_1}{2} \cdot {}^1\vec{e}_{x_1} \\ {}^2\vec{u}_{C_2} &= -\frac{a_2}{2} \cdot {}^2\vec{e}_{x_2} \end{aligned}$$

- Inertia tensor (because of point mass) is:

$${}^{C_1}I_1 = {}^{C_2}I_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- With no forces acting on TCP:

$$\begin{aligned} \vec{f}_2 &= \vec{0} \\ \vec{n}_2 &= \vec{0} \end{aligned}$$

- With no movement of robot base:

$$\begin{aligned} \vec{\omega}_0 &= \vec{0} \\ \dot{\vec{\omega}}_0 &= \vec{0} \end{aligned}$$

- With consideration of gravity:

$${}^0\dot{\vec{v}}_0 = g \cdot \vec{e}_{y_0}$$

- Vector to next coordinate system are:

$$\begin{aligned} {}^0\vec{u}_1 &= \begin{bmatrix} c_1 a_1 \\ s_1 a_1 \\ 0 \end{bmatrix} \\ {}^0\vec{u}_2 &= \begin{bmatrix} c_2 a_2 \\ s_2 a_2 \\ 0 \end{bmatrix} \end{aligned}$$

Rotation matrices between joint-frames (see chapter 8):

$$
\begin{aligned}
{}^{i}_{i+1}R &= \begin{bmatrix} \cos(\theta_{i+1}) & -\sin(\theta_{i+1}) & 0 \\ \sin(\theta_{i+1}) & \cos(\theta_{i+1}) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_{i+1} & -s_{i+1} & 0 \\ s_{i+1} & c_{i+1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
{}^{i+1}_{i}R &= \begin{bmatrix} \cos(\theta_{i+1}) & \sin(\theta_{i+1}) & 0 \\ -\sin(\theta_{i+1}) & \cos(\theta_{i+1}) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_{i+1} & s_{i+1} & 0 \\ -s_{i+1} & c_{i+1} & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

To outer iteration (1st step):

$$
\begin{aligned}
{}^{1}\vec{\omega}_1 &= {}^{1}_{0}R \cdot ({}^{0}\vec{\omega}_0 + \dot{\theta}_1 \cdot {}^{1}\vec{e}_{z_1}) = \vec{0} + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} \\
{}^{1}\dot{\vec{\omega}}_1 &= {}^{1}_{0}R \cdot ({}^{0}\dot{\vec{\omega}}_0 + \ddot{\theta}_1 \cdot {}^{0}\vec{e}_{z_0} + {}^{0}\vec{\omega}_0 \times \dot{\theta}_1 \cdot {}^{1}\vec{e}_{z_0}) = \vec{0} + \begin{bmatrix} 0 \\ 0 \\ \ddot{\theta}_1 \end{bmatrix} + \vec{0} \\
{}^{1}\dot{\vec{v}}_1 &= {}^{1}_{0}R \cdot ({}^{0}\dot{\vec{v}}_0 + {}^{0}\vec{\omega}_1 \times {}^{0}\vec{u}_1 + {}^{0}\vec{\omega}_1 \times ({}^{0}\vec{\omega}_1 \times {}^{0}\vec{u}_1)) \\
&= \begin{bmatrix} gs_1 \\ gc_1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ a_1 \cdot \ddot{\theta}_1 \\ 0 \end{bmatrix} + \begin{bmatrix} a_1 \cdot -\dot{\theta}_1^2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} gs_1 - a_1 \cdot \dot{\theta}_1^2 \\ gc_1 - a_1 \cdot \ddot{\theta}_1 \\ 0 \end{bmatrix} \\
{}^{1}\dot{\vec{v}}_{C_1} &= {}^{1}\dot{\vec{v}}_1 + {}^{1}\dot{\vec{\omega}}_1 \times {}^{1}\vec{u}_{C_1} + {}^{1}\vec{\omega}_1 \times ({}^{1}\vec{\omega}_1 \times {}^{1}\vec{u}_{C_1}) \\
&= \begin{bmatrix} gs_1 - a_1 \cdot \dot{\theta}_1^2 \\ gc_1 - a_1 \cdot \ddot{\theta}_1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{a_1}{2} \cdot -\ddot{\theta}_1 \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{a_1}{2} \cdot \dot{\theta}_1^2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} gs_1 - \frac{a_1}{2} \cdot \dot{\theta}_1^2 \\ gc_1 + \frac{a_1}{2} \cdot \ddot{\theta}_1 \\ 0 \end{bmatrix} \\
{}^{1}\vec{f}_{C_1} &= m_1 \cdot {}^{1}\dot{\vec{v}}_{C_1} \\
{}^{1}\vec{n}_{C_1} &= {}^{C_1}I \cdot \dot{\vec{\omega}}_{C_1} \times {}^{C_1}I \cdot \vec{\omega}_{C_1} = \vec{0} + \vec{0}
\end{aligned}
$$

Recursive description using the Newton-Euler method:

- ☺ Arbitrary number of joints
- ☺ Loads on links are calculated
- ☺ Small computational effort $O(n)$ (n=number of joints)
- ☹ Recursion

9.6 Dynamics Calculation

9.6.1 Lagrange Method

Equation of movement according to Lagrange is:

$$\tau_i = \frac{d}{dt}\frac{\partial l}{\partial \dot{\theta}_i} - \frac{\partial l}{\partial \theta_i}$$

With:

θ_i = Rotation angle or translation distance
$\dot{\theta}_i$ = Joint velocities
τ_i = force/torque vector in joint i

Lagrange function $L = E_{kin} - E_{pot}$ (in reference to base) deescribes the difference between kinetic and potential energy of a mechanical system.

Kinetic Energy

Kinetic energy $E_{kin,i}$ of joint i is:

$$E_{kin,i} = \underbrace{\frac{1}{2} m_i \cdot \vec{v}_{C_i}^T \cdot \vec{v}_{C_i}}_{\text{Linear portion}} + \underbrace{\frac{1}{2} \, {}^i\vec{\omega}_i^T \cdot \, {}^{C_i}I_i \cdot \, {}^i\vec{\omega}_i}_{\text{Rotational portion}}$$

$\vec{v}_{C_i}$ and ${}^i\vec{\omega}_i$ are dependent on position and velocity of joints. The total kinematic energy E_{kin} results from:

$$E_{kin} = \sum_{i=1}^{n} E_{kin,i}$$

Kinetic energy can be described dependent on position and velocity:

$$E_{kin}(\vec{\theta}, \dot{\vec{\theta}}) = \frac{1}{2} \dot{\vec{\theta}}^T \cdot M(\vec{\theta}) \cdot \dot{\vec{\theta}}$$

$M(\vec{\theta})$ here is an $n \times n$ mass matrix, in which every element is a complex function depending on $\vec{\theta}$. This is a positive-definite matrix, thus $\dot{\vec{\theta}}^T \cdot M(\vec{\theta}) \cdot \dot{\vec{\theta}}$ always yields a positive scalar. This equation corresponds to the common formulation of kinetic energy of a point mass:

$$E_{kin} = \frac{1}{2} m \cdot v^2$$

Potential Energy

Potential energy u_i of link i is:

$$E_{pot,i} = -m_i \cdot \, {}^0\vec{g}^T \cdot \, {}^0\vec{u}_{C_i} + E_{pot,ref_i}$$

With:

${}^0\vec{g}$	3×1 Gravitation vector, in reference to frame 0
${}^0\vec{u}_{C_i}$	3×1 Vector, describing the center of mass of$i$ (dependent on joint position)
E_{pot,ref_i}	Constant, so $E_{pot,i} \geq 0$ holds.

The total potential energy E_{pot} is given by

$$E_{pot} = \sum_{i=1}^{n} E_{pot,i}$$

The potential energy can also be formulated as a function $E_{pot}(\vec{\theta})$ in dependence of the joint values.

Lagrange Method

Thus, for the Lagrange function follows:

$$l(\vec{\theta}, \dot{\vec{\theta}}) = E_{kin}(\vec{\theta}, \dot{\vec{\theta}}) - E_{pot}(\vec{\theta})$$

For the equation of movement with torque vector $\vec{\tau}$:

$$\vec{\tau} = \frac{d}{dt}\frac{\partial l}{\partial \dot{\vec{\theta}}} - \frac{\partial l}{\partial \vec{\theta}}$$

For a manipulator follows:

$$\vec{\tau} = \frac{d}{dt}\frac{\partial E_{kin}(\vec{\theta}, \dot{\vec{\theta}})}{\partial \dot{\vec{\theta}}} - \frac{\partial E_{kin}(\vec{\theta}, \dot{\vec{\theta}})}{\partial \vec{\theta}} + \frac{\partial E_{pot}(\vec{\theta})}{\partial \vec{\theta}}$$

Analytical description using the Lagrange method:

- ☺ Formulating the equations is simple
- ☺ It is a closed model
- ☺ Analytical evaluation is possible
- ☹ Computationally very expensive, in complexitx class $\mathcal{O}(n^4)$ (n = number of joints)
- ☹ Only actuating torques are calculated

9.7 Comparison of approaches

9.7.1 Efficiency of the Approaches

Newton-Euler method

- Multiplications: $126n - 99$
- Additions: $106n - 92$

Lagrange method

- Multiplications: $32n^4 + 86n^3 + 171n^2 + 53n - 128$
- Additions: $25n^4 + 66n^3 + 129n^2 + 42n - 96$

For typical robots ($n = 6$ joints) the Newton-Euler method is 100× more efficient than lagrange method. For both methods optimizations are possible.

Requirements for Manipulators

- Reliable positioning : Accuracy (repeatability)
- Collision avoidance
- Execution of movement: Fluid with appropriate velocities and accelerations
- Adaptation to changing conditions

9.7.2 Fundamental Questions

Direct kinematics: Given all joint values. Where is the TCP?

Inverse kinematics: Given TCP-pose. Which joint values are required to achieve pose?

Dynamics: Which forces/torques do the actuators have to enact to accelerate TCP by a certain magnitude?

Trajectory planning: How does a „good“ trajectory that avoids collisions look like?

10. Continuous Path Control and Interpolation

10.1 Basics of Continuous Path Control

Movement of the robot are interpreted as state changes with respect to time (trajectory) relative to a stationary coordinate system (Cartesian space, joint space), whereby quality criteria, secondary and boundary problems as well as constraints must often are considered.

The pose of the manipulator at the start time (in Cartesian coordinates y_{start} or in the configuration space θ_{start}), and the pose of the manipulator at the target time (y_{target} or θ_{target}) are given.

We are looking for the trajectory that transfers the manipulator from the starting point to the end point.

10.2 Types of Planning

There are two types of planning:

- **PTP: Point to Point**
  - Planning of movement in configuration space
  - Time optimal path
  - Cartesian path not known
  - Use cases: Spot welding, handling tasks, ...
- **CP: Continuous Path**
  - Path control in Cartesian space
  - Path can be adapted to a desired shape

- Path out of the work space is possible
- Overstepping limits of the joints is possible
- Use cases: Path welding, laser cutting, varnishing

10.2.1 PTP: Movement Phase of Joints

There are three stages in the movement of each joint:

1. Acceleration of the joint
2. Movement with maximal or desired velocity
3. Slowing down, resting in target pose

Phases of Planning:

- Calculation of the necessary change of evry actuating variable $\theta_{target} - \theta_{start}$
- Determination of the acceleration and deceleration duration
- Calculating the time, for which the joint will move at maximal velocity (omitted if change is to small to reach maximal velocity)
- Generating the trajectory

The specified actuating variables can be obtained, for example, through teach-in, direct specification or the result of the inverse kinematics.

Types of Synchronization

Asynchronous: Planning of axes independently

Synchronous: Movement of all axes starts and ends simultaneously. The slowest joint (leading axle) serves as a reference.The advantage is only little stress on mechanics

Fully synchronous: Acceleration and deceleration are simultaneous. The advantage is the smoother movement in Cartesian space. The disadvantage is that the acceleration has to be specified.

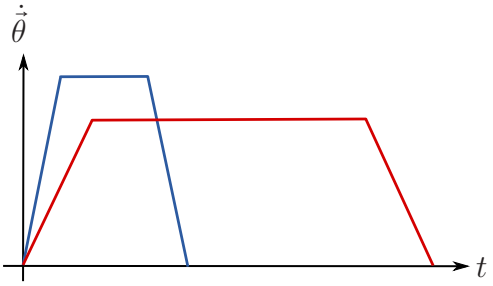


Figure 10.1: Asynchronous PTP

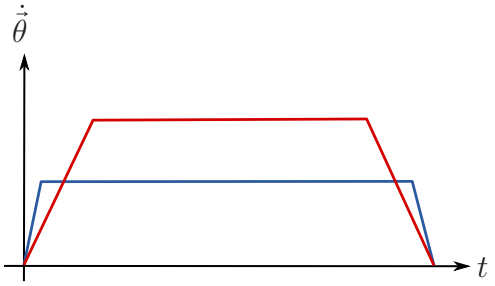


Figure 10.2: Synchronous PTP

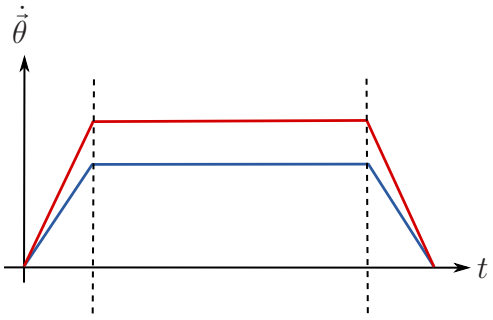


Figure 10.3: Fully Synchronous PTP

10.2.2 PTP: Point-to-Point Control

Advantages of the PTP are on the one hand the calculation of the manipulated variable trajectory is simple and does not cause any problems with singularities. On the other hand, the simple consideration of robot specific constraint such as joint angle limitation maximal speed and acceleration, are easy to consider. The time-optimized movement path is also an advantage.
However, the exact Cartesian path is difficult to foresee.

PTP: Constraints

Start and target state are known:

$$\vec{\theta}(t_{start}) = \vec{\theta}_{start}$$
$$\vec{\theta}(t_{target}) = \vec{\theta}_{target}$$

Velocity is zero at start and end:

$$\dot{\vec{\theta}}(t_{start}) = \vec{0}$$
$$\dot{\vec{\theta}}(t_{target}) = \vec{0}$$

Working space, velocity and acceleration are limited:

$$\theta_{min} \leq \theta(t_j) \leq \theta_{max}, \quad 0 \leq \dot{\theta}(t_j) \leq \dot{\theta}_{max}, \quad \ddot{\theta}_{min} \leq \ddot{\theta}(t_j) \leq \ddot{\theta}_{max}$$

The limits can also be selected independently of the mechanics, for example to achieve fast acceleration during parallel slow decleration.

Phases of Control

General path parameter $s(t)$, describes the distance which could be covered for linear joints and the angle which should be rotated for rotatinal joints.

Given:

- General parameters $s(t)$, $v(t)$, and $a(t)$.
- Maximal velocity v_{max} and acceleration a_{max}.
- Start and target position θ_{start}, θ_{target}.

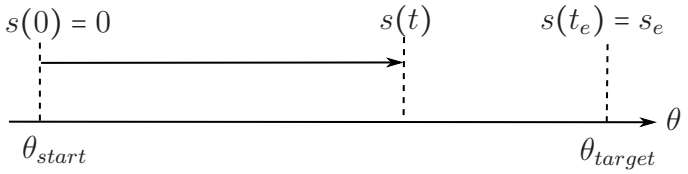


Figure 10.4: Phases of Control

$$\begin{aligned} s(0) &= \dot{s}(0) = v(0) = 0 \\ \dot{s}(t_e) &= v(t_e) = 0 \end{aligned}$$

procedure:

Step 1: Calculation of path which should be covered s_e for each joint

$$s_e = |\theta_{target} - \theta_{start}|$$

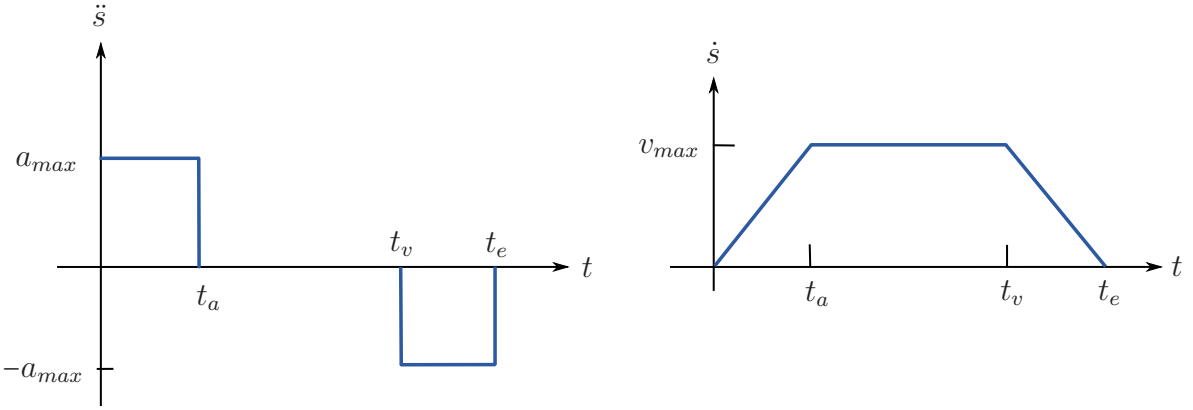
Step 2: Modification of inputs v_{max} and a_{max} for synchronous or fully synchronous PTP.

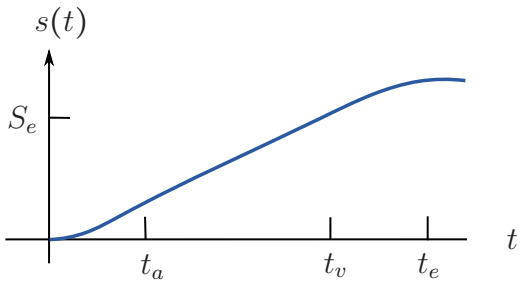
Step 3: Calculation of in-motion time t_e, acceleration time t_a and start of deceleration time t_v

Step 4: Interpolation: Calculation of intermediate points $s(t)$, $\dot{s}(t)$, $\ddot{s}(t)$

Step 5: Determination of reference values $\theta(t)$, $\dot{\theta}(t)$, $\ddot{\theta}(t)$

Square Wave Graphs for Interpolation





$$\begin{aligned} s_e &= |\theta_{target} - \theta_{start}| \\ t_a &= \frac{v_{max}}{a_{max}} \\ t_e &= \frac{s_e}{v_{max}} + t_a \\ t_v &= t_e - t_a \end{aligned}$$

Calculation of Parameters:

Acceleration time:

$$t_a = \frac{v_{max}}{a_{max}}$$

Integration of velocities:

$$s_e = s(t_e) = v_{max} \cdot t_a + v_{max} \cdot (t_v - t_a) = v_{max} \cdot t_a + v_{max} \cdot (t_e - 2 \cdot t_a)$$

Calculation of in-motion time:

$$t_e = \frac{s_e}{v_{max}} + t_a = \frac{s_e}{v_{max}} + \frac{v_{max}}{a_{max}}$$

Parameter for PTP:

$$0 \le t \le t_a: \quad \ddot{s}(t) = a_{max}, \quad \dot{s}(t) = a_{max} \cdot t, \quad s(t) = \frac{1}{2} \cdot a_{max} \cdot t^2$$

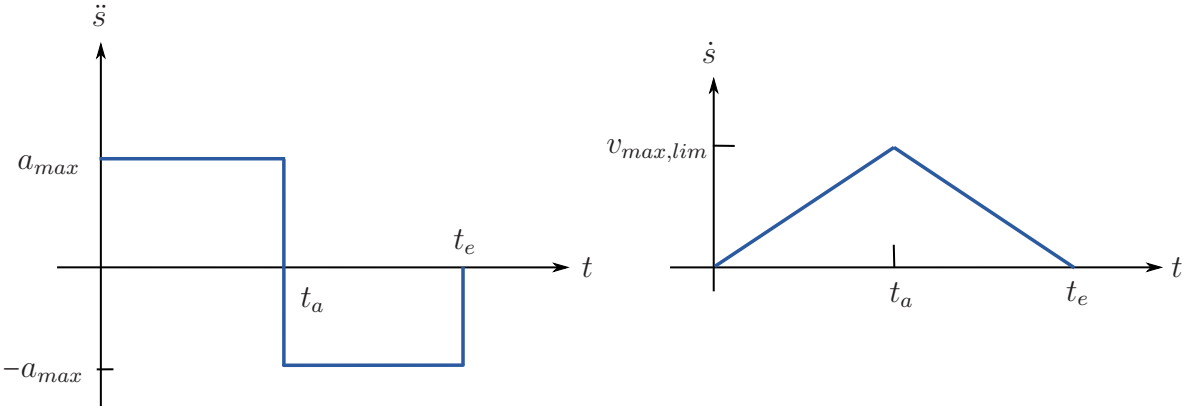
$$t_a \le t \le t_v: \quad \ddot{s}(t) = 0, \quad \dot{s}(t) = v_{max}, \quad s(t) = v_{max} \cdot t - \frac{1}{2} \cdot \frac{v_{max}^2}{a_{max}}$$

$$t_v \le t \le t_e: \quad \ddot{s}(t) = -a_{max}, \quad \dot{s}(t) = v_{max} - a_{max} \cdot (t - t_v), \quad (t) = v_{max} \cdot (t_e - t_a) - \frac{a_{max}}{2} \cdot (t_e - t)^2$$

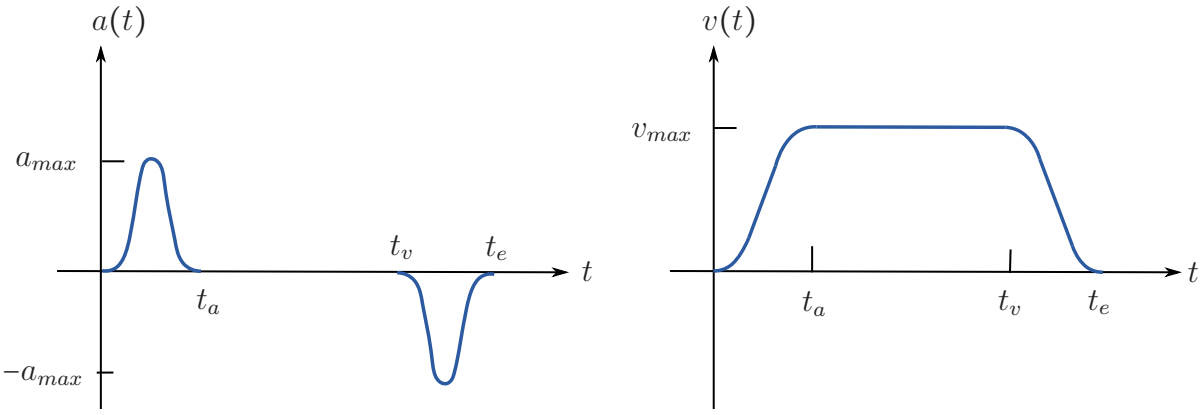
Time Optimal Path

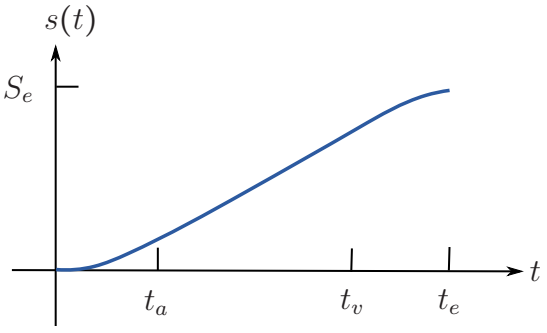
If v_{max} is too big compared to acceleration and path length, a time optimal path can be calculated by:

$$s_e = t_a \dot{v}_{max,lim} = \frac{v_{max,lim}^2}{a_{max}} \Rightarrow \sqrt{a_{max} \cdot s_e} \le v_{max}$$



Sine Wave Graphs for PTP-Control





$$
\begin{aligned}
s_e &= |\theta_{target} - \theta_{start}| \\
t_a &= \frac{2 \cdot v_{max}}{a_{max}} \\
t_e &= \frac{s_e}{v_{max}} + t_a \\
t_v &= t_e - t_a
\end{aligned}
$$

$$\begin{aligned} s_e &= |\theta_{target} - \theta_{start}| \\ t_a &= \frac{2 \cdot v_{max}}{a_{max}} \\ t_e &= \frac{s_e}{v_{max}} + t_a \\ t_v &= t_e - t_a \end{aligned}$$

$$\ddot{s}(t) = a_{max} \cdot \sin(\frac{\pi}{t_a} \cdot t) \tag{10.1}$$

Integrating 10.1 over time yields the velocity:

$$\dot{s}(t) = a_{max} \cdot (\frac{1}{2} \cdot t - \frac{t_a}{4 \cdot \pi} \cdot \sin(\frac{2 \cdot \pi}{t_a} \cdot t)) \tag{10.2}$$

For $t = t_a$ one gets v_{max} and 10.2 yields:

$$t_a = \frac{2 \cdot v_{max}}{a_{max}} \tag{10.3}$$

Covered path or angles during acceleration time can be calculated by integrating 10.2:

$$s(t) = a_{max} \cdot (\frac{1}{4} \cdot t^2 + \frac{t_a^2}{8 \cdot \pi} \cdot (\cos(\frac{2 \cdot \pi}{t_a} \cdot t) - 1)) \tag{10.4}$$

over the whole covered path or angle distance

$$\begin{aligned} s_e &= 2 \cdot s(t_a) + v_{max} \cdot (t_e - 2 \cdot t_a), \\ s(t_a) &= \frac{1}{4} \cdot a_{max} \cdot t_a^2 = \frac{v_{max}^2}{a} \end{aligned}$$

$$t_e = \frac{s_e}{v_{max}} + \frac{2 \cdot v_{max}}{a_{max}} = \frac{s_e}{v_{max}} + t_a \tag{10.5}$$

During phase of uniform movement one gets:

$$\dot{s}(t) = v_{max}$$

$$s(t) = s(t_a) + v_{max} \cdot (t - t_a) = v_{max} \cdot (t - \frac{1}{2} \cdot t_a) \tag{10.6}$$

Velocity and path during deceleration will be:

$$\dot{s}(t) = v_{max} - \int_{t-t_v}^{t} a(\tau - t_v) \cdot d\tau = v_{max} - a_{max} \cdot (\frac{1}{2} \cdot (t - t_v) - \frac{t_a}{4 \cdot \pi} \cdot \sin(\frac{2 \cdot \pi}{t_a} \cdot (t - t_v)))$$

$$\begin{aligned} s(t) &= s(t_v) + \int_{t-t_v}^{t} \dot{s}(\tau - t_v) \cdot d\tau \qquad (10.7) \\ &= \frac{a_{max}}{2} \cdot [t_e \cdot (t - t_a) - \frac{t^2 + t_e^2 + 2 \cdot t_a^2}{2} + \frac{t_a^2}{4 \cdot \pi^2} \cdot (1 - \cos(\frac{2 \cdot \pi}{t_a} \cdot (t - t_v)))] \end{aligned}$$

Synchronous PTP

Approach:

1. Determine path length $s_{e,i}$ for each joint i
2. Determine PTP-parameter $v_{max,i}$, $a_{max,i}$
3. Calculate time in-motion $t_{e,i}$
4. Determine axes with maximal time in-motion $t_e = t_{e,max} = max_i(t_{e,i})$
   Determined axle is leading axle
5. Set $max_i(t_{e,i}) = t_e$ for all joints
6. Calculate new velocities for each joint

Transformation of time in-motion t_e and calculation of new velocities
Graphs:

$$t_e = \frac{s_{e,i}}{v_{max,i}} + \frac{v_{max,i}}{a_{max,i}}$$

After transformation one gets:

$$v_{max,i}^2 - v_{max,i} \cdot a_{max,i} \cdot t_e + s_{e,i} \cdot a_{max,i} = 0$$

Solution is the smaller value since else $2 \cdot t_{a,i} > t_e$ and

$$v_{max,i} = \frac{a_{max,i} \cdot t_e}{2} - \sqrt{\frac{a_{max,i}^2 \cdot t_e^2}{4} - s_{e,i} \cdot a_{max,i}}$$

Sine wave path will be:

$$v_{max,i} = \frac{a_{max,i} \cdot t_e}{4} - \sqrt{\frac{a_{max,i}^2 \cdot t_e^2 - 8 \cdot s_{e,i} \cdot a_{max,i}}{16}}$$

Fully Synchronous PTP

Takes acceleration and deceleration times into account. Determination of leading axle with t_e and t_a and $t_v = t_e - t_a$, and determination of velocity and acceleration of the other axes via $v_{max,i} = \frac{s_{e,i}}{t_v}$ and $a_{max,i} = \frac{v_{max,i}}{t_a}$.

10.3 Path Control

10.3.1 Continuous Path (CP) Control in Cartesian Space

The trajectory is specified as a function of the the position of TCP. (e.g. with the description vector of the TCP: $\vec{y}_{TCP}(t), \dot{\vec{y}}_{TCP}(t), \ddot{\vec{y}}_{TCP}(t)$). Function e.g. linear, polynomial or spline path.

One advantage is that the definition of the trajectory explicitly takes place in cartesian space. Another advantages is that planning is independent of robot kinematics.

Disadvantages however are, in one hand, that calculation of transformation to joint angles for each point of the trajectory is needed. On the other hand, planned trajectory cannot always executable (limits of working space, singularities of the robot). Constraints of joints can not be taken into account either.

The procedure for path control in Cartesian space can be seen in figure 10.5.

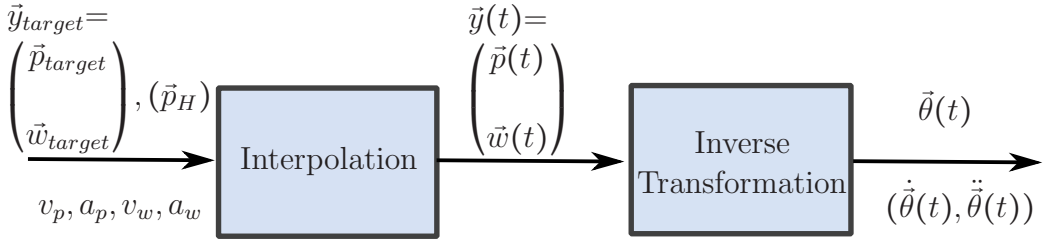


Figure 10.5: Path Control in Cartesian Space

Given:

Target position	$\vec{p}_{target} = (x_{target}, y_{target}, z_{target})^T$
Target orientation (Euler)	$\vec{w}_{target} = (\alpha_{target}, \beta_{target}, \gamma_{target})^T$
Intermediate point (optional)	$\vec{p}_H = (x_H, y_H, z_H)^T$
Linear velocity and acceleration	v_p, a_p
Rotational velocity and acceleration	v_w, a_w

Constraints are maximal velocity and acceleration of each joint.

10.3.2 Linear Interpolation

Path parameter $s_p(t)$ describes covered path at time t. The complete path is computed by:

$$s_{ep} = |\vec{p}_{target} - \vec{p}_{start}| = \sqrt{(x_{target} - x_{start})^2 + (y_{target} - y_{start})^2 + (z_{target} - z_{start})^2}$$

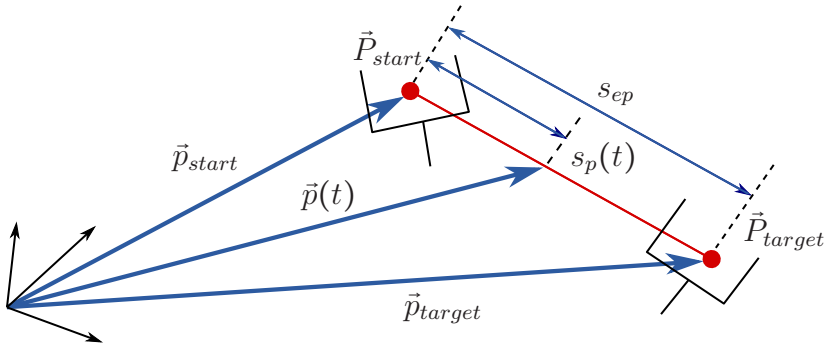


Figure 10.6: Linear Interpolation

Constraints:

$$\begin{aligned} s_p(0) &= \dot{s}_p(0) = v_p(0) = 0 \\ \dot{s}_p(t_e) &= v_p(t_e) = 0 \end{aligned}$$

With

$$\begin{aligned} v_{max} &= v_p, \quad a_{max} = a_p \\ t_e &= t_{ep}, \quad t_a = t_{ap}, \quad t_v = t_{vp} \\ s_e &= s_{ep}, \quad s = s_p \end{aligned}$$

$s_p(t)$ can be calculated from the already described equations in PTP, via sine or square wave. The current position of the TCP at time t results from:

$$\vec{p}(t) = \vec{p}_{start} + s_p(t) \cdot \frac{(\vec{p}_{target} - \vec{p}_{start})}{s_{ep}}$$

The change in orientation is calculated in the same way as for calculating the change in position. The complete change in orientation is calculated by:

$$s_{ew} = |\vec{w}_{target} - \vec{w}_{start}| = \sqrt{(\alpha_{target} - \alpha_{start})^2 + (\beta_{target} - \beta_{start})^2 + (\gamma_{target} - \gamma_{start})^2}$$

It makes sense that the position and orientation changes are completed at the same time. To do this, times in-motion must be adapted to the maximal and velocity be reduced accordingly.

$$t_e = max(t_{ep}, t_{ew})$$

For a robot control the calculated Cartesian poses must be transformed to joint angles at every sampling interval.

10.3.3 Circular Interpolation

Apart from lines often times also an arc of a circle can be used as parts of a path. Simplest model of an arc of a circle via a start point $\vec{P}_{Start}$, an end point $\vec{P}_{target}$, and an Intermediate point $\vec{P}_H$. From the three points, the center point M, radius r and angle ϕ_z can be determined by the intersection of the median perpendicular bisectors. See figure 10.7.

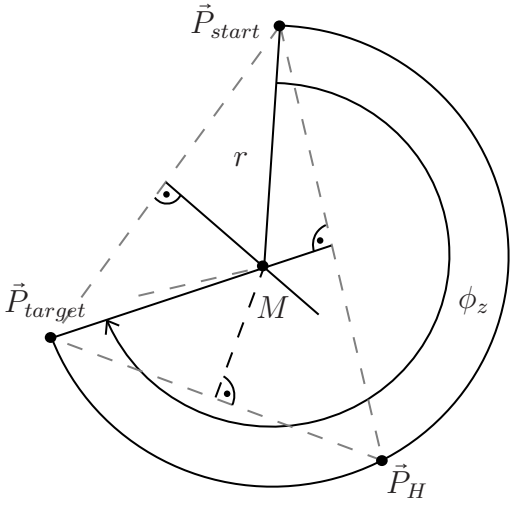


Figure 10.7: Circular Interpolation

Path parameter $s(t)$ describes covered angle $\varphi(t)$ (see figure 10.8). This can be calculated as in linear CP with equations from PTP. To calculate Cartesian position will introduce subsidiary coordinate system XYZ_Z.

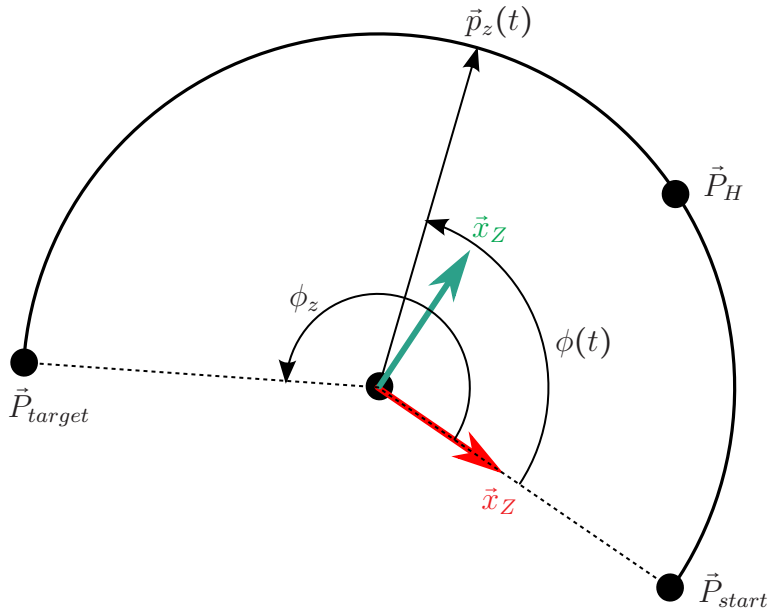


Figure 10.8: Circular Interpolation

Position $\vec{p}_z(t)$ on the arc of the circle XYZ_Z can be computed with r and $\varphi(t)$:

$$\vec{p}_z(t) = \begin{pmatrix} r\cos(\varphi(t)) \\ r\sin(\varphi(t)) \\ 0 \end{pmatrix}$$

$\vec{p}_z(t)$ can be transformed homogeneously in the BCS. The rientation is interpolated in the same way as with linear interpolation. For a robot control the calculated Cartesian poses must be transformed to joint angles at every sampling interval.

10.3.4 Spline-Interpolation

If the trajectory of a kinematic system consists of linear segments, very high accelerations occur at each start and end point of a segment. In order to avoid the associated jerk and the high forces on the kinematics, it is necessary either to decelerate at each segment end or to smooth the paths accordingly. The following section will deal with how to generate such smooth trajectories. An elegant solution is to model the path as a *spline*. A suitable spline consisting of the segments $S_0 \dots S_{n-1}$ would have to traverse the points $P_0 \dots P_n$ of the trajectory, where the derivatives at the end of a segment would have to be identical to the derivative of the segment following it (see figure 10.9). Usually, such splines are interpolated by means of polynomials of degree n.

Figure 10.9: The points P_0 to P_n are connected by means of the spline segments S_0 to S_{n-1}.

10.3.5 Piecewise Interpolation

Path is defined by piecewise polynomials, called *splines*. Cubical splines are the common case:

$$\vec{p}(t) = \vec{a}_3 \cdot t^3 + \vec{a}_2 \cdot t^2 + \vec{a}_1 \cdot t + \vec{a}_0$$

Where $\vec{p}(t)$ describes the path between the position vectors $\vec{p}_{start}$ and $\vec{p}_{target}$, with a time duration of t_e. Four conditions are needed to calculate the parameters $\vec{a}_j$ of a spline $\vec{p}(t)$ uniquely. Two conditions are described by the interpolation at the supporting points:

$$\begin{aligned} \vec{p}(t=0) &= \vec{p}_{start} \\ \vec{p}(t=t_e) &= \vec{p}_{target} \end{aligned}$$

The two remaining conditions can be determined by the desired velocity vectors:

$$\begin{aligned} \dot{\vec{p}}(t=0) &= \dot{\vec{p}}_{start} \\ \dot{\vec{p}}(t=t_e) &= \dot{\vec{p}}_{target} \end{aligned}$$

Calculating the parameters from the described conditions yields:

$$\begin{aligned} \vec{a}_0 &= \vec{p}_{start} \\ \vec{a}_1 &= \dot{\vec{p}}_{start} \\ \vec{a}_2 &= \frac{3}{t_e^2}(\vec{p}_{target} - \vec{p}_{start}) - \frac{1}{t_e}(\dot{\vec{p}}_{target} + 2\dot{\vec{p}}_{start}) \\ \vec{a}_3 &= -\frac{2}{t_e^3}(\vec{p}_{target} - \vec{p}_{start}) + \frac{1}{t_e^2}(\dot{\vec{p}}_{target} + \dot{\vec{p}}_{start}) \end{aligned}$$

Example:

Given $\vec{p}_I = \begin{pmatrix}0\\0\end{pmatrix}, \vec{p}_{II} = \begin{pmatrix}1\\2\end{pmatrix}, \vec{p}_{III} = \begin{pmatrix}2\\1\end{pmatrix}, \vec{p}_{IV} = \begin{pmatrix}4\\4\end{pmatrix}$,

$\dot{\vec{p}}_I = \cdots = \dot{\vec{p}}_{IV} = \begin{pmatrix}1\\0\end{pmatrix}$ and $t_e = 1$

Solution:
Parameter for the first polynomial, for the others analogously:

$$
\begin{aligned}
\vec{a}_0 &= \vec{p}_I \quad \vec{a}_2 = \frac{3}{1}(\vec{p}_{II} - \vec{p}_I) - \frac{1}{1}(\dot{\vec{p}}_{II} + 2\dot{\vec{p}}_I) \\
\vec{a}_1 &= \dot{\vec{p}}_I \quad \vec{a}_3 = -\frac{2}{1}(\vec{p}_{II} - \vec{p}_I) + \frac{1}{1}(\dot{\vec{p}}_{II} + \dot{\vec{p}}_I) \\
\Rightarrow \vec{p}_1(t) &= \begin{pmatrix} 0 \\ -4 \end{pmatrix} \cdot t^3 + \begin{pmatrix} 0 \\ 6 \end{pmatrix} \cdot t^2 + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \cdot t + \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\
\vec{p}_2(t) &= \begin{pmatrix} 0 \\ 2 \end{pmatrix} \cdot t^3 + \begin{pmatrix} 0 \\ -3 \end{pmatrix} \cdot t^2 + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \cdot t + \begin{pmatrix} 1 \\ 2 \end{pmatrix} \\
\vec{p}_3(t) &= \begin{pmatrix} -2 \\ -6 \end{pmatrix} \cdot t^3 + \begin{pmatrix} 3 \\ 9 \end{pmatrix} \cdot t^2 + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \cdot t + \begin{pmatrix} 2 \\ 1 \end{pmatrix}
\end{aligned}
$$

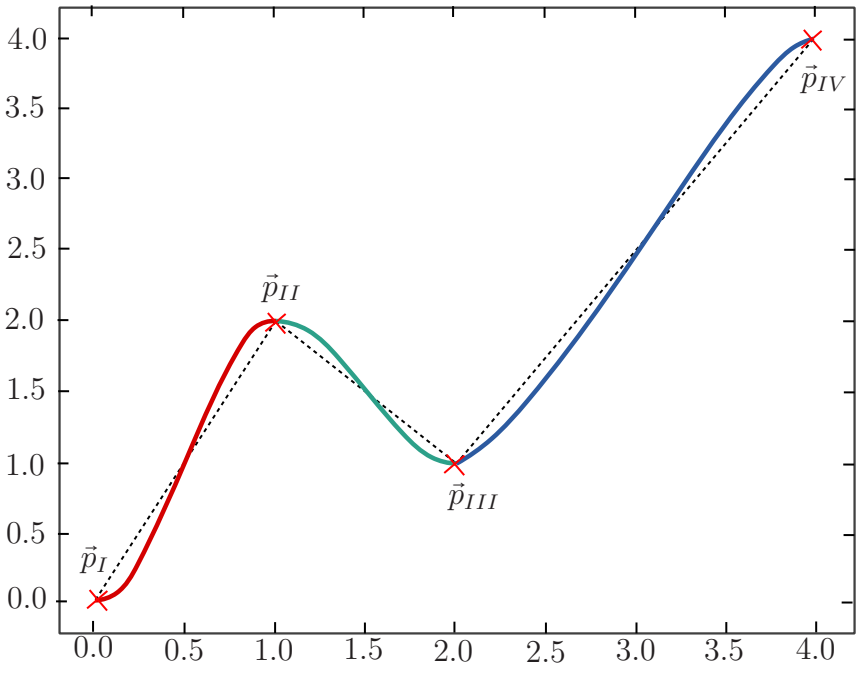


Figure 10.10: Piecewise Interpolation

10.3.6 Bernstein Polynomial

Figure 10.11 determines an appropriate path for Stäubli RX 170:

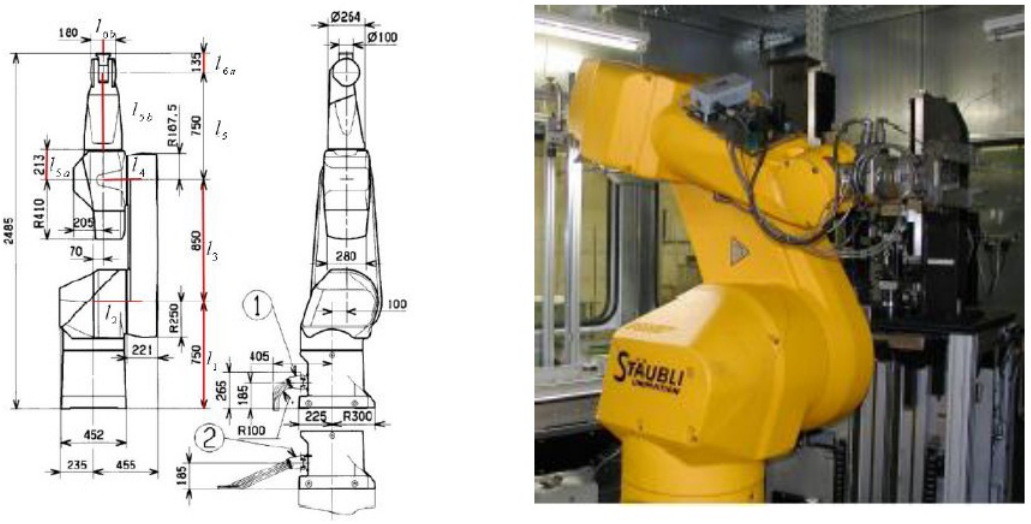


Figure 10.11: Stäubli RX 170

The function is:

$$X(t) = \sum_{i=0}^{n} b_i B_i^n(t) \tag{10.8}$$

The polynomial is of degree n and has the location vectors $\vec{p}_i$ of the nodes P_i. The following applies:

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i \tag{10.9}$$

The Bernstein polynomials have the property of adding up to 1 for any t between $[0,1]$:

$$\sum_{i=0}^{n} B_i^n(t) = 1 \tag{10.10}$$

are always symmetrical:

$$B_i^n(t) = B_{n-i}^n(1-t) \tag{10.11}$$

and positiv:

$$B_i^n(t) \geq 0. \tag{10.12}$$

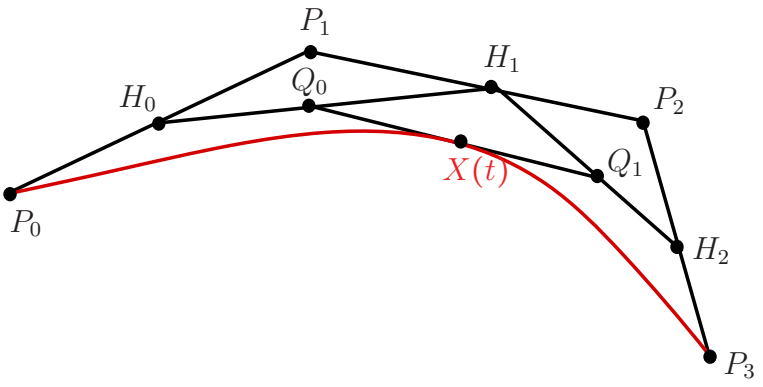


Figure 10.12: Bernstein Polynomials: Construction Principle

Intermediate Step

One can calculate the arbitrary intermediate steps. The Bernstein polynomials for the cubic case can also be represented graphically (see figure 10.12). For the cubic case of the Bernstein polynomial, the following also applies:

$$X(t) = \sum_{i=0}^{n} b_i B_i^3(t) \tag{10.13}$$

with the polynomial

$$B_i^n(t) = \binom{3}{i} t^i (1-t)^{3-i} \tag{10.14}$$

So the Bernstein polynomial for the cubic case is:

$$\vec{x}(t) = P_0(1-t)^3 + P_1 3(1-t)^2 t + P_2 3(1-t)t^2 + P_3 t^3 \tag{10.15}$$

Supporting points are approximated from below, but not all forms are possible.
See also figure 10.13.

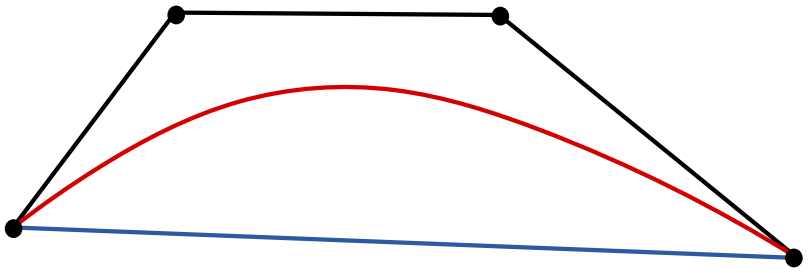


Figure 10.13: Intermediate Step

10.3.7 Supporting Points

In order to calculate of supporting points in the 2D-case from equation 10.15, it is advisable to multiply this out for the implementation:

$$x(t) = P_{0x}(-t^3 + 3t^2 - 3t + 1) + P_{1x}(3t^3 - 6t^2 + 3t) + P_{2x}(-3t^3 + 3t^2) + P_{3x}t^3$$

and,

$$y(t) = P_{0y}(-t^3 + 3t^2 - 3t + 1) + P_{1y}(3t^3 - 6t^2 + 3t) + P_{2y}(-3t^3 + 3t^2) + P_{3y}t^3$$

The values for P_0 and P_3 are already given with the start and end point of the spline segment. A problem that still needs to be solved is the calculation of the values for P_1 and P_2. In order to guarantee the property of continuity, the point P_1 is calculated by extending the distance $\overline{S_v P_0}$ by the factor τ:

$$P_{1,x} = P_{0,x} + \tau(P_{0,x} - S_v)$$

S_v represents the last supporting point of the previous segment. The same applies to the point P_3, only $\overline{E_n P_3}$ is now extended:

$$P_{2,x} = P_{3,x} + \tau(P_{3,x} - E_n)$$

E_n is the first node of the following spline segment. Figure 10.14 illustrates this. There are two last special cases to note: At the beginning of the spline, no previous support point S_v and no successor support point E_n is determined on the last spline segment. In order to be able to calculate P_{1y} and P_{2y}, the derivative in point P_0 in the first segment or P_3 in the last segment is set to zero. See Figure 10.15.

$$P_{1,y} = P_{0,y} + \tau(P_{3,y} - P_{0,y})$$

The same applies to the last spline segment:

$$P_{2,y} = P_{3,y} + \tau(P_{0,y} - P_{3,y})$$

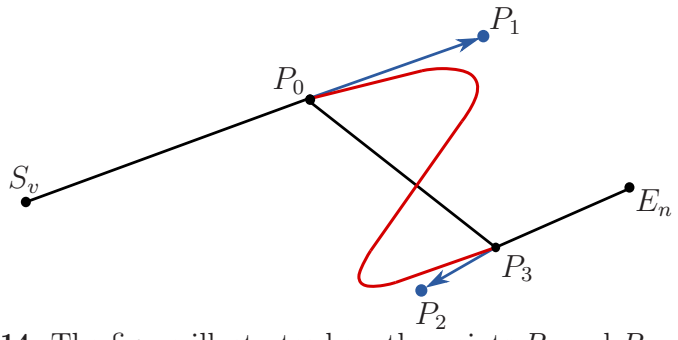


Figure 10.14: The figure illustrates how the points P_1 and P_2 are formed.

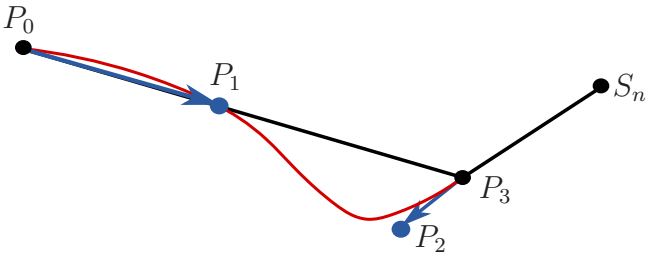
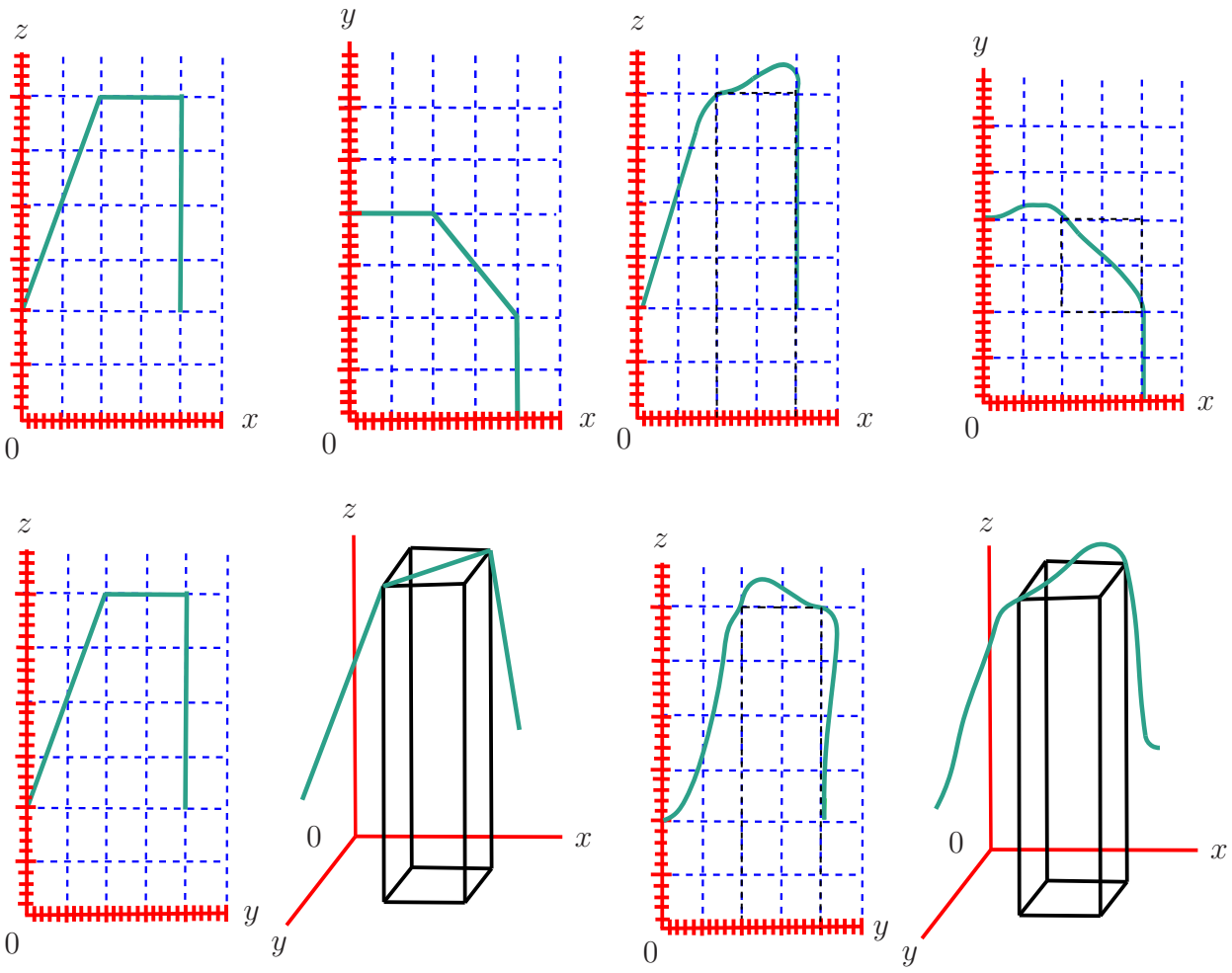


Figure 10.15: The figure illustrates how the point P_1 is formed in the case of the first spline segment.

Example: Spline with 3 Segments $\tau = 1$ and $\tau = 1.3$



10.3.8 Comparison CP & PTP

Comparison CP & PTP in configuration Space is shown in figure 10.16:

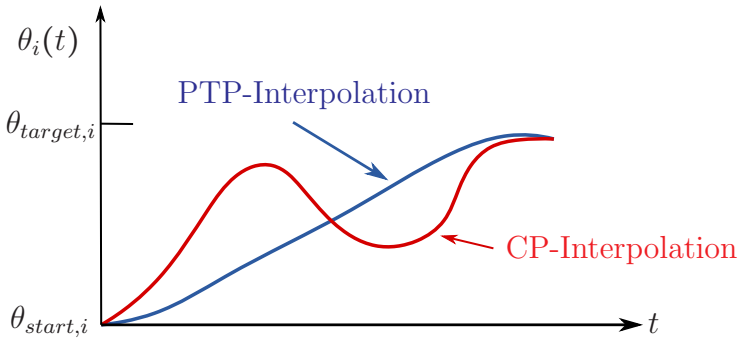


Figure 10.16: Comparison CP & PTP in Configuration Space

Comparison CP & PTP in cartesian space is shown in figure 10.17:

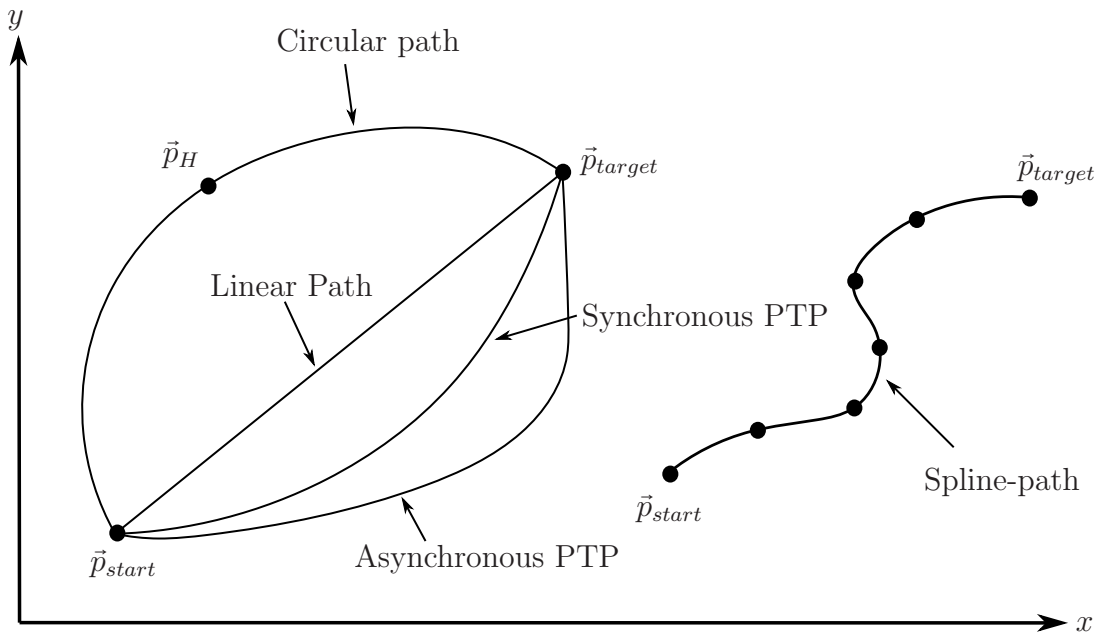
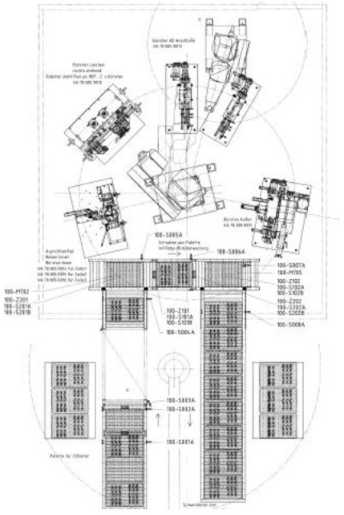


Figure 10.17: Comparison CP & PTP in cartesian Space

Example: Bosch Homburg



11. End Effectors and Grip Planning

Grippers/End Effectors manipulation is the result of the interaction between the end effector and the handled object. In industrial robots, the grippers are essentially used for transport tasks. They are used to change the position of an object by applying forces and moments. Gripping systems for industrial robots are mechanical grippers, grippers with suction units and magnetic grippers.

11.1 Foundations

Characterization of Technical End Effectors:

- Mechanics and principles of action
- Number of fingers
- Number of finger joints
- Type of force and form fit
- Movement possibilities
- Actuator types
- Gripping force
- Sensors
- Size and weight

Control Parameter of Gripping Systems:

- Position of the finger joints
- Gripping force

- Gripping path
- Gripping velocity
- Position of the object between the gripping plates
- Acting forces and moments

Gripping Force and Gripping Path Determination by:

- Work-piece weight
- Center of mass
- Geometry and position of the work-piece
- Grip points
- Positions of engagement

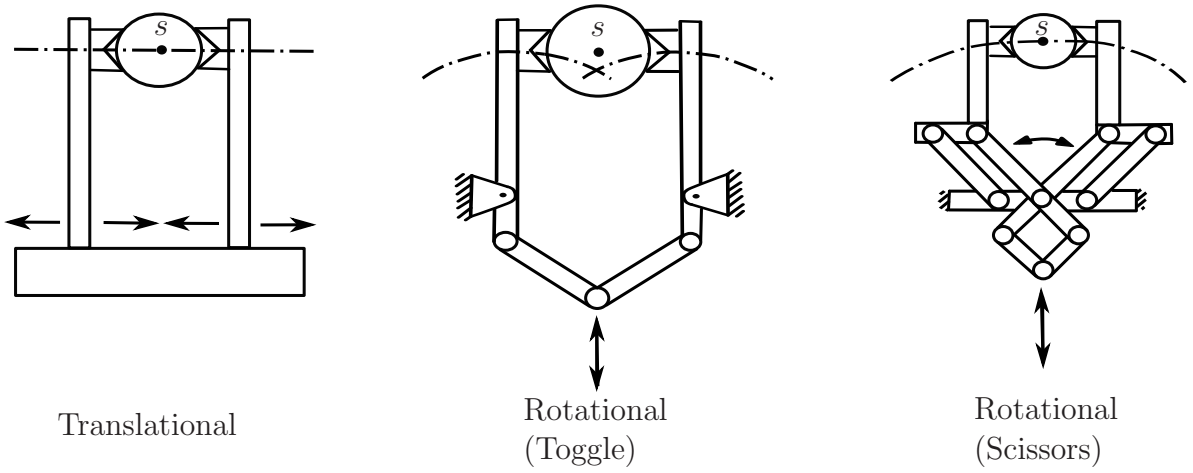


Figure 11.1: Mechanical Gripper

Enlargement of the Effective Surfaces

Figure 11.2 shows the two finger gripping system. In this system, the slipping hazard is high, and effective area is necessarilly enlarged.

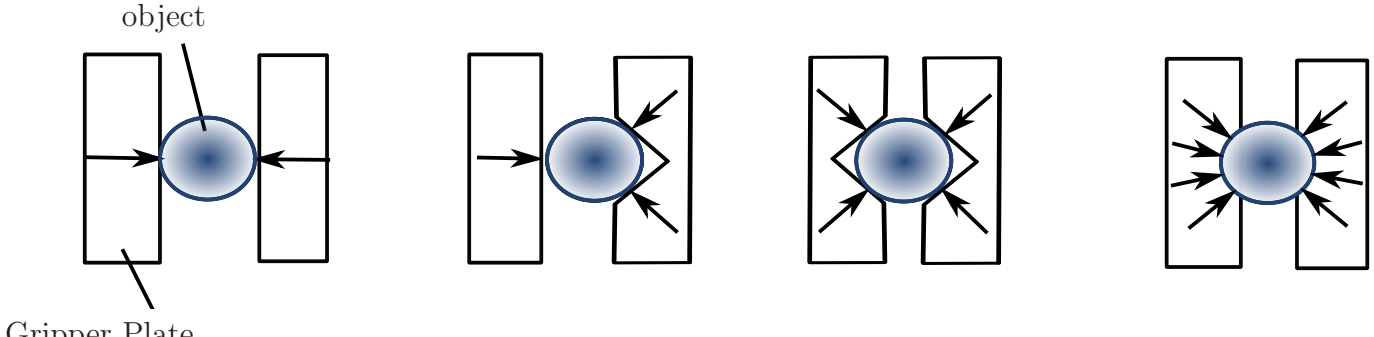


Figure 11.2: Enlargement of the Effective Surfaces: Two fingers system

Figure 11.3 shows the three fingers gripping system, with higher stability, and optimum positive locking.

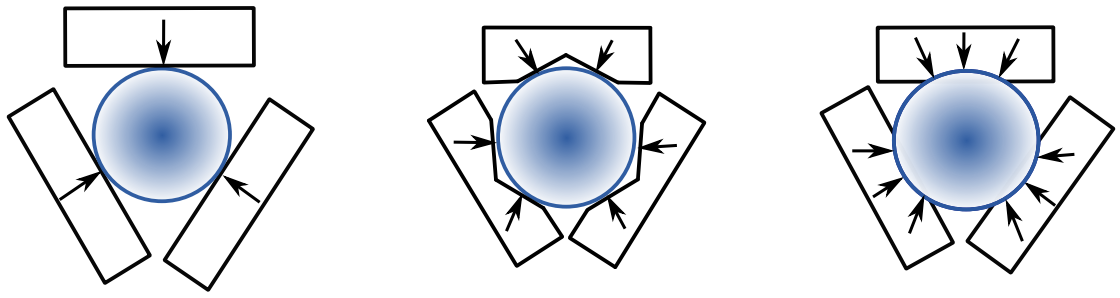


Figure 11.3: Enlargement of the Effective Surfaces: Three Fingers System

In the following, different types of grippers are shown:

1. **Scissor Gripper**

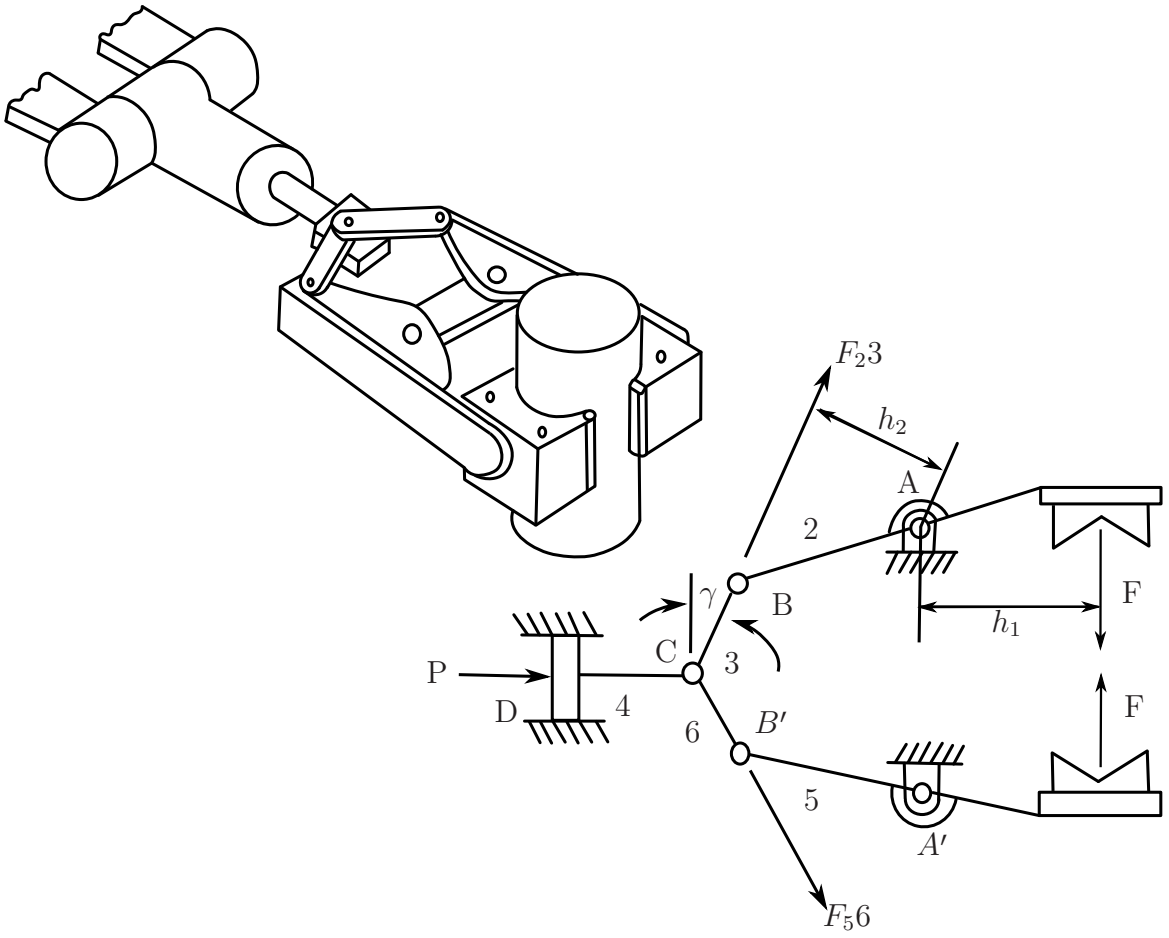


Figure 11.4: Scissor Gripper

2. **Pincer Gripper**

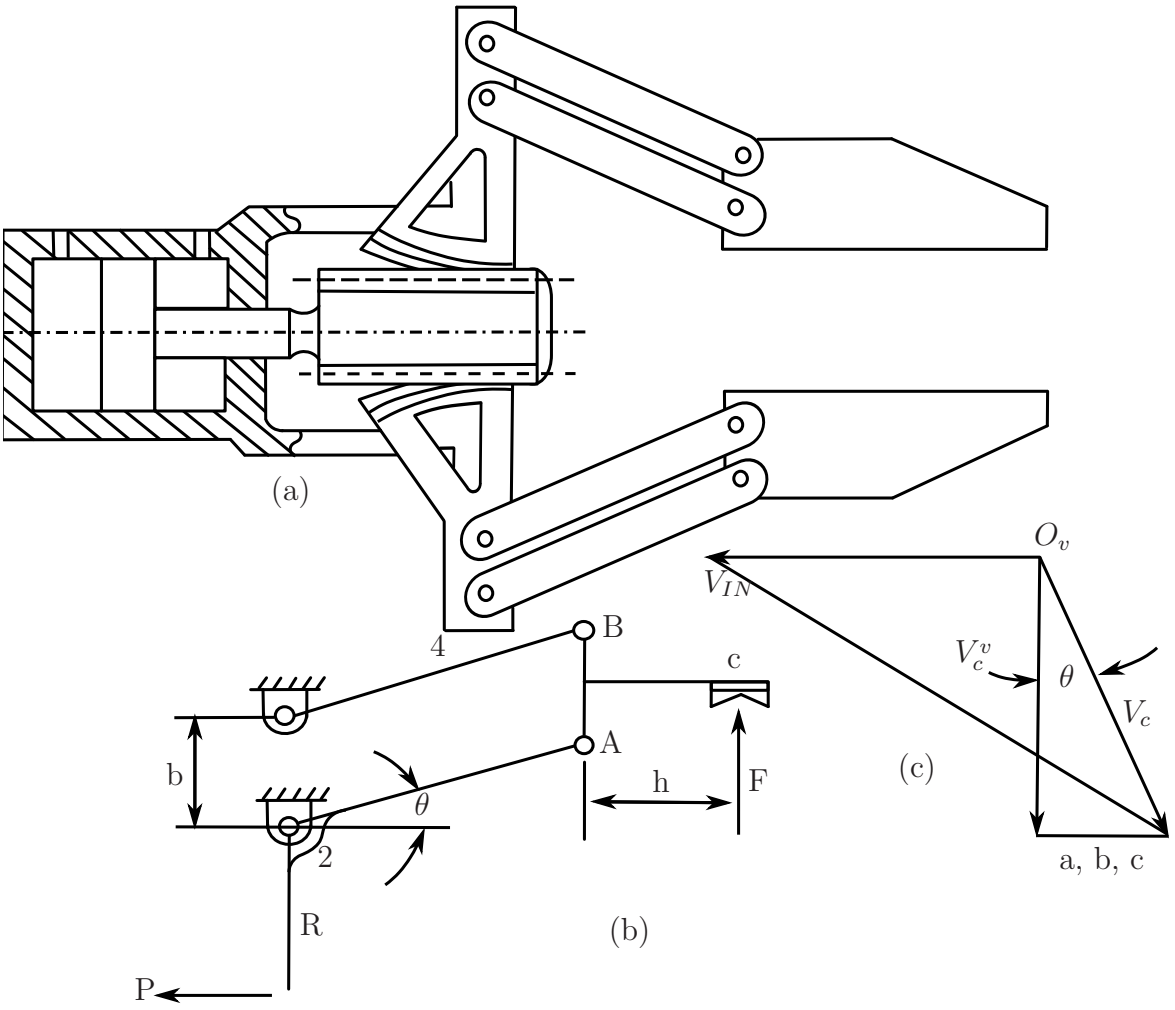


Figure 11.5: Pincer Gripper

3. **Suction Gripper**

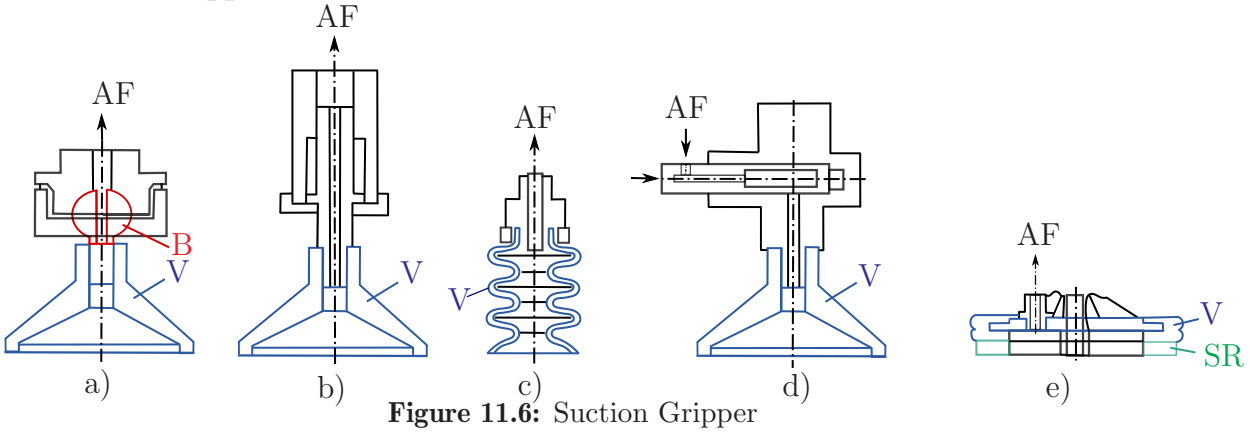


Figure 11.6: Suction Gripper
V=Vacuum
B=Ball joint
SR=Sponge Rubber
AF=airflow

In figure 11.6, part **a)** to **e)** show different type of suction gripper:

a) Suction cup with ball joint

b) Spring-loaded suction cup

c) Sucker for sensitive objects

d) Adhesive suction cup with valve for releasing air

e) Suction cup for concrete slabs

4. **Magnetic Gripper**

   Magnetic gripper has a simple construction. It has no wear on any parts, No moving elements and contact surface sufficiently. Materials are ferromagnetic, so for thin materials, several can be gripped at once. The grippers with electromagnets do not have any power in the event of a power failure. The gripping force results from:

$$F_G = \frac{B^2 A}{2\mu_O}$$

   with magnetic field B the area A, and vacuum permeability μ

11.1.1 Flexibility of Gripping Systems

The aim is the object-related adaptation in terms of force and form fit. Possible solutions for this are adjustable plate profiles and adjustable operating points of the gripper, interchangeable grippers, multi-jointed fingers, and sensor-guided flexible plate profiles.

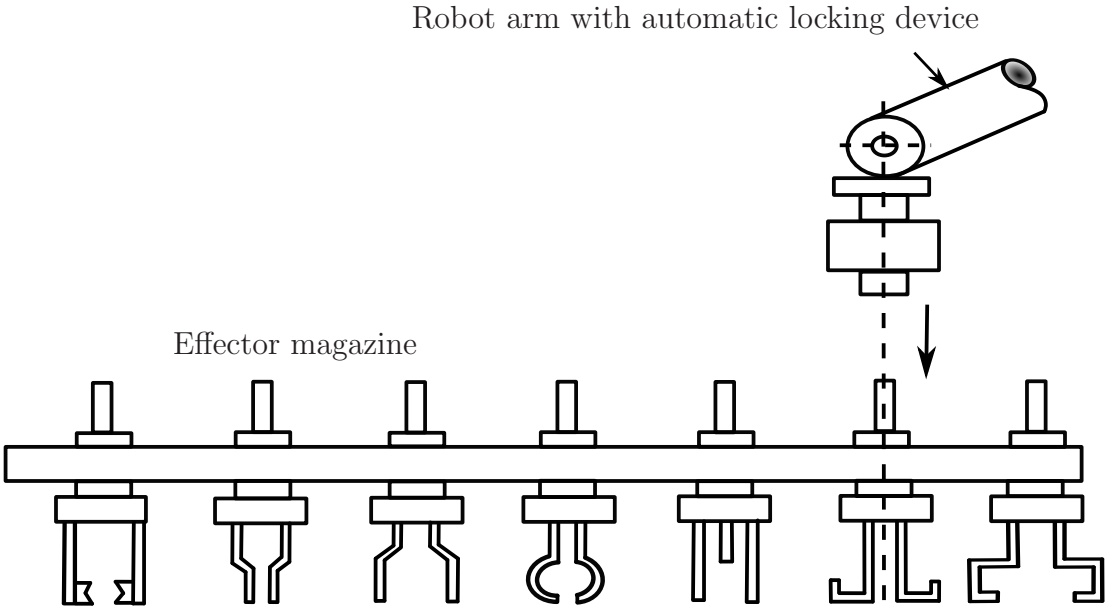


Figure 11.7: Flexibility of Gripping Systems

Improvement of the Form Fit

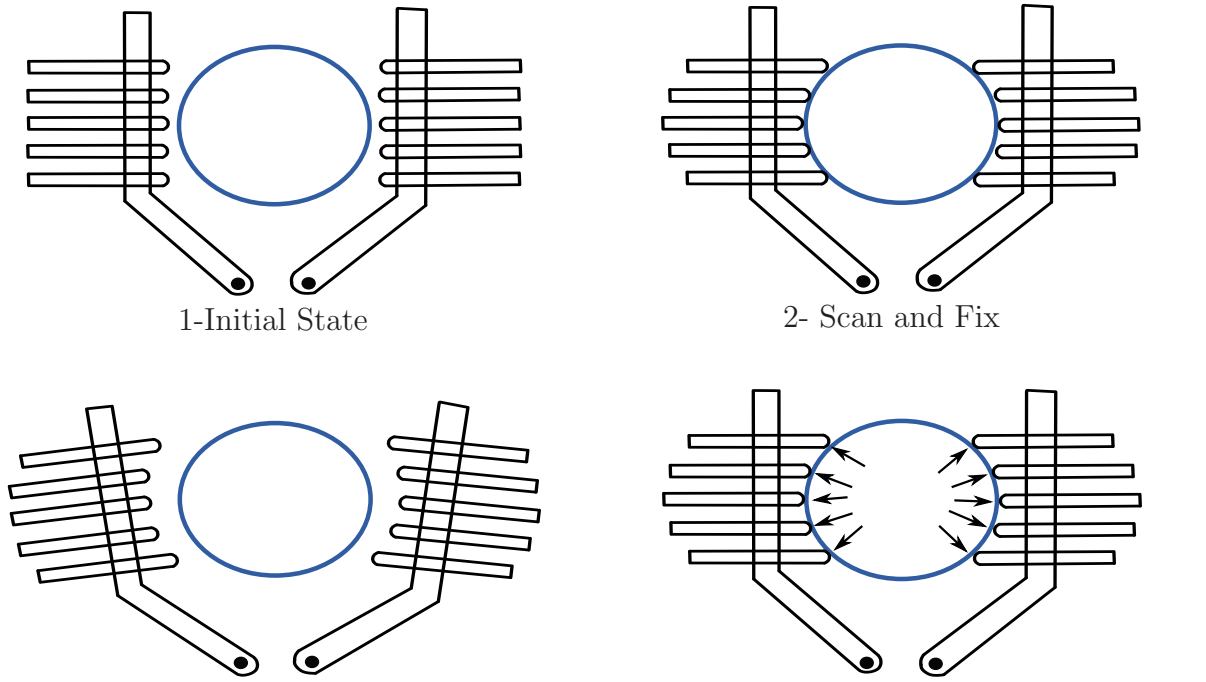


Figure 11.8: Improvement of the Form Fit

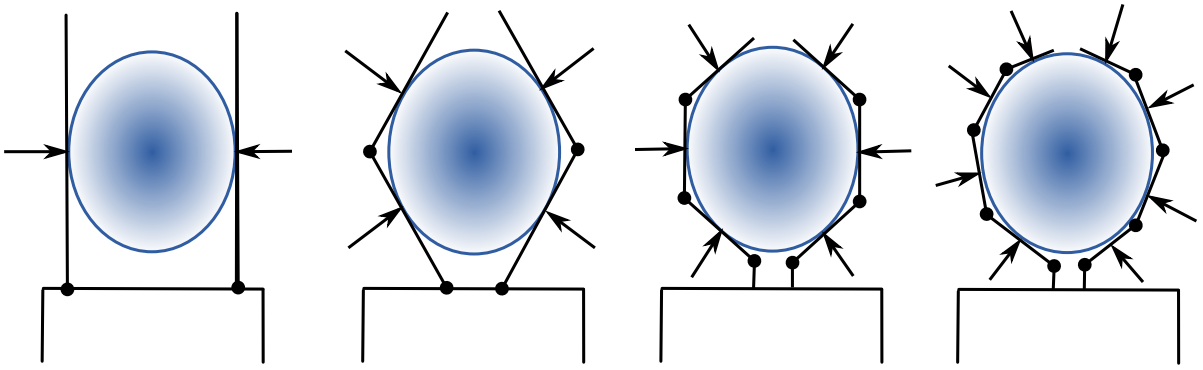


Figure 11.9: Improvement of the Form Fit

The Human Hand

The human hand is a universal gripper with 16 joints and 22 degrees of freedom.
The common modeling are kinematic model and area-based geometry model. See figure 11.10 and 11.11.

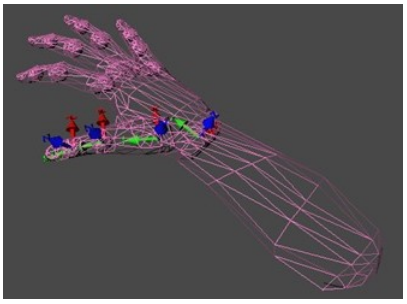


Figure 11.10: Kinematic Model Hand

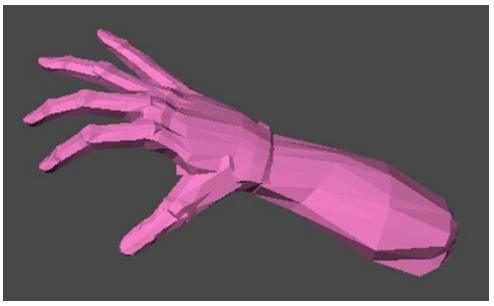


Figure 11.11: Area-based Geometry Model Hand

Grip Taxonomy According to Cutkosky

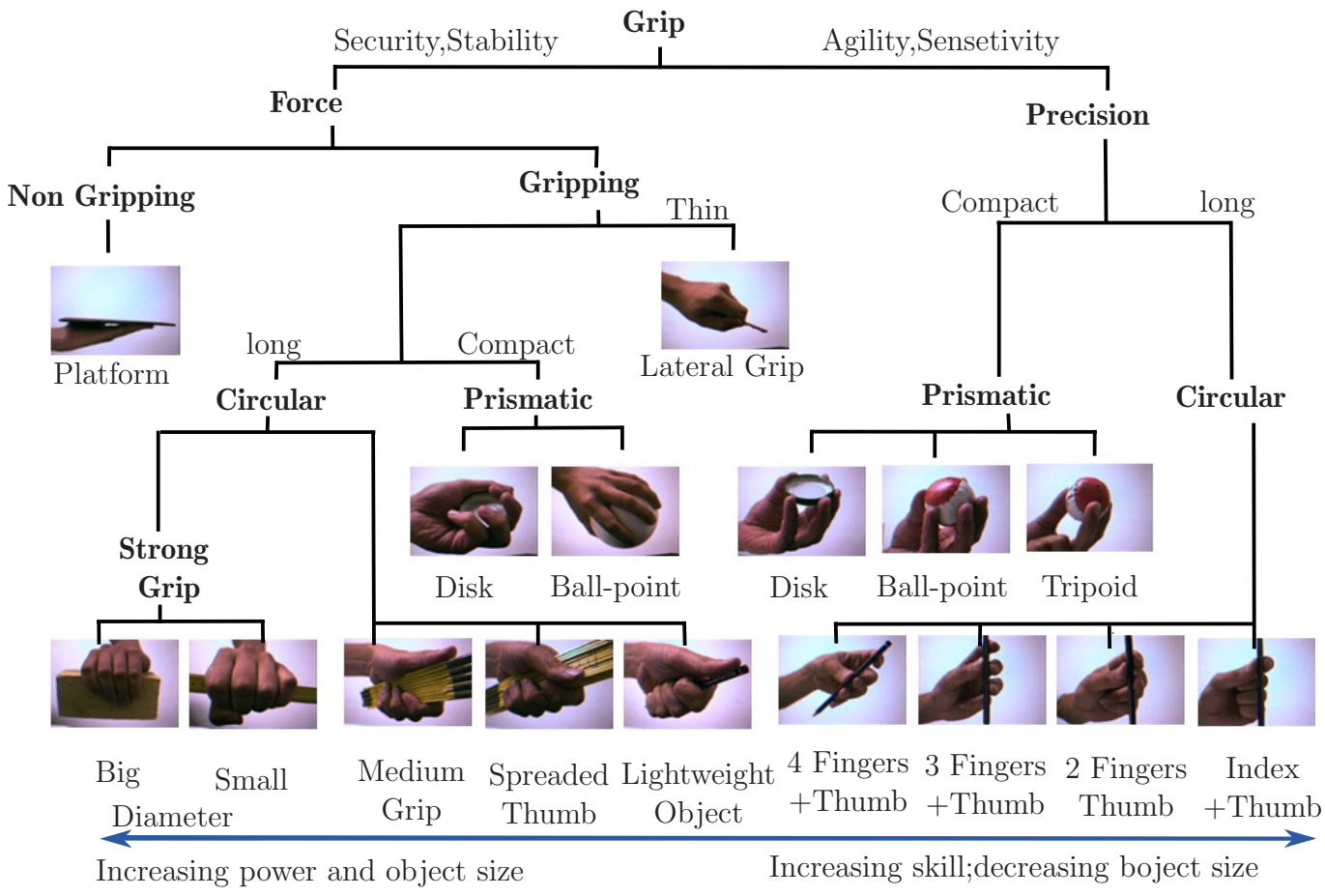


Figure 11.12: Grip Taxonomy According to Cutkosky

In figure 11.13, FZK–Hand (Forschungszentrum Karlsruhe) today KIT is shown:

Figure 11.13: FZK–Hand

11.2 Gripping Operations

Example Sequence

Instruction: Mount parts A and B according to installation plan P

Possible sequence (actions):

Action 1: Move robot hand in position of engagement of part A

Action 2: Move to the gripping position of part A

Action 3: Close claw fingers

Action 4: Drive with gripped part A into the repositioning position of part A

Action 5: Move hand with part A in position of engagement of part B

Action 6: Move hand with part A to mounting position of A and B

Action 7: Connect parts A and B according to the specification of P

Action 8: Open gripper fingers

Action 9: Move to the release position of part B

The following types of movement can be distinguished:

Grasp/release object with mounted gripper Selection of a secure grip, i.e. the determination of a suitable geometric relation of the gripper fingers to the object gripped. collision avoidance between the gripper and the object to be grabbed and objects in the environment (actions 3 and 8).

Up/down movement of the gripper Planning the movement (position and orientation) collision avoidance between gripper, object to be grabbed, robotic arm and objects of the environment (actions 2,9).

Up/down movement of the gripper with gripped object Motion planning of the gripper with gripped object collision avoidance between gripper, gripped object, robotic arm and objects of the environment (actions 4,6)

Connecting the gripped object with other objects Sensor-monitored and/or sensor-guided movements (action 7)

Transfer movement of the gripper with/without a gripped object Higher execution speeds and lower accuracy requirements compared to above movement types (actions 1,5)

Pick-and-Place See figures 11.14 to 11.16.

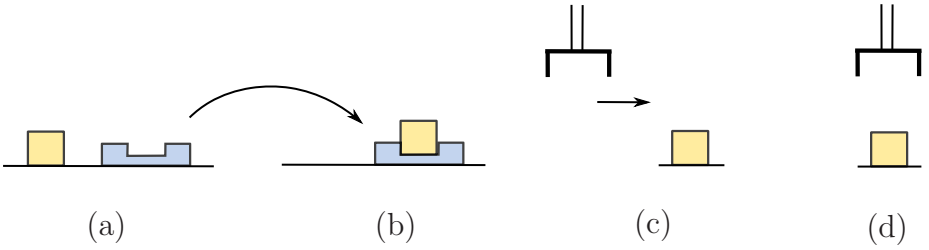


Figure 11.14: Gripping Operations: Pick-and-Place 1
(a) Picking configuration
(b) Placing configuration
(c) Transfer movement of the gripper
(d) Movement of the gripper to position of engagement

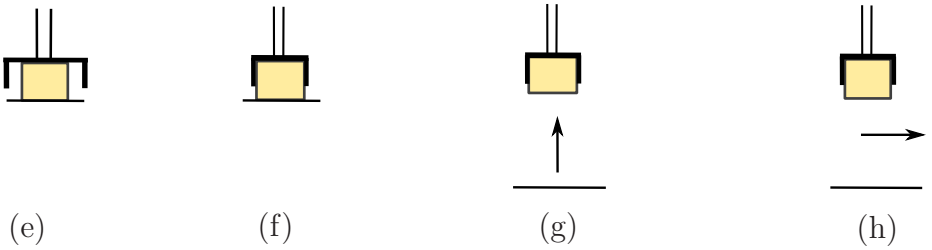


Figure 11.15: Gripping Operations: Pick-and-Place 2
(e) Reaching the picking configuration
(f)Gripping the object
(g) Upward movement of the gripper with the gripped object
(h) Transfer movement of the gripper with gripped object

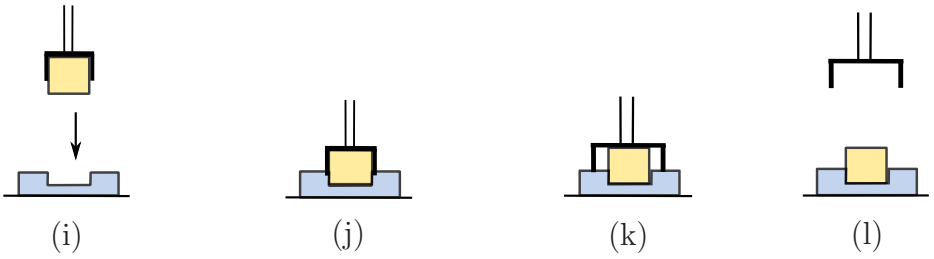


Figure 11.16: Gripping Operations: Pick-and-Place 3
(i) Downward movement of the gripper with the gripped object
(j) Reaching the place configuration
(k) Letting go of the object
(l) Upward movement of the gripper

Internal Constraints

Internal Constraints are:

I1 – Validity of a grip: Overlap between gripping features of the object to be gripped and the gripper fingers.

I2 – Collision free gripping: No collisions between gripper and gripped object

I3 – Accessibility of a grip: Handle is reachable without collision for grippers

External Constraints

External Constraints are:

E1 – Collision free movement of the gripper to position of engagement: No collisions between robot arm, gripper, adjacent objects and working plane.

E2 – Collision free movement of the gripper with the gripped object: See E1.

E3 – Consideration of the robot kinematics: Selected grip lies in the workspace of the robot, and corresponding trajectories of the up/down movement can be traversed by the robot.

E4 – Stability of a grip: Relative position and orientation of the object to be gripped or already gripped object to the gripper does not change (during gripping and transfer movement)

E5 – Stability of the scene: No influence on the scene stability during the removal of the gripper with gripped object.

E6 – Task dependency of a gripper: Selection of a suitable handle for pick-and-place operations with regards to pick and place configuration.

Planning Implications

The consequences for the corresponding planning are that no grip can be determined considering constraints for pickup and tray configuration. That means you have to determine a suitable recapturing sequence. Further the execute of a grip with special forces and torques on the gripped object is a consequence, which is why the gripping position, required forces and torques must be determined.

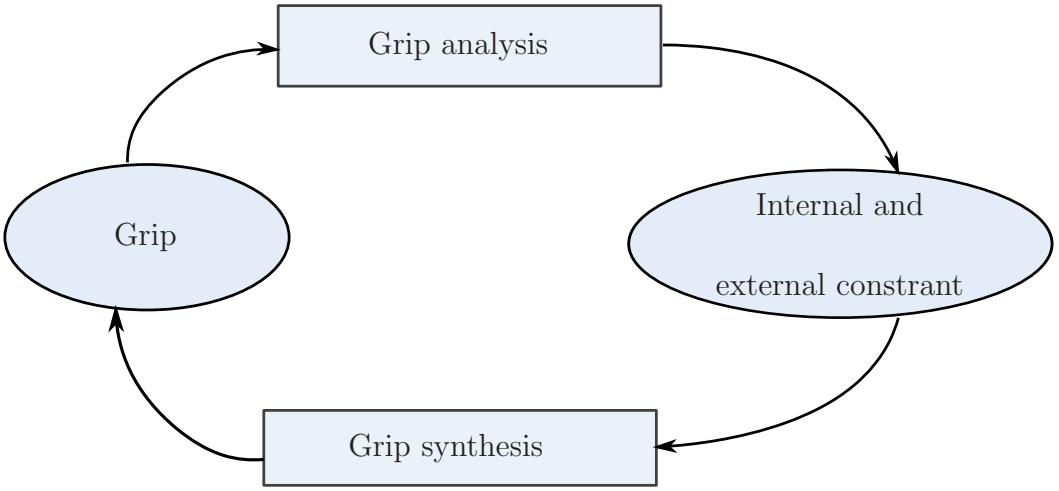


Figure 11.17: Planning steps for generating gripping operations

11.3 Fingertip Contact

Grip Model

Simplification of the synthesis of possible grips of an object by determining suitable contact points on the surface of the object to be gripped (Constraint I1). Disadvantage here is the failute to observe of fundamental constraints of the gripping process, such as the collision freedom and accessibility of a handle (Constraint I2 and I3). See figure 11.18.

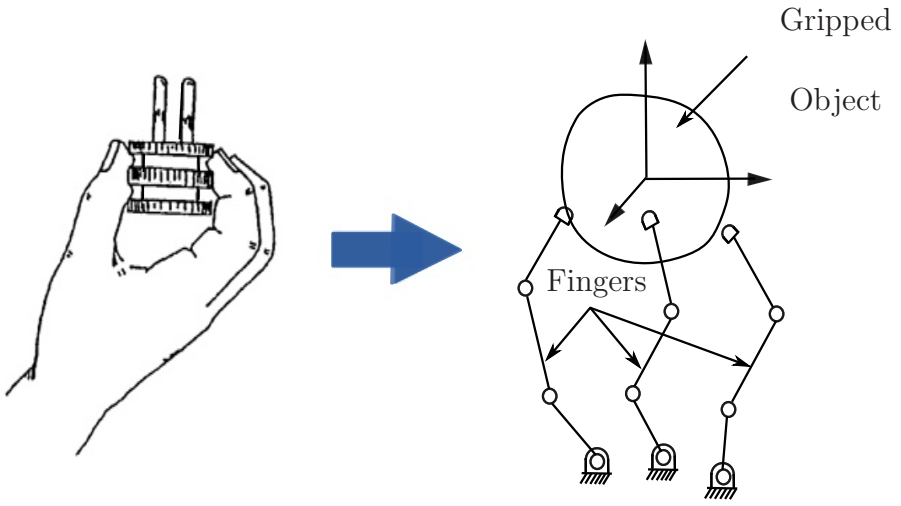


Figure 11.18: Fingertip Contact: Grip Model

A distinction between different fingertip contacts to the object surface is made with the following assumptions:

Point contact without friction: A force acting on a surface of an object at a point contact without friction, and effect exclusively normal to the surface. See figure 11.19.

Rigid point contact with friction: A force acting at a rigid point contact with friction on a surface of an object, effects both normal and tangential to the surface. The both forces are linked via Coulomb's friction law. See figure 11.20.

Non-rigid point contact with friction: A force acting on a non-rigid point contact with friction on a surface of an object, effects both normal and tangential to the surface. Both forces linked via Coulomb's friction law. See figure 11.21.

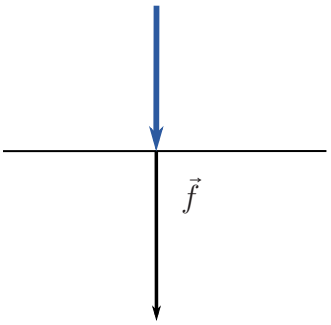


Figure 11.19: Point contact without friction

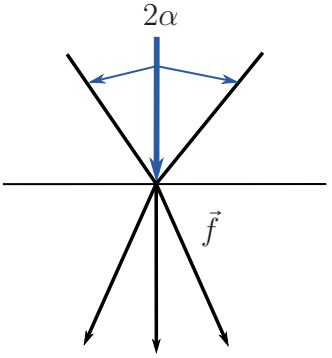


Figure 11.20: Rigid point contact with friction

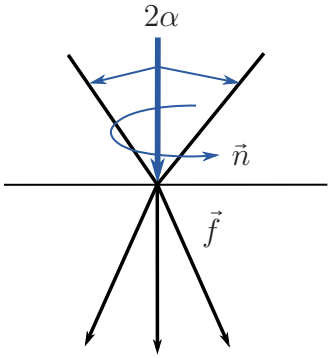


Figure 11.21: Non-rigid point contact with friction

11.4 Grip Hierarchy

Wrench Vector $\vec{w}$

The forces $f_i = \vec{f} = (f_x, f_y, f_z)^T$ and torque $\tau_i = \vec{\tau} = (\tau_x, \tau_y, \tau_z)^T$ with can be combined in a vector acting on the contact point $\vec{P}$. Such a vector is referred to in the following as *Wrench Vector* $\vec{w}$ (Wrench: turning, winding).
For this applies:

$$\begin{aligned} \text{Planar grip:} \quad \vec{w} &= (f_x, f_y, \tau_z)^T \in \mathbb{R}^3 \\ \text{Spatial grip:} \quad \vec{w} &= (f_x, f_y, f_z, \tau_x, \tau_y, \tau_z)^T \in \mathbb{R}^6 \end{aligned}$$

Depending on the type of i-th contact point, wrench vectors describe the normal n and tangential forces t and the axial torque θ acting on the contact point.

In the following identifier referred to as ${}^i\vec{w}_n, {}^i\vec{w}_t, {}^i\vec{w}_\theta$. The corresponding scalars are specified by ${}^i\vec{c}_n, {}^i\vec{c}_t, {}^i\vec{c}_\theta$.

Gripper Matrix

Represents geometric and physical properties of a fingertip grip:

Wrench vectors can be represented as a spatial vector as column vectors of a $6 \times 3m$ matrix G:

$$G = \left[{}^1\vec{w}_n, {}^1\vec{w}_t, {}^1\vec{w}_\theta, \ldots, {}^m\vec{w}_n, {}^m\vec{w}_t, {}^m\vec{w}_\theta \right]$$

The matrix G represents the geometric and physical properties of a fingertip grip and is referred to below as the *gripping matrix*. For the scalars we get the vector:

$$\vec{c} = \left[{}^1\vec{c}_n, {}^1\vec{c}_t, {}^1\vec{c}_\theta, \ldots, m\vec{c}_n, {}^m\vec{c}_t, {}^m\vec{c}_\theta \right]^T \in \mathbb{R}^{3m}$$

11.4.1 Equilibrium Grip

A grip specified by gripping matrix G, to which an external force and an external torque

$$\vec{e} = (f_x, f_y, f_z, \tau_x, \tau_y, \tau_z)^T \in \mathbb{R}^6$$

will be applied, if:

$$\begin{aligned} \forall i \in [1,m] \quad &: \quad {}^{i}c_n \geq 0, \quad {}^{i}\mu_t \cdot {}^{i}c_n \geq |{}^{i}c_t|, \quad {}^{i}\mu_\theta \cdot {}^{i}c_n \geq |{}^{i}c_\theta| \\ \exists \quad \vec{c} \in \mathbb{R}^{3m}, \quad \vec{c} \neq \vec{0} \quad &: \quad G \cdot \vec{c} + \vec{e} = 0 \end{aligned}$$

${}^{i}\mu_t, {}^{i}\mu_\theta \in \mathbb{R}$ are Coulomb friction coefficients at contact point i.

Limitation of the acting tangential forces t and axial moments θ with respect to the absolute value of the corresponding normal forces n.

A grip is considered an equilibrium grip if the sum of all forces f_i and torques τ_i acting on the gripped object is equal to zero. Equilibrium grip of an object is based on two rigid point-contacts without friction. An external force $\vec{f}$ acts on the object‘s center of gravity. See figure 11.22.

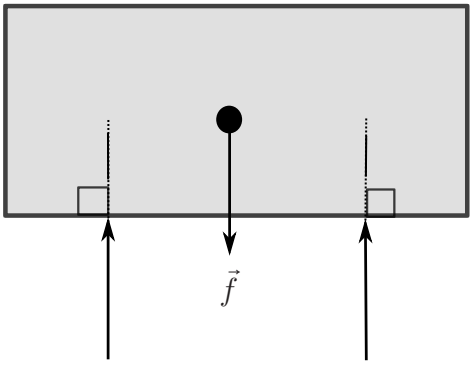


Figure 11.22: Equilibrium Grip

11.4.2 Force Closed Grips

Motivation:

During transfer movement and assembly operation various previously unknown external forces and moments act on an object. The solution is:
Stability of a grip through the balance of forces ans forces and moments that are exerted on objects by gripper fingers must compensate external forces and moments.

Grips specified by gripping matrix G on which any external forces and moments $\vec{e} = (f_x, f_y, f_z, \tau_x, \tau_y, \tau_z)^T \in \mathbb{R}^6$ will be applied, if

$$\forall \vec{e} = (f_x, f_y, f_z, \tau_x, \tau_y, \tau_z)^T \in \mathbb{R}^6$$
$$\exists \vec{c} \in \mathbb{R}^{3m}, \vec{c} \neq \vec{0} : G \cdot \vec{c} + \vec{e} = 0$$

Contact Points

The following applies to the minimum number of contact points required for a force-closed fingertip grip:

Force closure with point contacts and without friction: Provided that the object to gripped has no rotational symmetry, a planar, force-closed grip needs at least 4 contact points. Become any 3-dimensional objects are considered, a maximum of 12 contact points are required. If the class of objects to be gripped is restricted to polyhedra, a general upper limit of 7 contact points applies.

Force closure with point contacts and friction: Every planar object can be gripped in a force-closed manner using a fingertip grip based on 3 contact points (see figure 11.23). For the spatial case, a lower limit of 4 contact points applies.

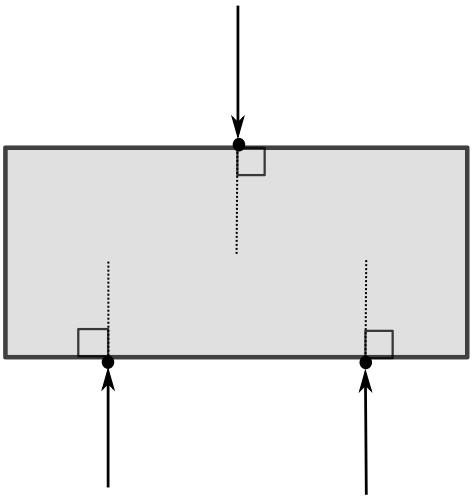


Figure 11.23: Force Closed Grips

11.4.3 Form Closed Grips

Planar form closed grip of an object is based on three non-rigid point contacts with friction. See figure 11.24.

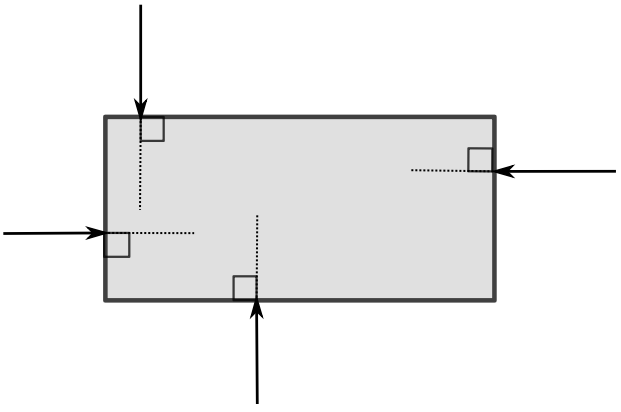


Figure 11.24: Form Closed Grip of an Object

For each contact point, consider only the non-penetrating properties co-linear to the corresponding external surface normal vector. Form-closed grids only dependent on the

position of the contact points and the corresponding external surface normal vectors. No consideration of normal or tangential forces and torques, which appear due to friction, among other things. External surfaces normal vectors corresponding to the contact points specify the contact geometry of the fingertip grip.

The grip matrix is:

$$G' = \left[{}^1\vec{w}_n, {}^2\vec{w}_n, \ldots, {}^m\vec{w}_n\right] \in \mathbb{R}^{6m}$$

The following applies to the minimum number of contact points required for a closed fingertip grip:

At least 4 contact points are required for a closed, planar grip. With any, 3-dim. objects, the number increases to at least 7 contact points.

Grip specified by modified gripping matrix G', on which any external forces and moments $\vec{e} = (f_x, f_y, f_z, \tau_x, \tau_y, \tau_z)^T \in \mathbb{R}^6$ can be applied, if:

$$\forall \vec{e} = (f_x, f_y, f_z, \tau_x, \tau_y, \tau_z)^T \in \mathbb{R}^6$$

$$\exists \vec{c} \in \mathbb{R}^6 : G' \cdot \vec{c} + \vec{e} = 0$$

11.5 Stable Grips

The previous requirement was rigid gripper fingers. However, it is desirable and necessary to imprive the modeling of finger forces that compensate for small changes in the nominal position of the gripped object through flexibility.

This can work through a description with the help of a potential function: $V : \mathbb{R}^6 \to \mathbb{R}$. The potential function V specifies the potential energy stored in the grip as a function of the position and orientation of the gripped object.

Definition:

If the potential energy stored in an equilibrium grip of an object is specified via a potential function V, and if

$$\vec{\delta q} = (\delta_x, \delta_y, \delta_z, \delta_{\phi y}, \delta_{\phi y}, \delta_{\phi z})^T \in \mathbb{R}^6 \neq 0$$

describes an infinitesimal change in position of the gripped object and the resulting change in the potential energy, then the grip is stable if

$$\forall \vec{\delta q} \in \mathbb{R}^6 : \delta V > 0$$

Thus, an equilibrium grip is unstable when a position change exists for which the resulting change in potential energy is less than zero.

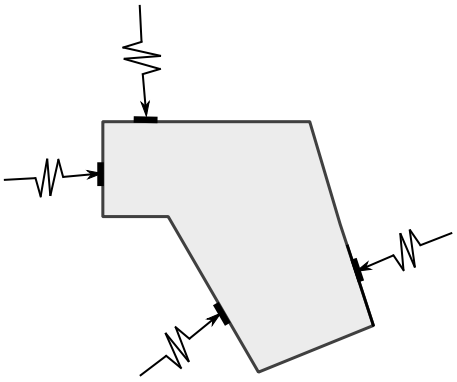


Figure 11.25: Stable and locked grip of a polygon based on 4 non-rigid point contacts with friction

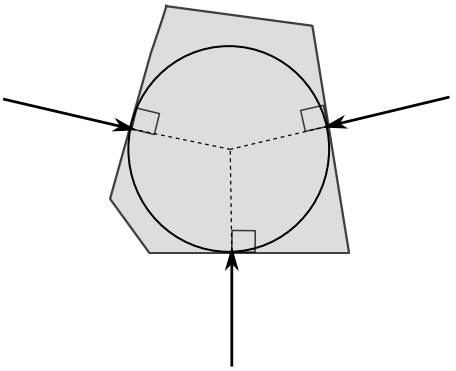


Figure 11.26: Stable triangle handle of a polygon; Not locked

12. Planning Systems

12.1 Forms of Planning in Robot Applications

Action planning: Which actions are derived from tasks?

Sequence planning: Temporal relationship between actions

Resource planning: Which robots/devices execute actions?

Time planning: When are actions started?

Execution planning: Which parameters for action execution (trajectory- and grasp planning)?

⇒ Scheduling: Resource and time planning

12.2 Planning in Robotics

Planning System

A planning system for robots is generally understood to be a system that, based on an initial state and the description of a desired goal state, generates action plan (a sequence of action) which gradually stepwise tranforms the system from initial state to the desired goal state. The plan is supervises with help of sensors and re-planned if deviation is effected.

Plan

The action sequences or graphs generated by a planner are called a *plan*. In the state space, the plan describes a set of state transition which lead to the desired goal state. These status transition can be sequential, concurrent or coordinated-concurrent. In general, plans can be divided into partial plans. On one hand in plan segments (complete sub-plans), and on the other hand in plan skeletons (incomplete sub-plans). An atomic plan segment or state transition is referred to as an action primitive or a plan primitive. A linear plan can therefore be compared with the linear decomposition of the manipulated variable for minimization of controlled system´s goal deviation.

Planning Methods

Planning is finding of a sequence of steps in order to arrive at a goal situation from an initial situation. The chain of state transitions can be calculated, derived, searched for, or chosen. Planning methods process exclusively symbol sets. Senseless plans are possible if knowledge base contains erroneous states or infeasible state transitions.

Planning problems are described with:

- Initial state
- Possible actions (operations), that can change the state
- Goal state (or set of goal states)

In the easiest case, plan is a simple sequence (or acyclic graph) of actions. In general case, any (known from programming languages) control structure is possible.

Edge- and Pre-conditions

Not every action is possible in given situation. Using the example grasping: Robot can grasp if:

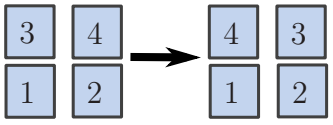
- Object is reachable
- Object‘s mass is movable by robot
- Grasper is in open position
- Grasper is at object‘s grasping position
- Grasper is appropriate for grasping object

Constraints in Assembly Planning

- Minimal processing time or in constant time span
- As few tool changes as possible
- Mechanical stability of partially assembled components
- Form of bonding (e.g. consider curing times when gluing)
- Material characteristics (e.g. deformability)
- Observability (e.g. camera)
- Precision of relative positioning of parts

12.3 Forms of Planning

Linear:	Strict temporal sequence of execution
Non-linear:	Complex temporal relations of actions (sequential, overlapping or parallel execution)



Placing on table not possible

Monotonic:	Every part is immediately placed in final position (no immediate placemer
Not monotonic	With intermediate placements
Central:	Complete knowledge about all sub-processes
Distributed:	Multi-agent-systems
Single-stage:	Plan generates actions directly
Multi-stage:	Plan generates sub-goals

Compatibility of Sub-Goals

Initial state: A and B on table, C on B
Goal state: A on C, and B is in grasper

Sub-goals:

- A on C, and B is in grasper
- First A on C → First sub-goal prevents second
- First B in grasper → Second sub-goal prevents first

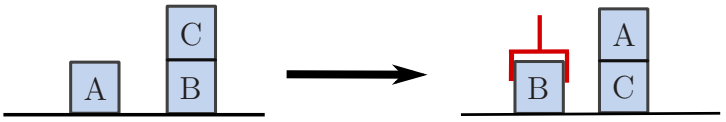


Figure 12.1: Compatibility of Sub-Goals

Fundamental Concept of Planning

Let:

- Z Set of possible states
- Initial state $M_a \in Z$, goal state $M_e \in Z$
- Set O of actions α with precondition V_α
- State transformation $g_\alpha : Z \to Z$ describes state transition if V_α valid

Searched is sequence M_i of states with $M_1 = M_a$, $M_n = M_e$

$$\forall i = 1, \ldots, n \quad \exists \alpha_i \in O \quad \text{with} \quad M_{i-1} \succ V_{\alpha_i} \quad \text{and} \quad g_{\alpha_i}(M_{i-1}) = M_i$$

Solution:
Plan generation through Sequence of logical operations.
Building of a tree with states as nodes and edges as actions.
Usage of search methods for plan generation such as depth-first search, breadth-first search, heuristic search method.

12.4 Planning as Logical Mapping

Solution of planning problems by deriving them via a logical calculus if it is defined, as follows:

"Can the target state be reached based on the current state?"

The calculus used is either the first-order predicate logic or the situation calculus. Deriving achievable states through planning through inference. Planning through monotonous logical derivation for complex problems or reactive planning systems can only be used to a limited extent. Many phenomena in the real world and thus also in the real environment in which agents act cannot be logically derived in a strictly monotonous manner.

For most practical purposes non-monotonous logic has to be extended with exception rules. This makes it unnecessary to implement a planner for an autonomous robot as a pure „logical proof system".

In order to be able to derive if a state is reachable from another, the states need to be formally describable:

Assuming a set of sensors or observers, which can indicate characteristics of individuals and their relations among each other. Then the set of characteristics and relations defines the current state of the process.

If a property x_j applies in a process state y, then one writes for it:

$$holds(x_j, y)$$

Terms can be constructed using logical connectives and quantifiers. Such terms are constructible, which make universally valid statements, new characteristics of individuals and their relations are derivable, even if they are not directly measurable using sensors. Example is, if x_j and x_k hold, then x_l also holds.

For process description (interaction, action, behavior) the situation calculus introduces a result operator ($result$). When in process state y an operation o or action u for state

transition is detected, then a new situation $result(u, y)$ arises. This allows for establishing the axiom, saying that for every state y and action u the state $result(u, y)$ is reachable. This state is also reachable from every state y_s, from which y is reachable.

$$\begin{array}{ll} \forall y: & \text{reachable}(y, y) \\ \forall y, u: & \text{reachable}(result(u, y), y) \\ \forall y, y_s, u: & \text{reachable}(result(u, y), y_s) \Leftrightarrow \text{reachable}(y, y_s) \end{array}$$

This axiom allows for deriving the reachability of a state from another. Depending on the chosen inference mechanism, the realization of planning through derivation can correspond to a search (Prolog depth search).
The proof problem can be described as follows:

Demonstrate that there exists at least one arbitrary state y_z that can be reached from another arbitrary state y_s, and y_s is defined by the characteristic x_s and y_z by x_z.

$$\forall y_s : \text{holds}(x_s, y_s) \Rightarrow \exists y_z : \text{holds}(x_z, y_z) \wedge \text{reachable}(y_z, y_s)$$

Example: Situation Calculus

The task of the proof system is to derives if a plan exists that transfers the initial state 12.2 to the goal state 12.3.
The goal state is defined by block A

$$holds(clear(A), y_z)$$

The initial state y is defined by

$$holds(on(C, B), y_s) \wedge holds(on(B, A), y_s) \wedge holds(clear(C), y_s)$$

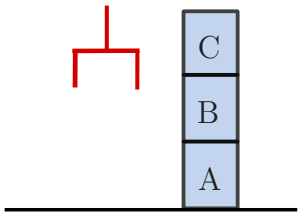


Figure 12.2: Initial state

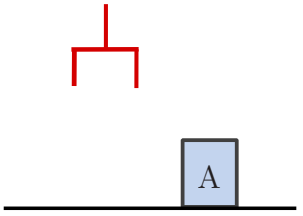


Figure 12.3: Goal State

The rule is defined for a stacking motion:

$\forall y : holds(clear(top), y) \wedge holds(on(top, bot1), y) \wedge holds(clear(bot2), y)$
$\Rightarrow holds(on(top, bot2), result(puton(top, bot2), y)) \wedge holds(clear(bot1), result(puton(top, bot2), y))$

The condition to be proven is:

$\forall y_s : holds(on(C,B), y_s) \wedge holds(on(B,A), y_s) \wedge holds(clear(C), y_s)$
$\Rightarrow \exists y_z : holds(clear(A), y_z) \wedge reachable(y_z, y_s)$

Applying the derivation rules one then obtains:

$\forall y_s : holds(on(C,B), y_s) \wedge holds(on(B,A), y_s) \wedge holds(clear(C), y_s)$
$\Rightarrow \exists y_z : holds(clear(A), y_z)$
$\wedge reachable(y_z, result(puton(B, table), result(puton(C, table), y_s)))$

12.5 Planning via Searching

Example

Possible plan:

- Block 4 on block 5
- Block 3 on block 4
- Block 2 on t_3
- Block 3 on block 2
- Block 4 on block 1
- Block 5 on block 3

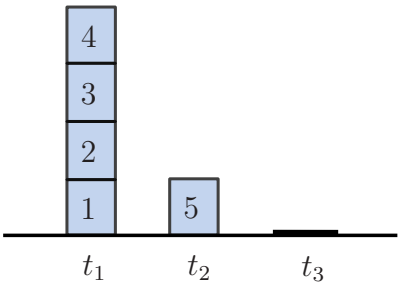


Figure 12.4: Planning via Searching

Initial State

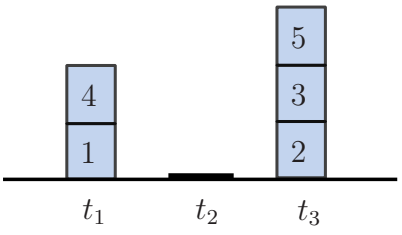


Figure 12.5: Planning via Searching

Goal State

12.5.1 Planning with Sub-Goals

Example: Sub-goal 1: $on(1, t_x)$ and $on(4, 1)$ (see figures 12.6 to 12.8), where the block A can also consist of several blocks.

- Block 4 on table

- Block A on table
- Block 4 on block 1

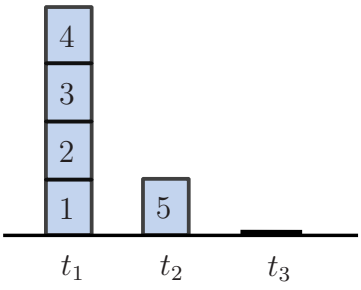


Figure 12.6: Initial State

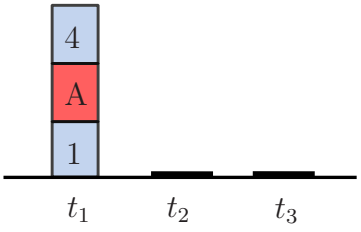


Figure 12.7: Initial State Sub-Problem 1

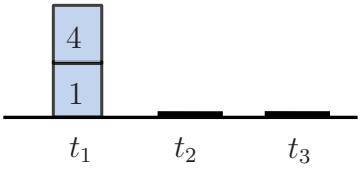


Figure 12.8: Goal State Sub-Problem 1

Sub-goal 2: $on(2, t_y)$, $on(3, 2)$ and $on(5, 3)$
There are two possible initial states. See figures 12.9 to 12.11):

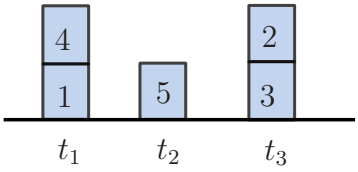


Figure 12.9: Initial State 1 Sub-Problem 2

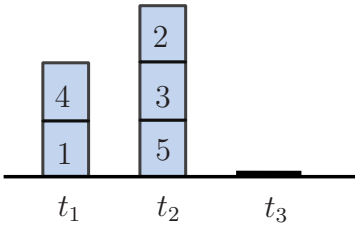


Figure 12.10: Initial State 2 Sub-Problem 2

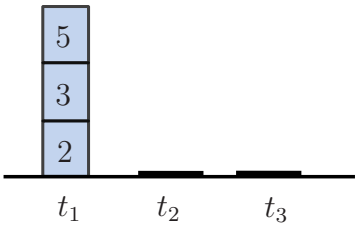


Figure 12.11: Goal State Sub-Problem 2

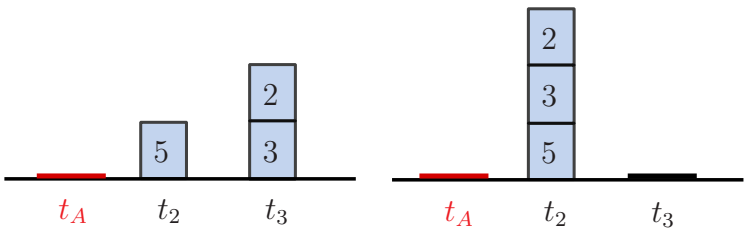


Figure 12.12: Abstract Initial States

Plan for sub-goal 2

- Block 2 on table

- Block 3 on block 2
- Block 5 on block 3

There are three possible solutions. See figures 12.13 to 12.15.

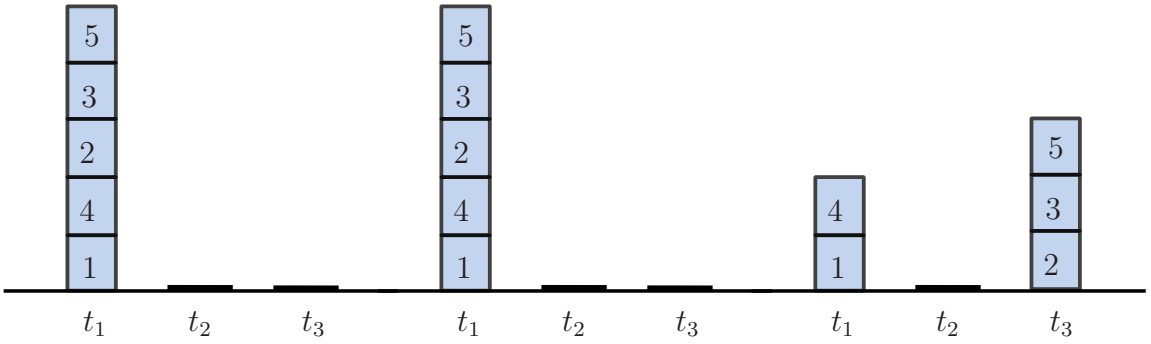


Figure 12.13: Solution 1 Sub-Problem 2

Figure 12.14: Solution 2 Sub-Problem 2

Figure 12.15: Solution 3 Sub-Problem 2

Example: Optimal Overall Solution

For optimal overall solution see also 12.16 and 12.17):

- Block 4 on t_3
- Block 3 on block 5
- Block 2 on block 3
- Block 4 on block 1
- Block 2 on t_3
- Block 3 on block 2
- Block 5 on block 3

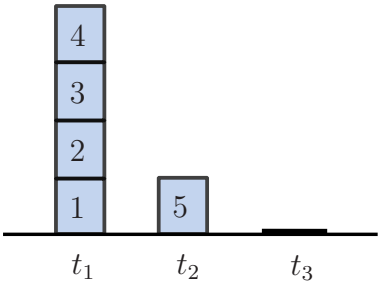


Figure 12.16: Initial Satet

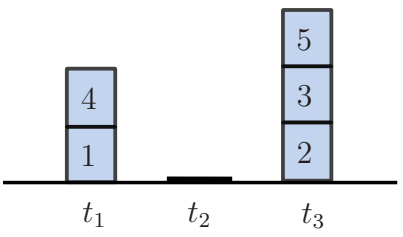


Figure 12.17: Goal State

12.5.2 STRIPS

STRIPS: Process state $x(y)$ as conjunction of a set of logical formulas x_i, which make statements about characteristics or relations of the process y:
$x(y) = \bigcup x_i$ with x_i holds in the state y.
Formulas x_i make statements about the actual process status. Based on information from sensor data (a), stored prior knowledge (b) or calculation (c). See images 12.18 to 12.20

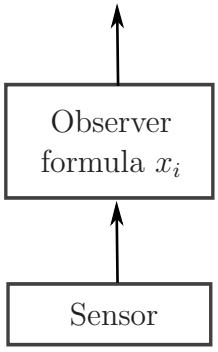


Figure 12.18: (a)

Sensor Data

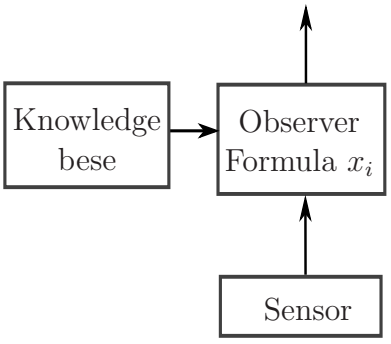


Figure 12.19: (b)

Stored Prior Knowledge

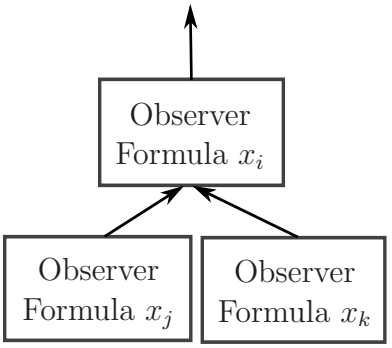


Figure 12.20: (c)

calculation

States are manipulated via operators o or plans consisting of operators p. An operator is a quadruple $o = (u, cond, add, sub)$.

Description u **:** Semantic meaning of modification

Pre-condition *cond* **:** Logic formula,examins if operator is applicable to state.

Add set *add***:** Additional formulae of the new state compared to the original state.

Sub set *sub***:** Missing formulae of the new state compared to the original state.

Representation of Block World

Current state:

$$x(y) = clear(A), clear(B), red(B)$$

Operator-quadruple

$$(puton(A, B), \{clear(A), clear(B)\}, \{clear(A), on(A, B)\}, \{clear(B)\})$$

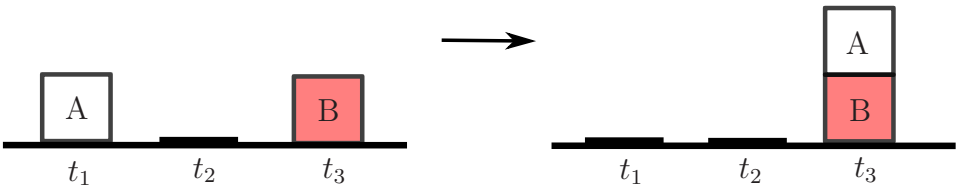


Figure 12.21: Representation of Block World

Application of operator:

- $puton(A, B)$ can be applied, because $\{clear(A), clear(B)\}$ is valid
- $clear(A)$ still holds
- Additinally $on(A, B)$ holds
- $clear(B)$ no longer valid
- No statement regarding $red(B)$ is made

The result of operator application is (see figure 12.21):

$$x(y) = \{clear(A), red(B), on(A, B)\}$$

Characteristics:

In the *add-set*, all formulae which hold in the new state need to be included. For example, if $clear(A)$ were missing, $puton(A, B)$ could lead to a state in which $clear(A)$ is not valid. In the *sub-set*, all formulae which do not hold in the new state anymore need to be included.

12.5.3 Examples for Planning Systems

- ASTRIPS (hierarchical)
- HACKER (single-stage)
- NOAH (hierarchical)
- ATLAS
- SHARP
- TWAIN (multi-stage)

Examples for planning phases for simple assembly are:

- World modelling
- Task specification
- Task analysis
- Building priority graph
- Planning of execution details
- Execution

12.6 Cranfield Assembly Benchmark

Example:

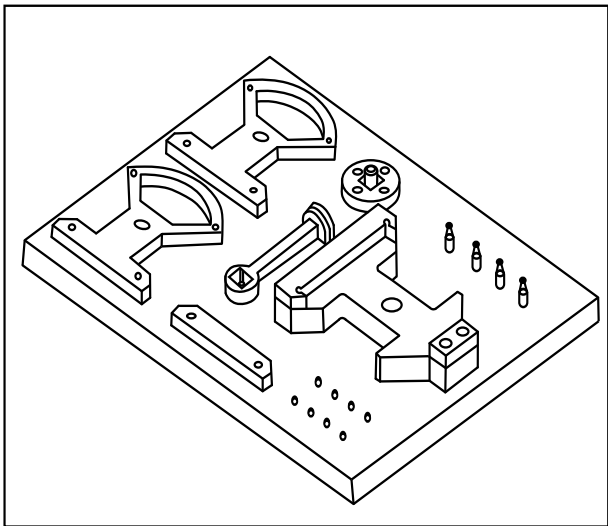


Figure 12.22: Initial Configuration

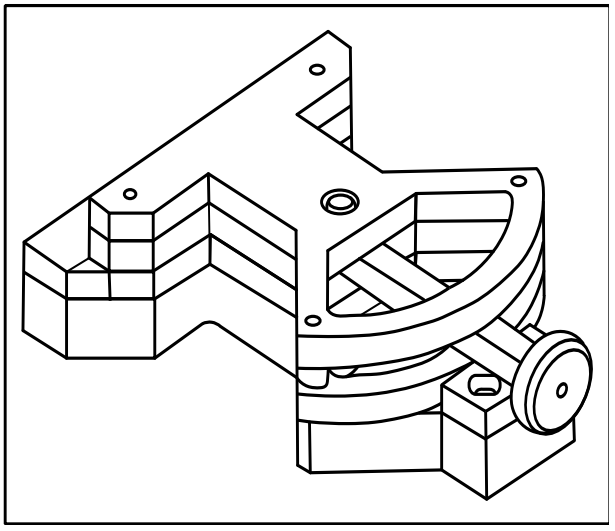
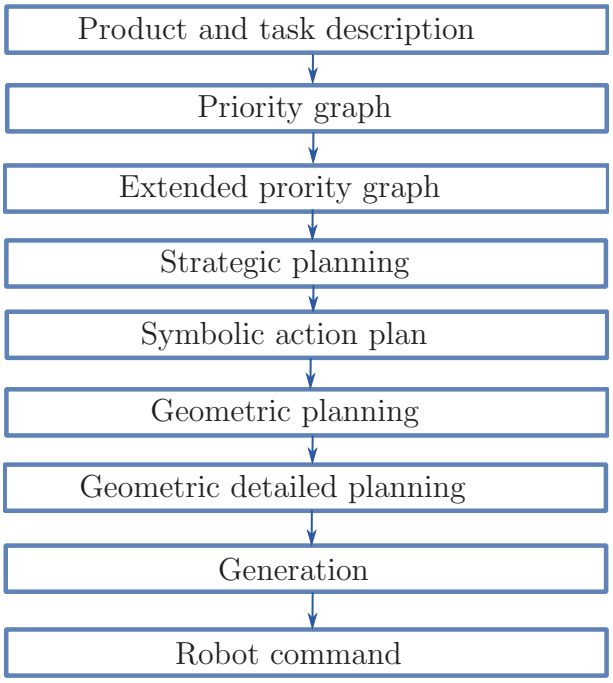


Figure 12.23: End Configuration

12.7 Planning and Supervision of Assembly



Task Specification

Modeling the initial and end configuration of the assembly.
The configuration consists of geometry (3D), physical properties (center-of-mass, material) and Cartesian coordinates of the item. The specification is a graphical editor for specification the position of item and specification of the grasper configurations.

Task Analysis

Criteria for Task Analysis are:

- No intersection: see figure 12.24
- No side-effects: see figure 12.25
- Stability: see figure 12.26

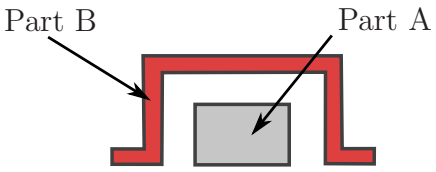


Figure 12.24: No Intersection

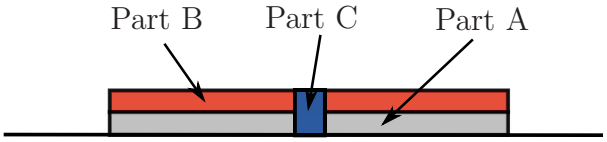


Figure 12.25: No side-effects

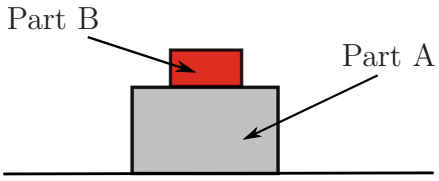


Figure 12.26: Stability

Example: Arrangement

See figure 12.27

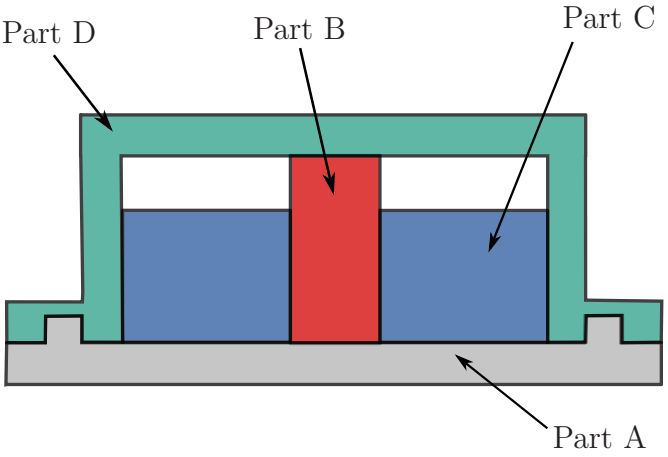


Figure 12.27: Arrangement

Example: Priority Conditions

Constrain source	Resulting priority condition
Intersection rule	$OR(AND(\pi(\vec{B},\vec{A}),\pi(\vec{C},\vec{A})),$ $AND(\pi(\vec{B},\vec{A}),\pi(\vec{C},\vec{C})),$ $AND(\pi(\vec{B},\vec{D}),\pi(\vec{C},\vec{D})),$ $AND(\pi(\vec{B},\vec{D}),\pi(\vec{C},\vec{A})))$
Stability rule	$AND(\pi(\vec{A},\vec{B}),\pi(\vec{A},\vec{C})\pi(\vec{A},\vec{D}))$
Side-effect rule	$OR(\pi(\vec{B},\vec{C}),\pi(\vec{D},\vec{B}))$

Example: Search Tree

See figure 12.28

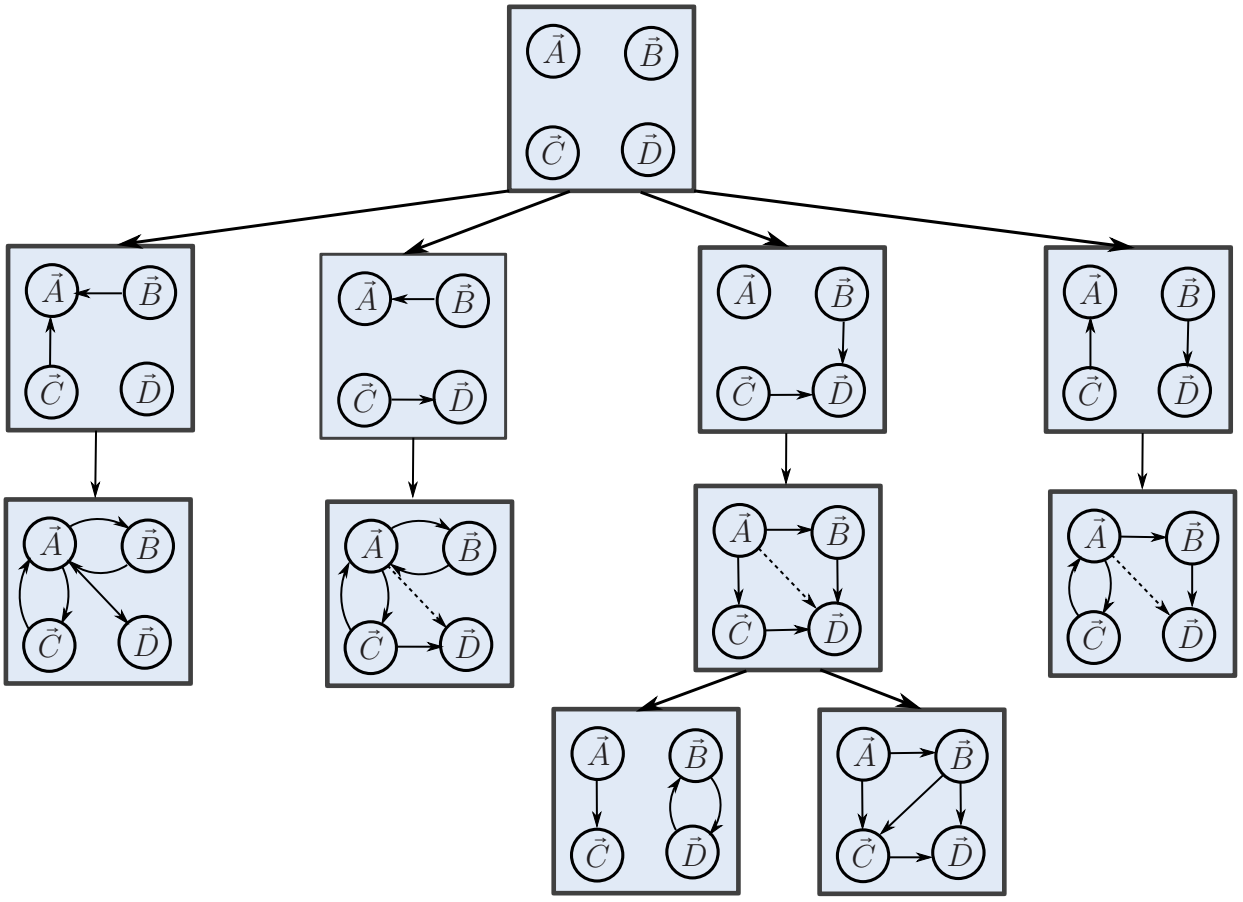


Figure 12.28: Search Tree

Priority Tree of Cranfield-Benchmarks

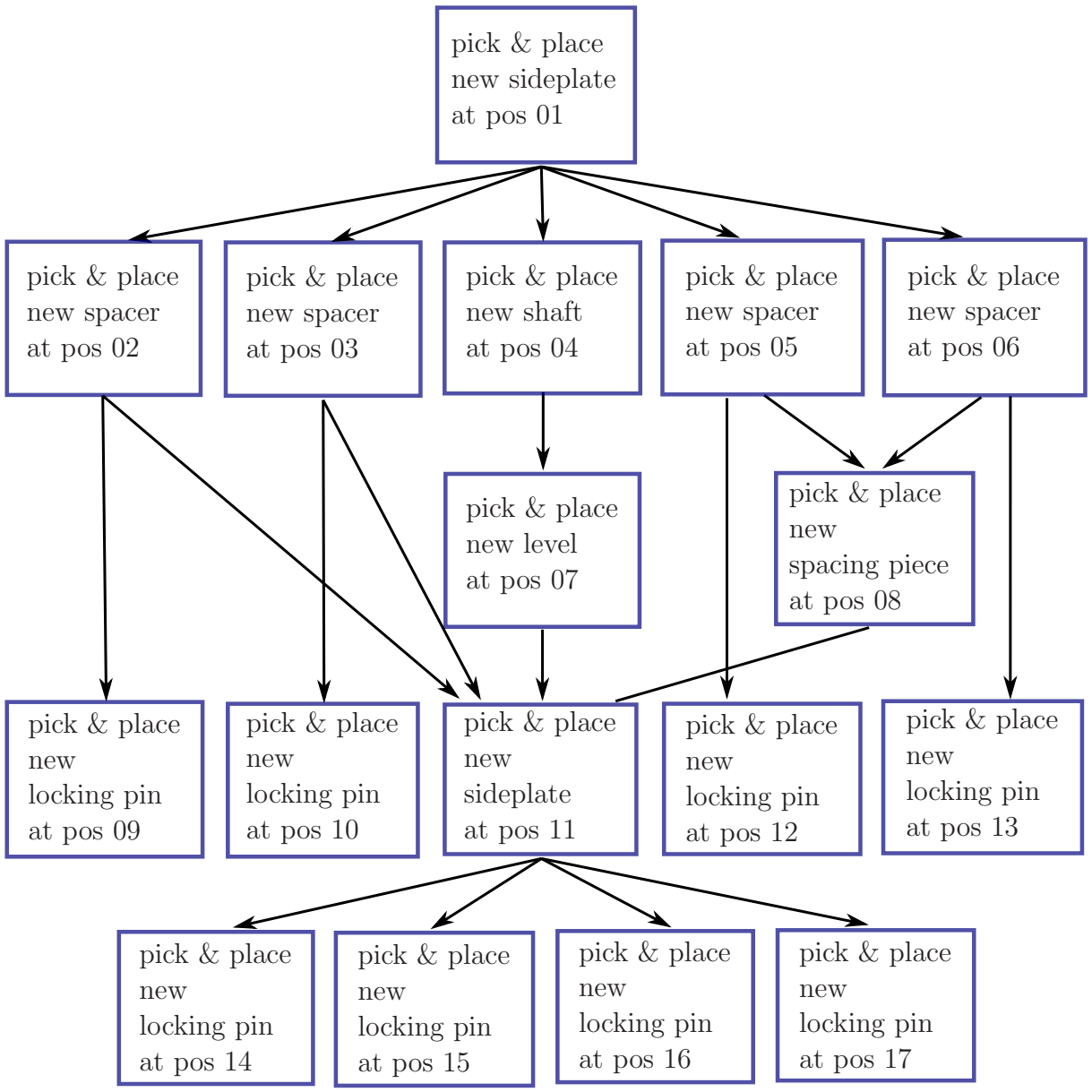


Figure 12.29: Priority Tree of Cranfield-Benchmarks

Detailed Planning

Planning is about robot selection and layout planning, i.e. finding a robot position. The criteria for this are reachability, accuracy and efficiency. Motion planning means a collision-free motion for grasping and releasing of items, collision-free minute motion to induce targeted contact of two items, and collision-free rough manipulator motion for transporting items.

Synthesis of Priority Graph

Fusion of rule set arising from task analysis criteria. The „best“ priority graph is selected, The redundant edges are eliminated and cycles are detected.

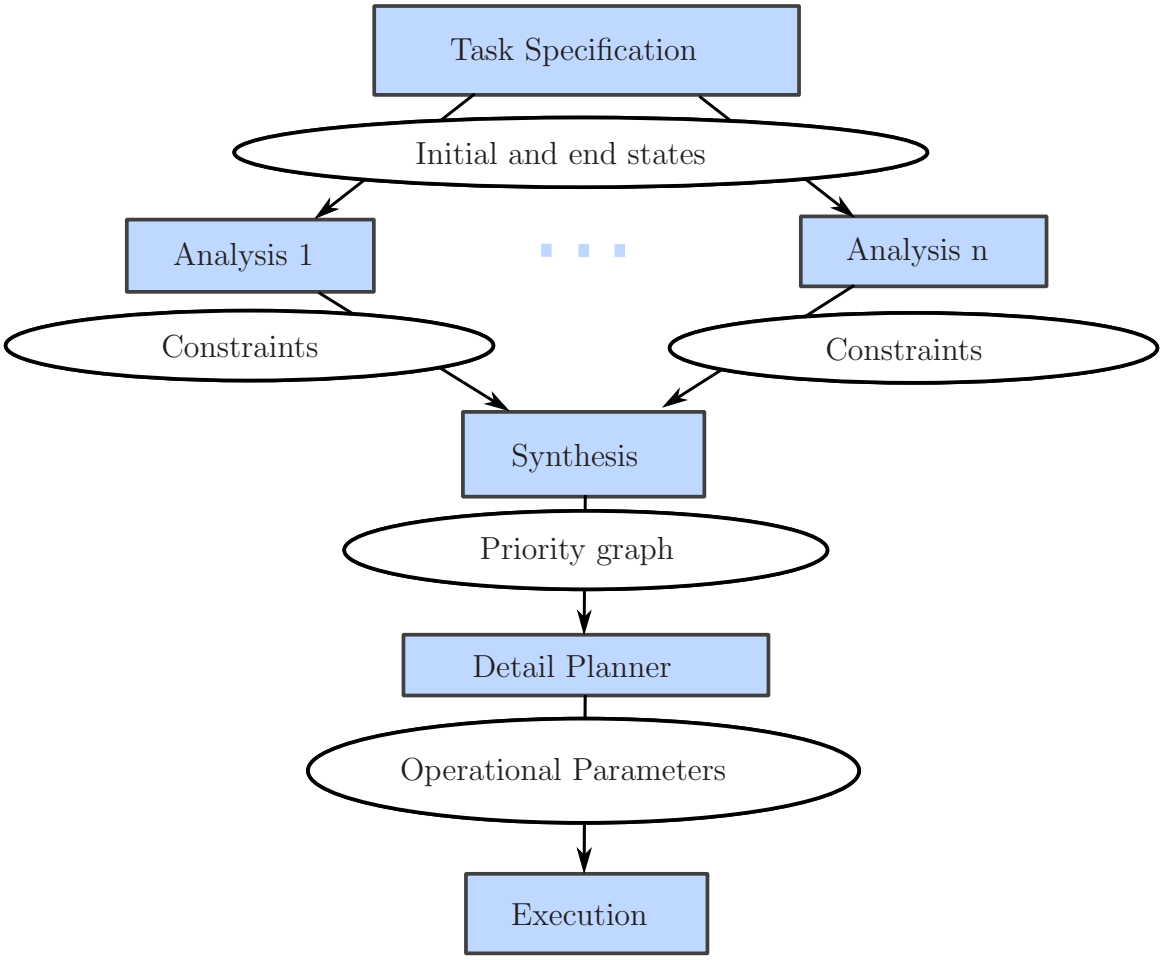


Figure 12.30: Overall System

13. Robot Architecture

This chapter contains general capabilities of a robotic system, and schematic visualization of the 4 main architectures:

- Hierarchical deliberative (function based architectures)
- Distributed deliberative (function based architectures)
- Hierarchical reactive (behavior based architectures)
- Distributed reactive (behavior based architectures)

13.1 General Capabilities of a Robotic System

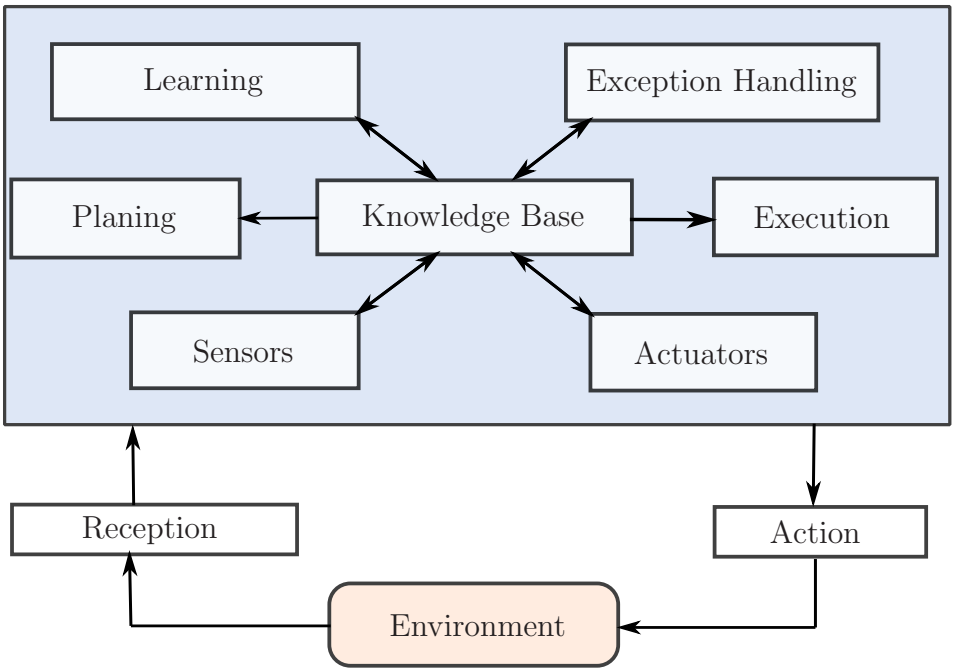


Figure 13.1: General Capabilities of a Robotic System

13.2 Main Architectures

Classification

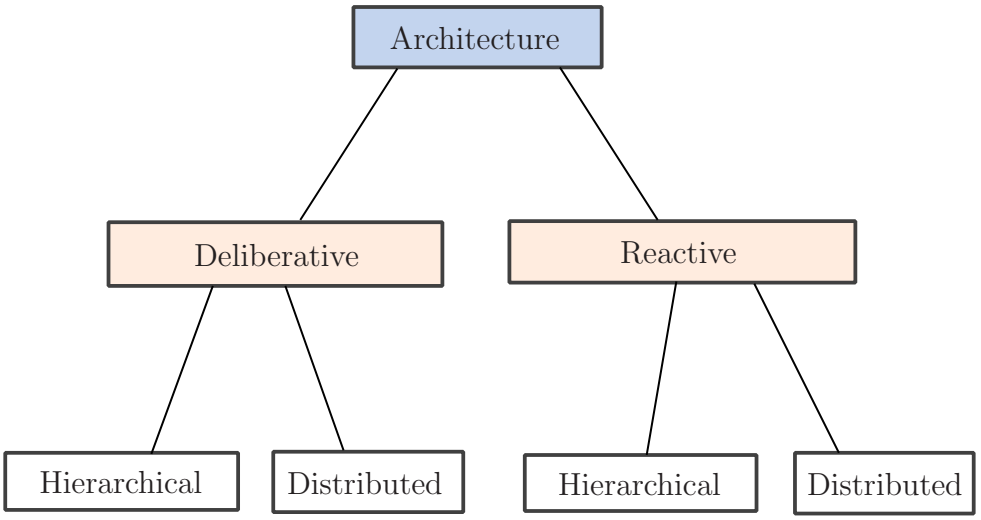


Figure 13.2: Main Architectures: Classification

Schematic Visualization

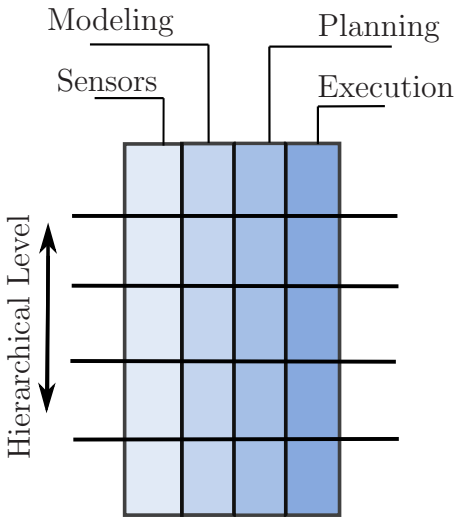


Figure 13.3: Hierarchical deliberative

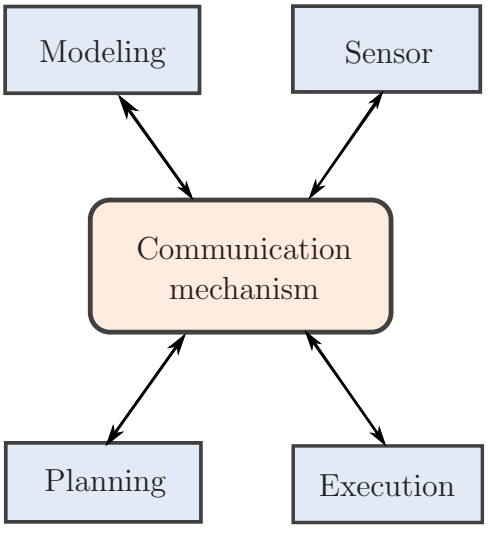


Figure 13.4: Distributed deliberative

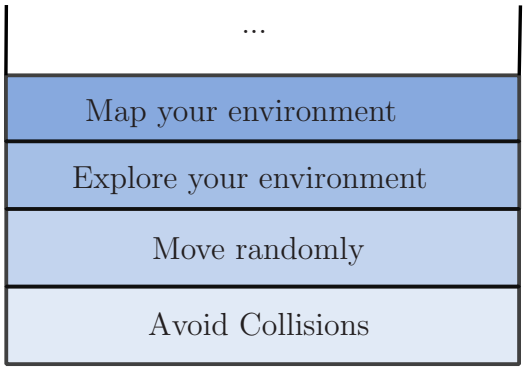


Figure 13.5: Hierarchical reactive

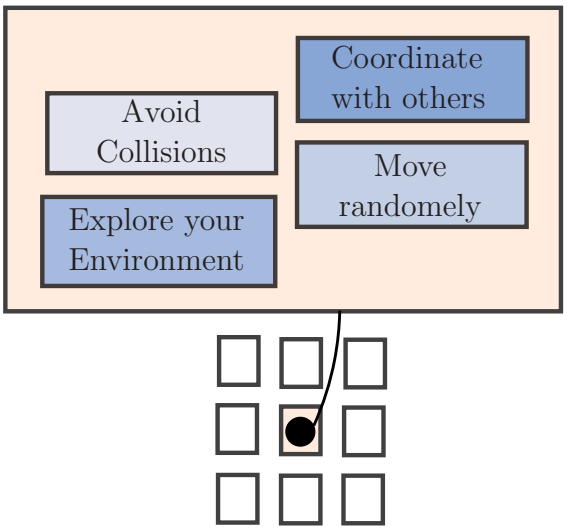


Figure 13.6: Distributed reactive

13.3 Hierarchical Deliberative Architectures

- RCS: Real-time Control System
- Reference-Arch. for development of intelligent control system
- Combination of planning and reactive control structures
- Hierarchical ordered set of nodes

Example: NASREM-Model
See [Albus 89].

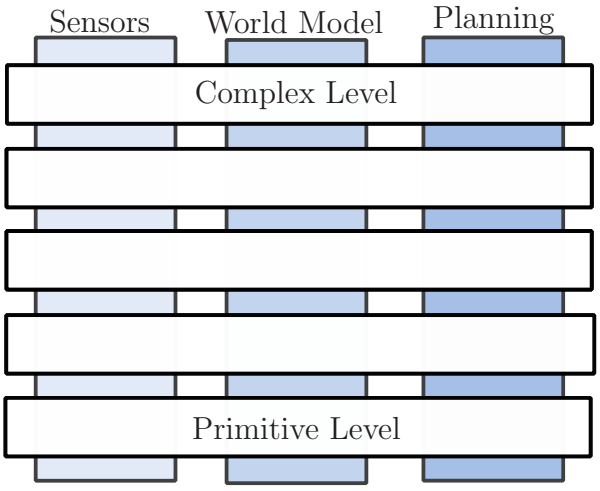


Figure 13.7: NASREM-Model

Divide in 4 to 6 levels. There are 3 Modules per level:

- Sensor processing module G_i
- World model and reference data module M_i
- Task division, planning and execution module H_i

Modules are ordered hierarchical by levels.

13.4 Modules

- **Planning module:**
  Plans and supervises the execution of actions.
  Considers data from world model and reference data module.
- **Sensor processing module:**
  Processes and filters sensor data.
  Compares observations with internal world model.
  Determines deviations.
  Detects events, objects and situations.
- **World model module**
  Refreshes reference data with sensor data.
  Predicts sensor results.
  Simulation of results in hypothetical plans.
  Information on the world (variables, objects, rules, maps)
- **User interface:**
  Interaction with processes or knowledge base

13.4.1 Communication between Modules

Communication of commands from H-modules to H-modules at the next lower level. Status/sensor information from G-module to G-module of the next higher level. Furthermore, the exchange of information within a level between G-module and M-module as well as between M module and H module. There are no special communication mechanisms. Examples are point-to-point connections, network connections, and shared memory.

Nodes in RCS

Implementation of the function within the nodes, e.g. as an extended finite automaton. Every automaton has defined initial states. Inputs from one or more input buffers and the status changes are based on information that has been read. The state change initiates actions. The outputs are stored in output buffers and the communication system distributes information from output buffers in corresponding input buffer. Synchronous or asynchronous control of the finite automaton. The execution frequency depends on the level.

Hierarchical Structured Sub-Functions

Figures 13.8 to 13.10 show the breakdown of control functions into hierarchically structured sub-functions:

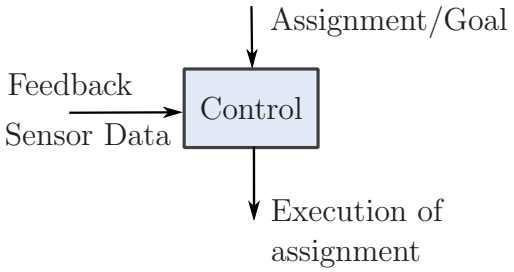


Figure 13.8: 1)

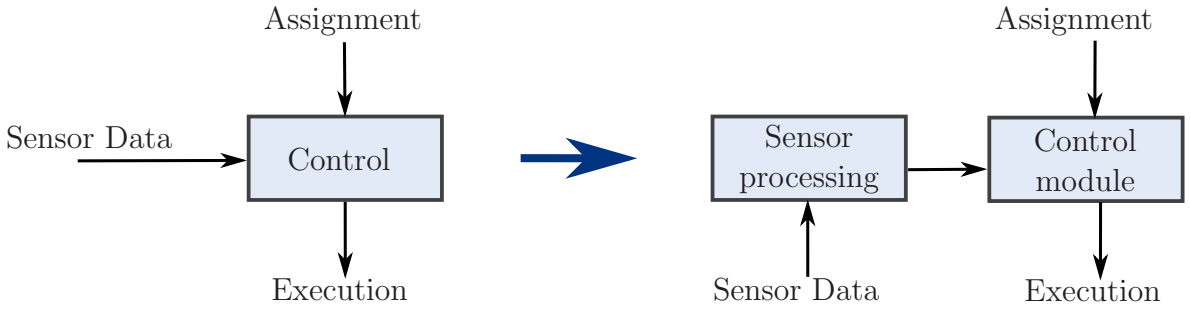


Figure 13.9: 2)

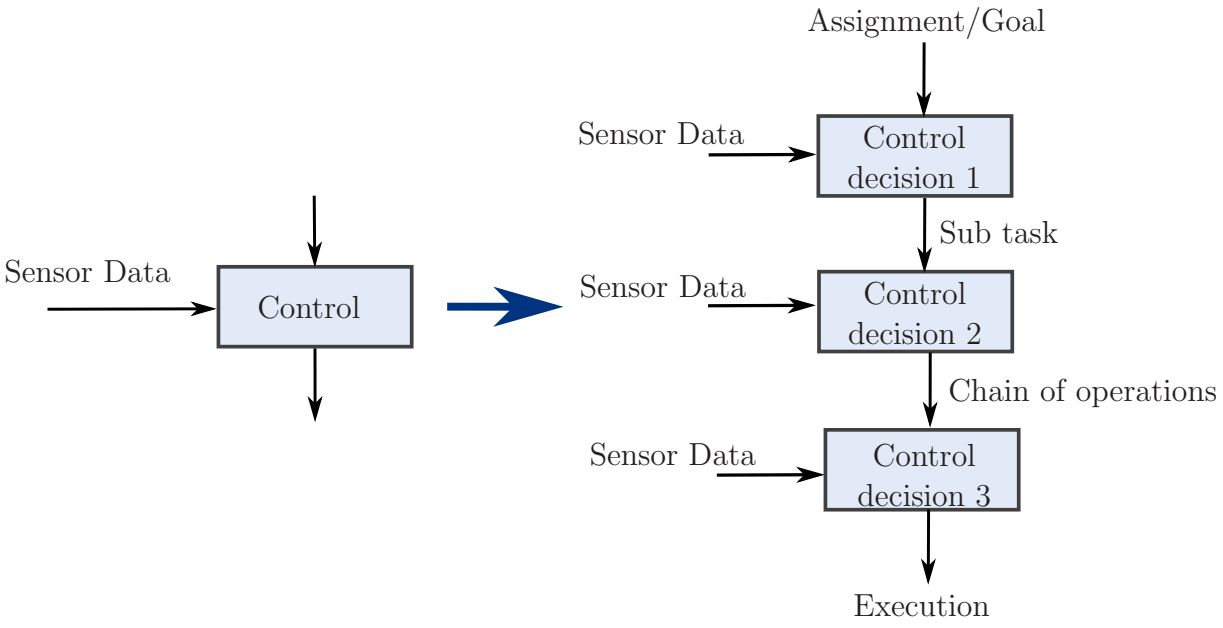


Figure 13.10: 3)

13.4.2 Modules of a Robot Architecture

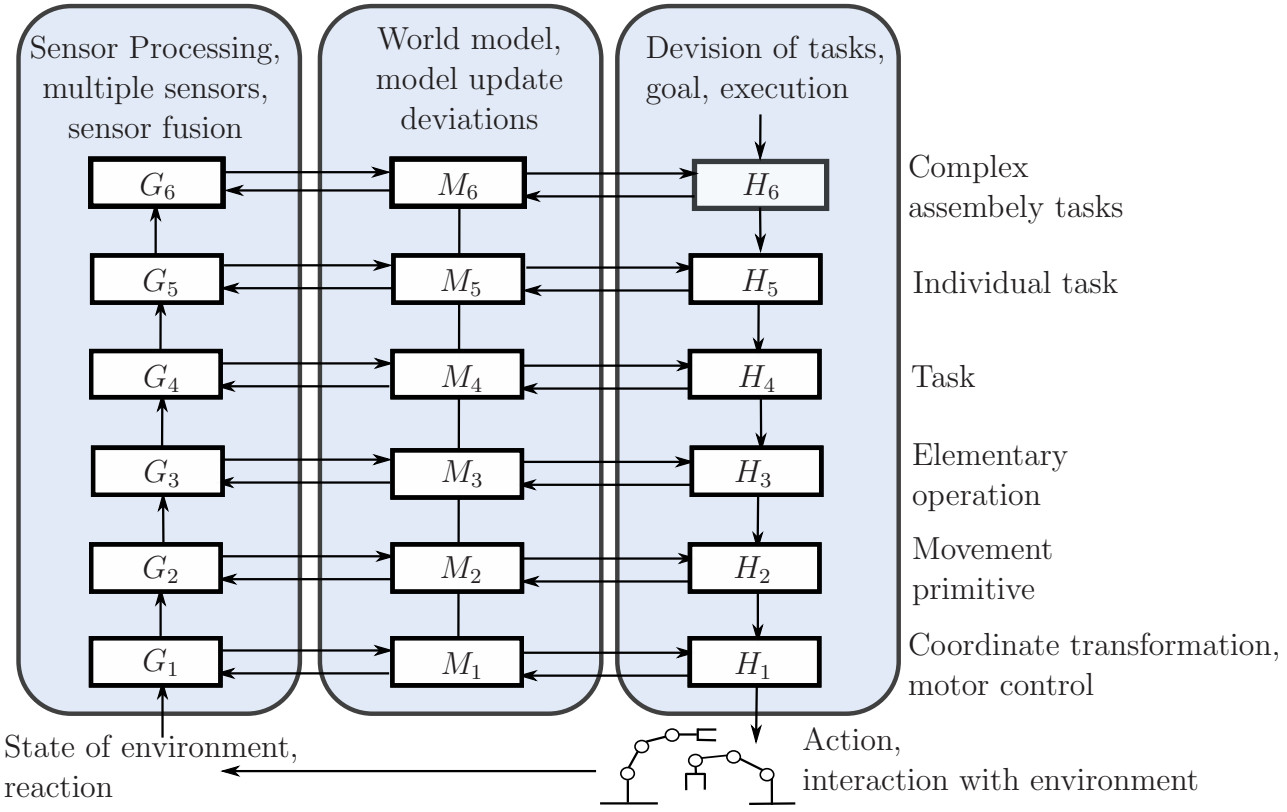


Figure 13.11: Modules of a Robot Architecture

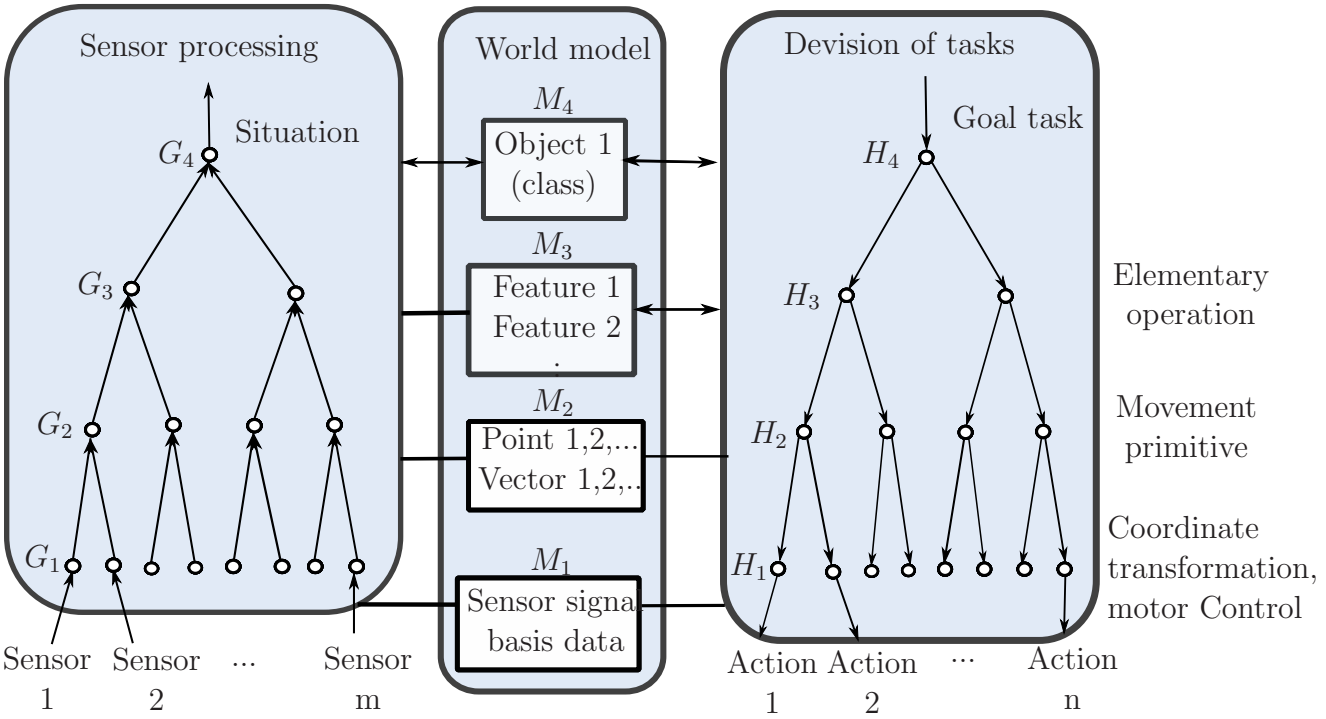


Figure 13.12: Sensor Processing, Planning and Division of Tasks

13.5 H-Module

The H-module contains task from the upper level. With the data from the M module, the task is broken down into sub-tasks. The M-module is updated and the subtasks are individually forwarded to the lower level.

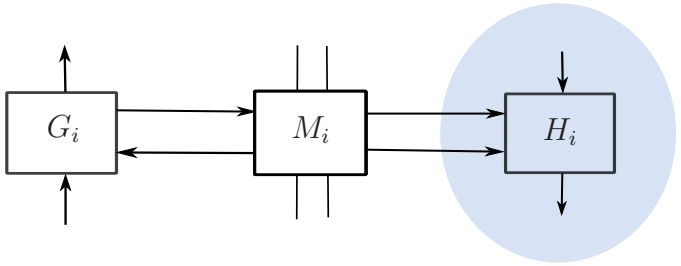


Figure 13.13: H-Module

13.5.1 Sensor Processing, Planning and Task Division

Task Division

See figure 13.14.

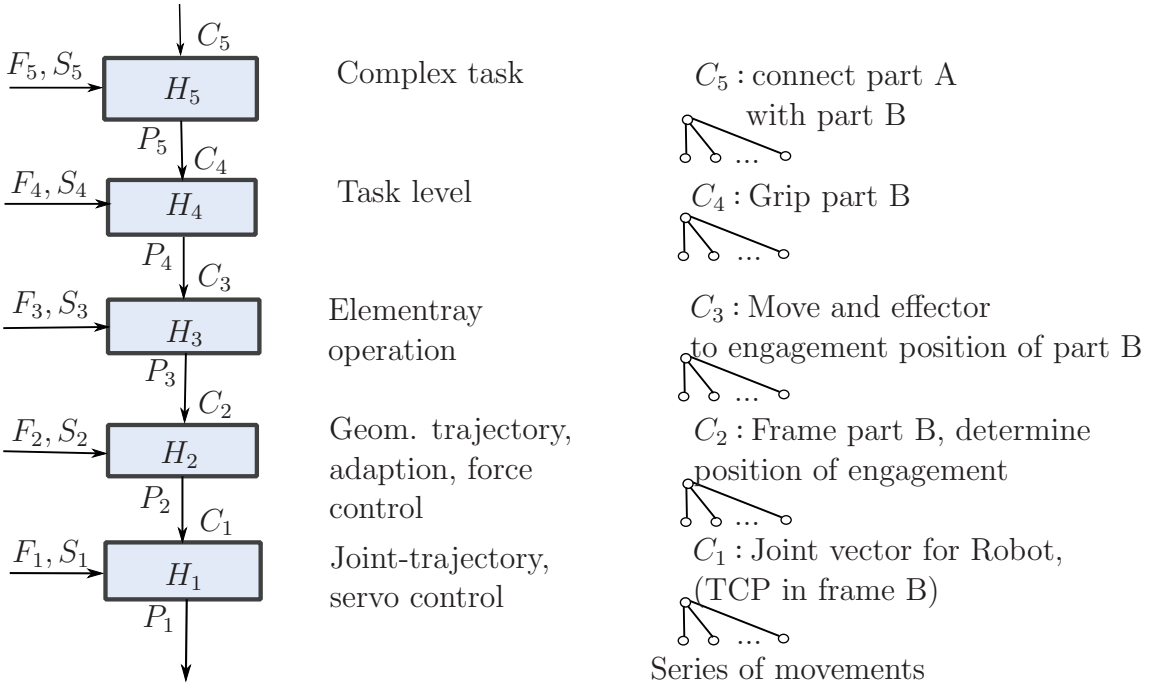


Figure 13.14: H-Module: Task Division

Division of an instruction across control levels, where the following applies:

$$\begin{aligned}
\text{Output vector} &: P_i \sim C_{i-1} \\
\text{Assignment vector} &: C_i \rightarrow \text{Series of } P_{i+1} \\
\text{Sensor vector} &: F_i \\
\text{Input vector} &: S_i = C_i + F_i \\
\text{Operator } H_i &: P_i^k = H_i(s_i^{k-1}) \quad P_i(t) = H_i(S_i(t - \Delta t))
\end{aligned}$$

Figure 13.15 shows breakdown of tasks into partial plans and their execution:

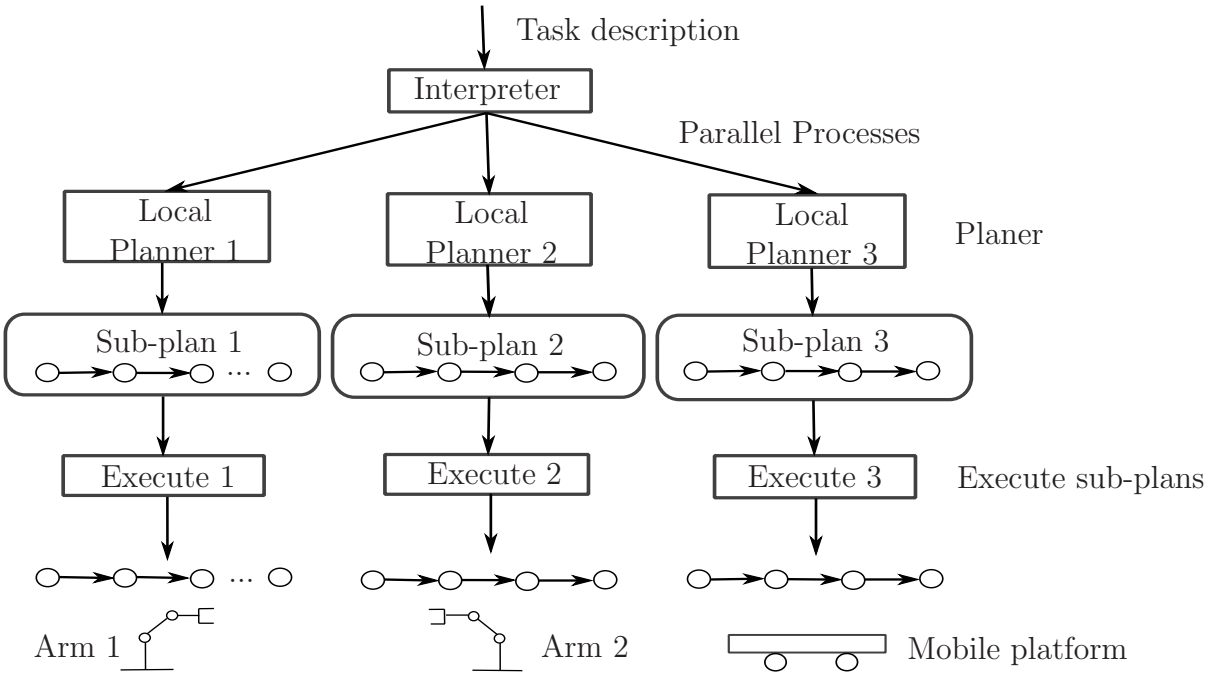


Figure 13.15: H-Module: Task breakdown into partial plans

Operations

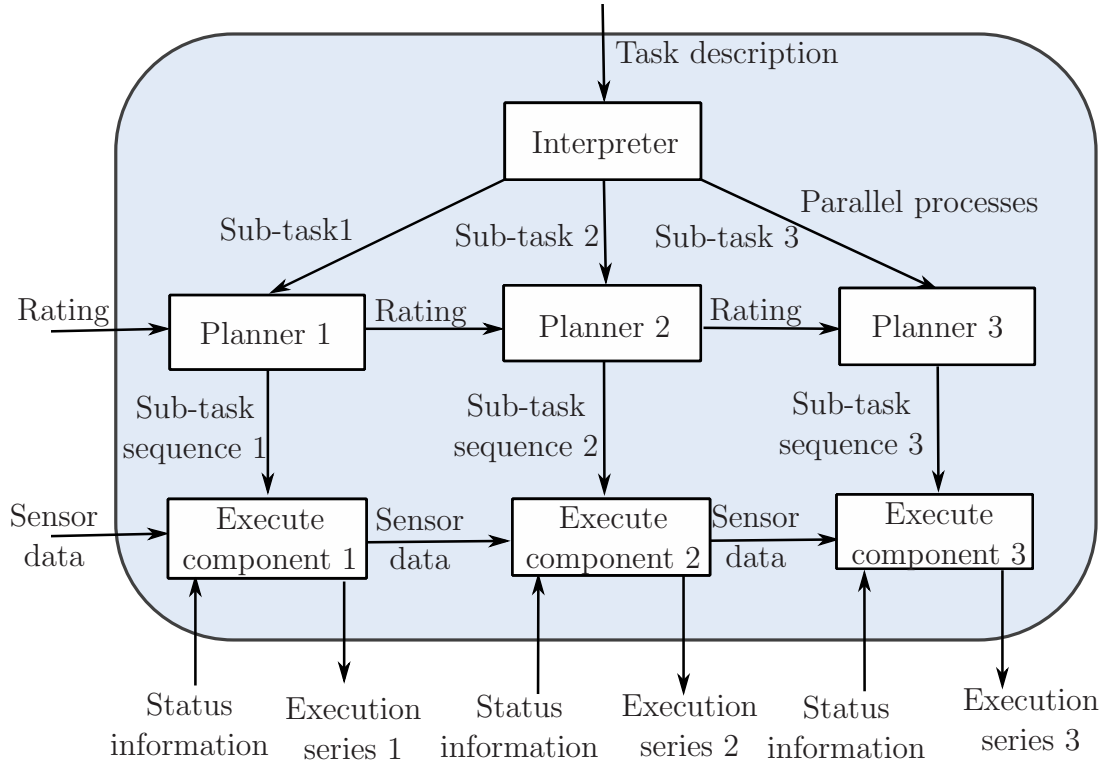


Figure 13.16: H-Module: Operations

Task Division into sub-tasks

Figure 13.17 shows task breakdown into a plan with subtasks:

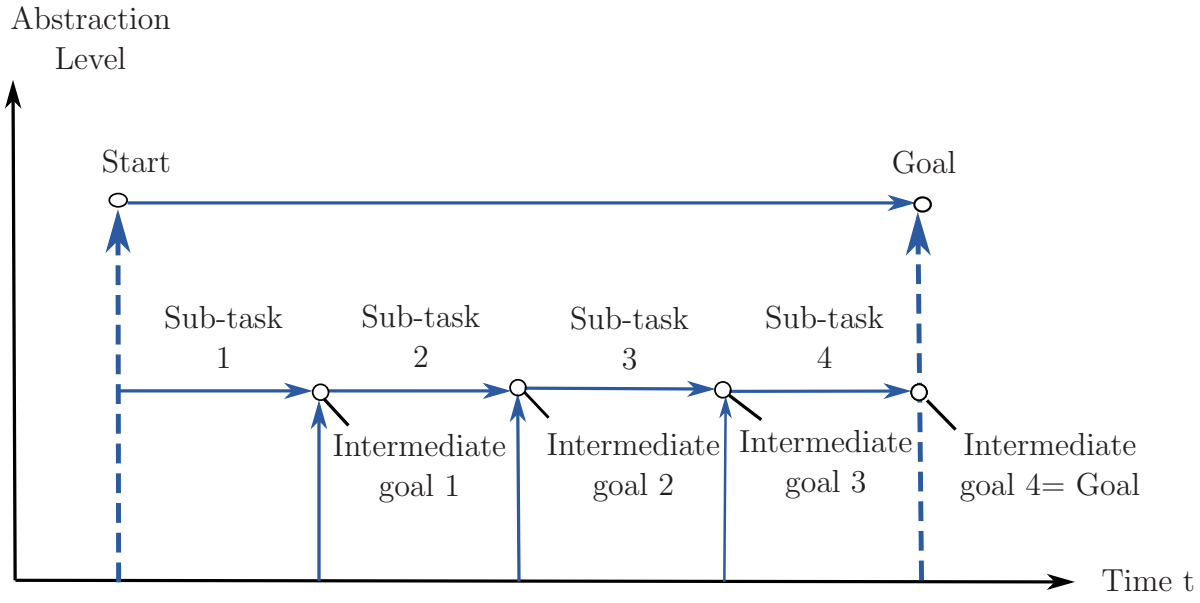


Figure 13.17: H-module: Task Division into a Plan

Input Series as Linear Graph

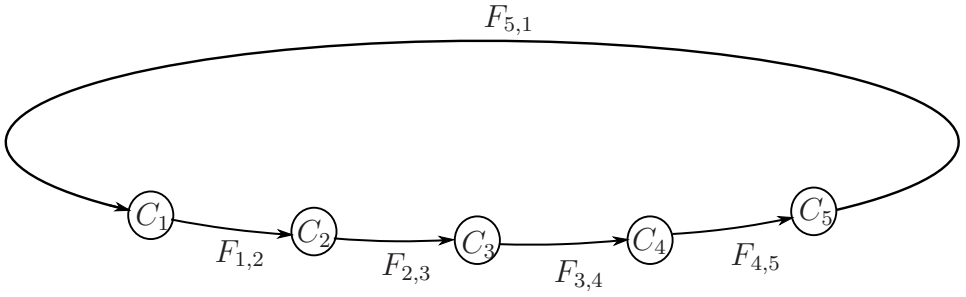


Figure 13.18: Simple Linear Case

C_i : Command i

$F_{i,j}$: Transition from i to j

Input Series as Tree

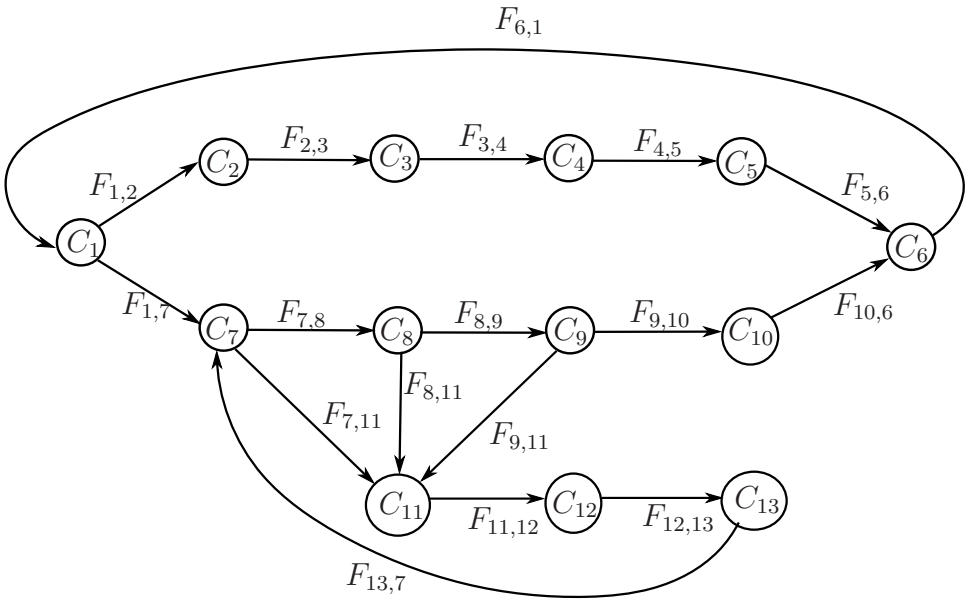


Figure 13.19: Input Series as Tree

Example: Assembly Sequence

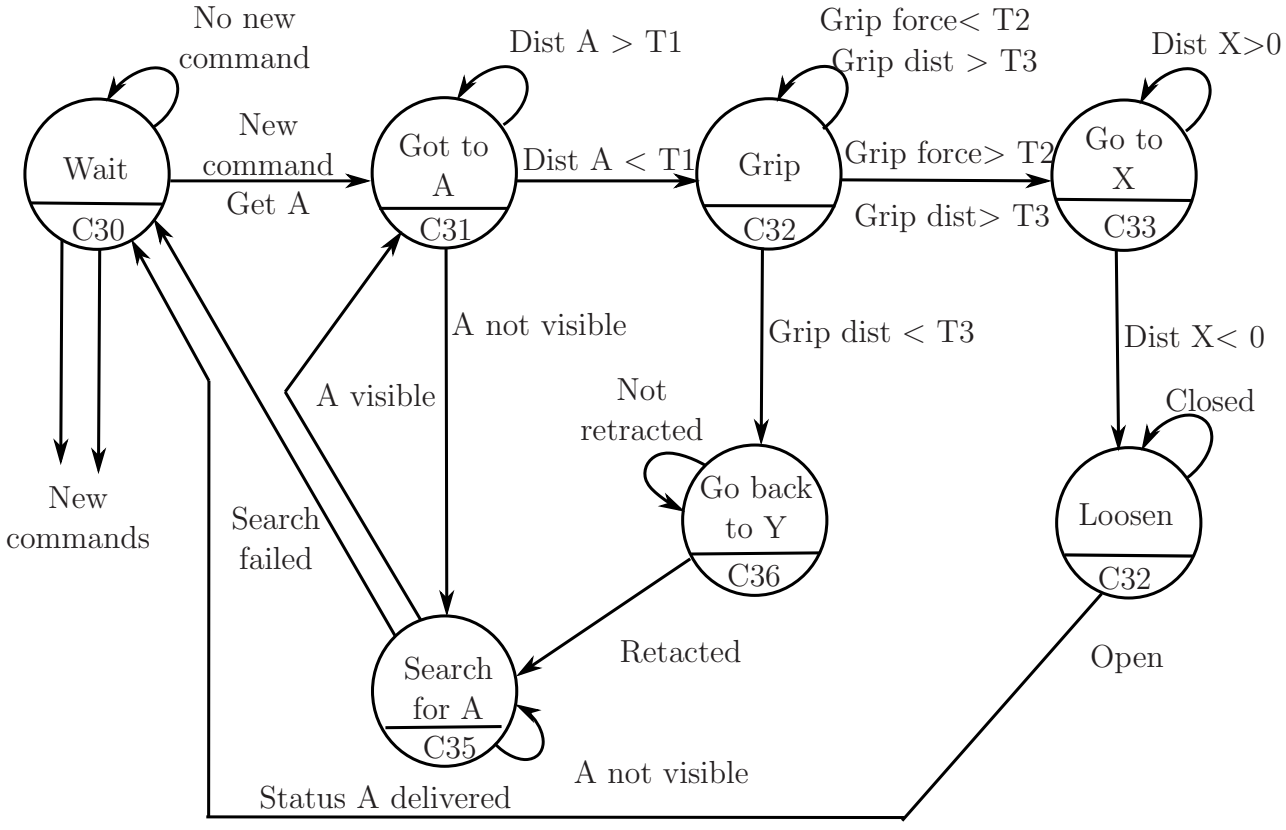


Figure 13.20: H-Module: Example Assembly equence

13.6 G-Module

G-module contains status/sensor information from the lower level. This data is processed with the M-module. Then, M-module is updated. It sends information for the next higher level.

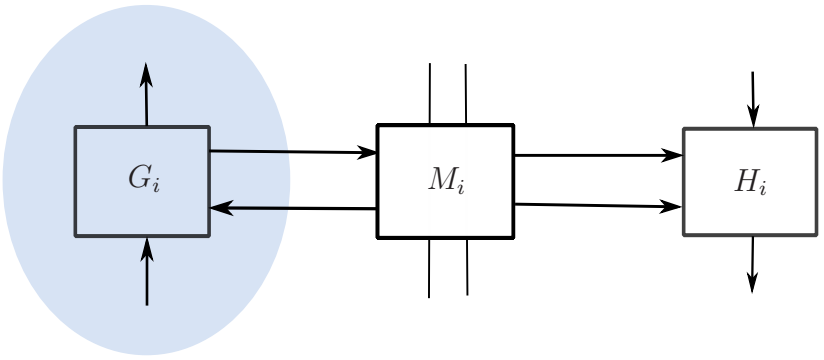


Figure 13.21: G-Module

For comparing predicted model states with measured ones, see figure 13.22:

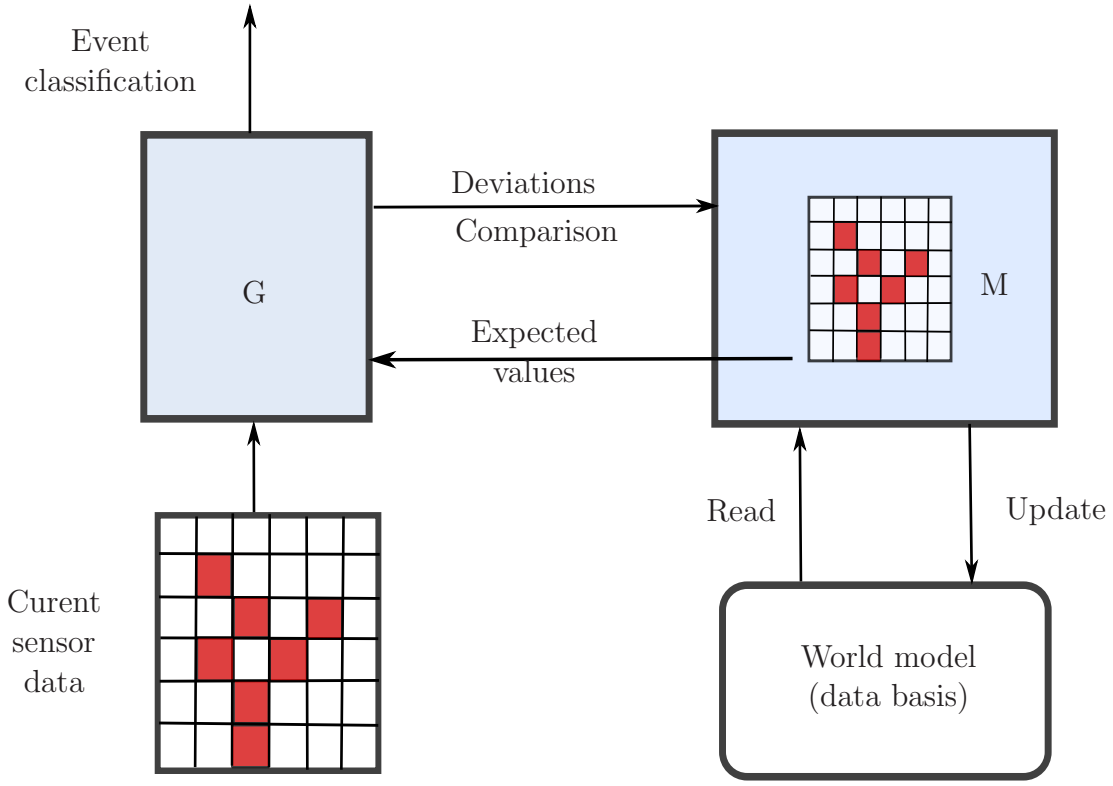


Figure 13.22: G-Module

G-Module: Signal Processing

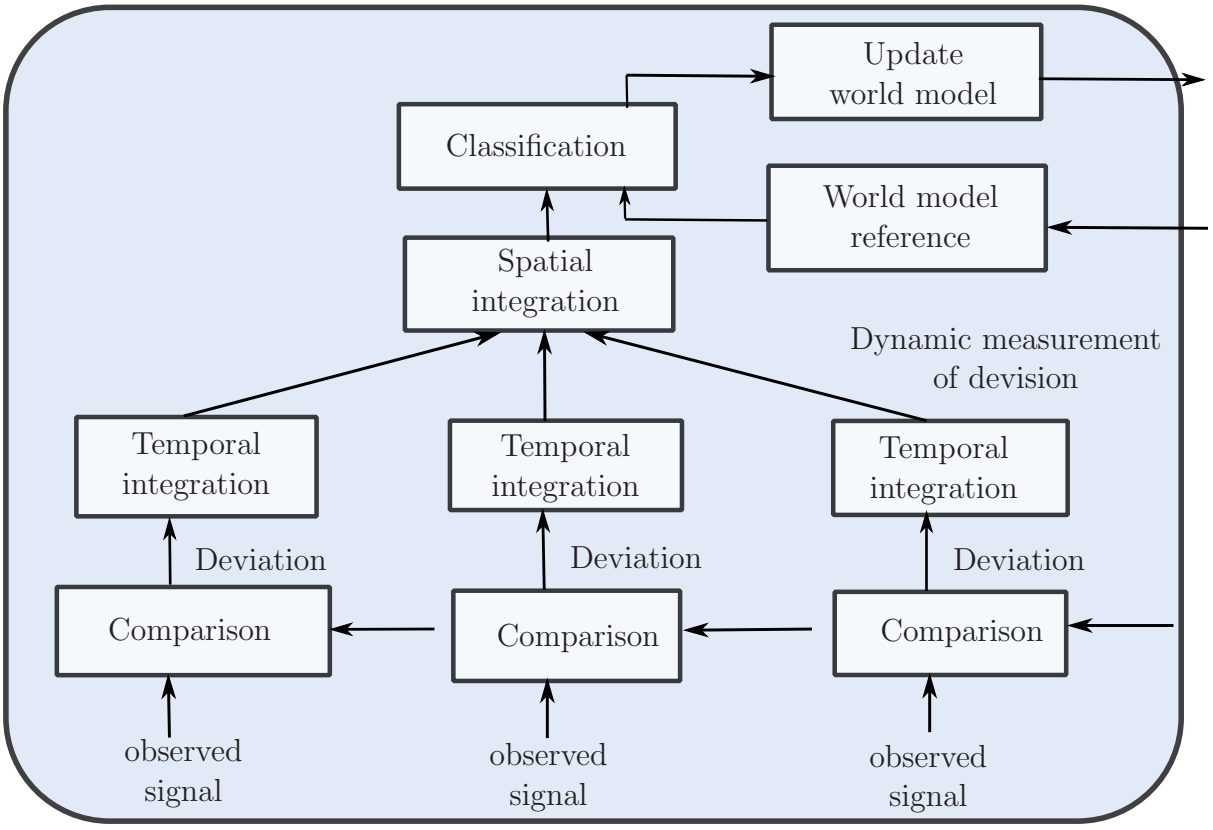


Figure 13.23: G-Module: Signal Processing

G-Module: Application

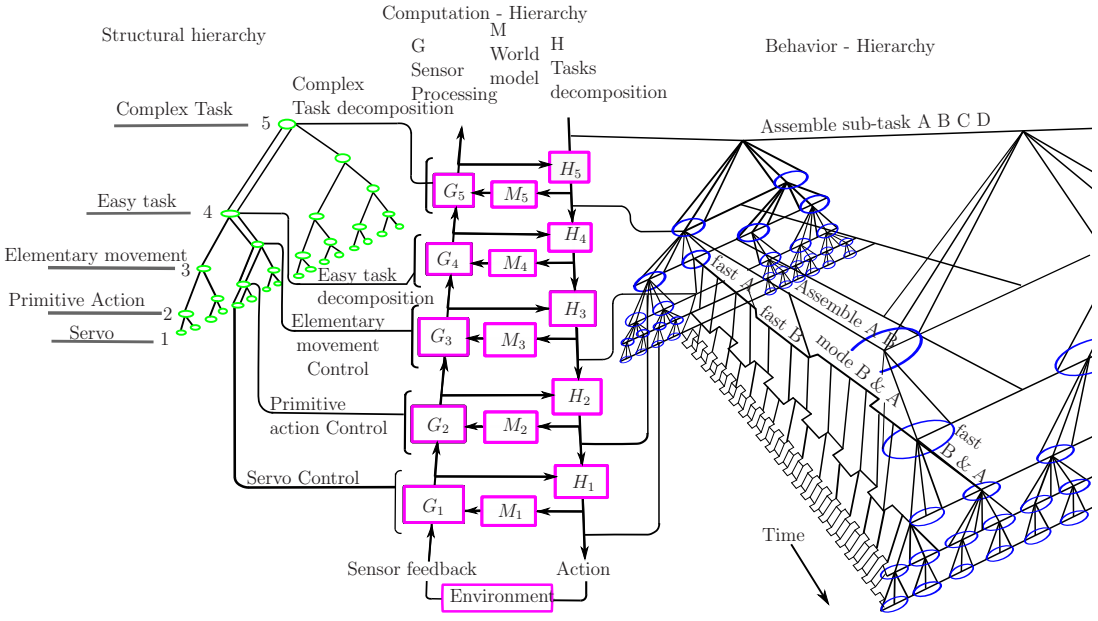


Figure 13.24: G-Module: Application

13.7 M-Module

M-Module consist of a single module contains data with corresponding abstraction level. G- and H-Modules extract this data with the degree of abstraction of the respective level. Because of a difference an error is detected between the predicted (M-module) and the actual data (G-module).

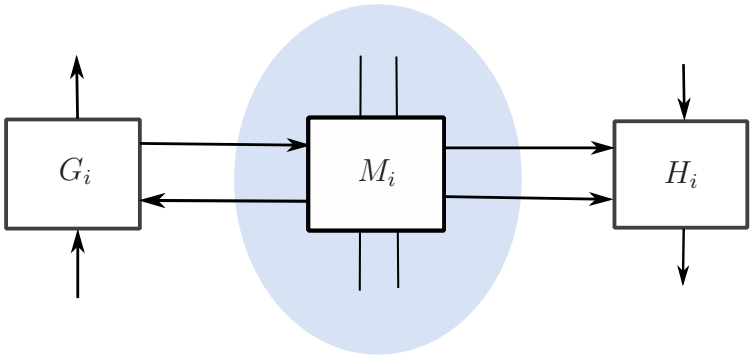


Figure 13.25: M-Module

World Model

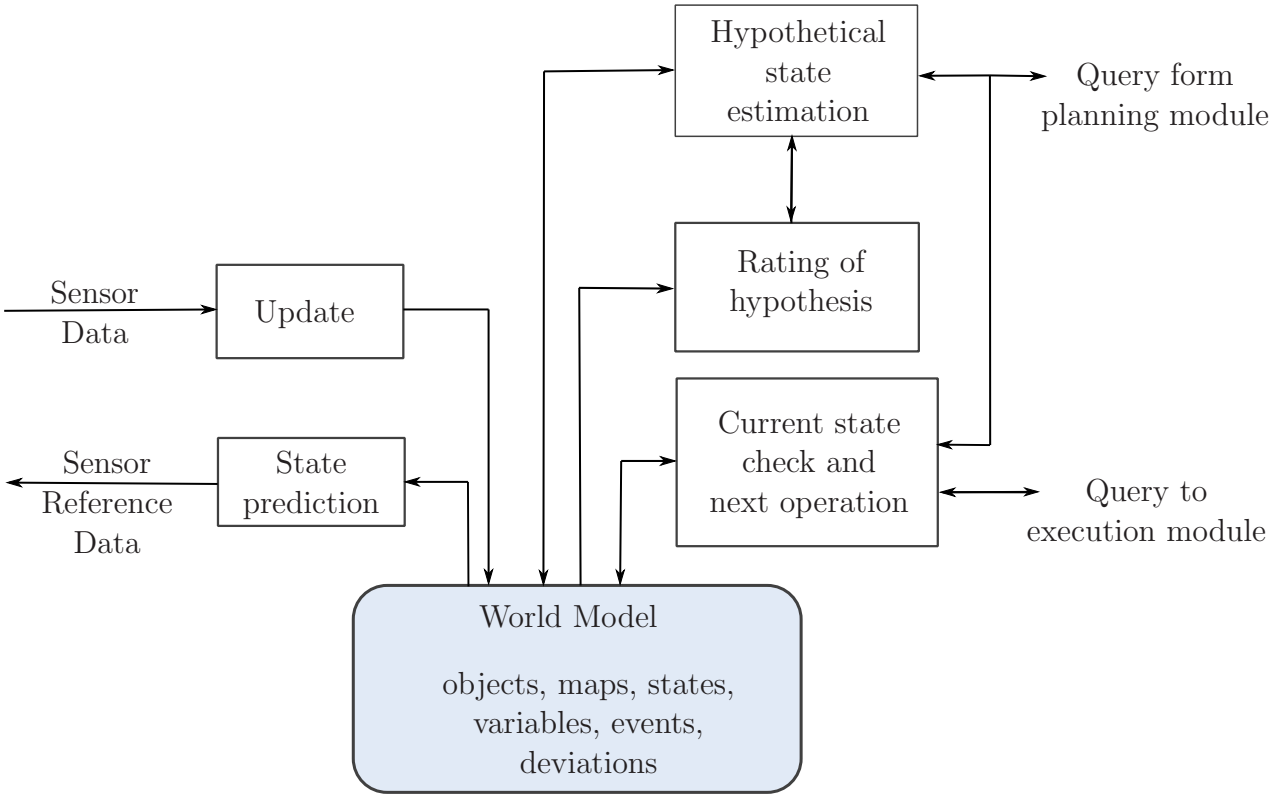


Figure 13.26: M-Module: Operarions on the World Model

Example: T3 – Intelligent Control Architecture

The developers are NASA and Metric Inc. Robotics and Automation Group. There are three interacting levels, also shown in figure 13.27:

- Dynamic reprogrammable set of reactive skills
- Coordination via *skill-manager*
- Sequential control for activation and deactivation of skills to solve specific tasks
- *Reactive action packages* (RAPs)
- planning components to determine goals, resources and timings (*adversarial planner* (AP))

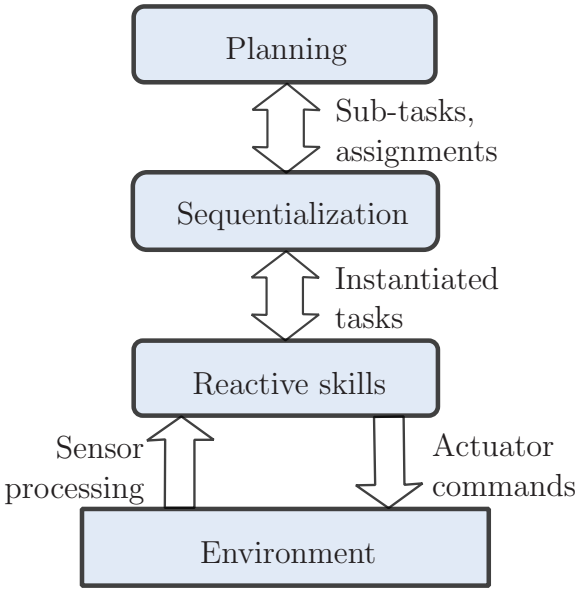


Figure 13.27: T3 - Intelligent Control Architecture

Example: T3 – Sequential Control

- Task division (series of actions or RAPs)
- Further division and execution of RAPs on the sequential level by the RAP-interpreter
- Activation of even monitors to notify the sequential control of certain events
- Activated skills effect the events and the environment
- Sequential level terminates actions, if:
  - monitored events occur
  - certain time limit is exhausted
  - changes in the plan are send by the deliberation level

13.8 Distributed Deliberative Architectures

There are a set of specialized subsystems that communicate via a central component. An example of this is the *Nav-Lab-System*. (See figure 13.28).

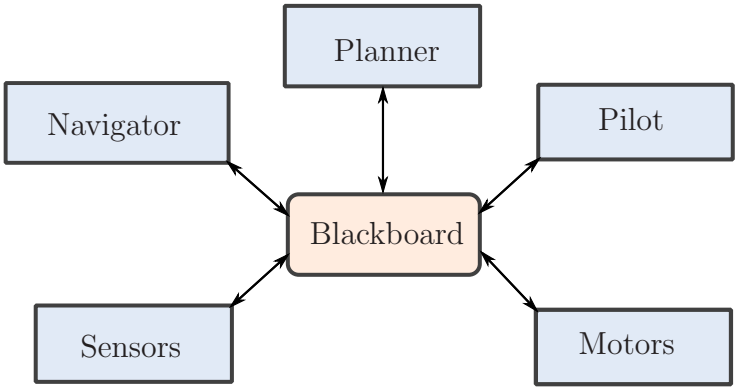


Figure 13.28: Nav-Lab-System

13.9 Hierarchical reactive Architectures

Control via behaviors or reflexes. The system reacts to certain sensor stimulus (environmental situation). It is arranged in behavioral levels and hierarchical structure on competence level. An example of this is the subsumption architecture and behavior schemes as finite state machine networks. Requirements for the control systems are:

Multiple Goals: Robot can pursue multiple, potentially contradicting, goals.

Multiple Sensors: Fusion of multiple sensors.
problem: Inconsistency

Robustness: Robust with respect to noise and sensor failure.

Additivity: Possibility to add new sensors or computing power

13.9.1 Subsumption Architecture

The architecture is divided into Levels of competence. At each level, the behavior system is a complete control system, and the higher level behaviors can overwrite output of lower levels. This enables a hierarchical structure. See figure 13.29.

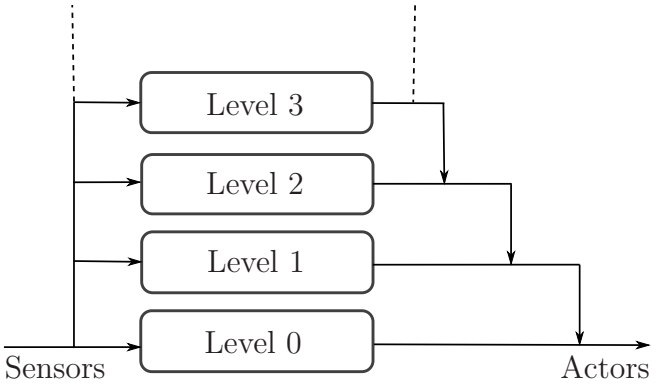


Figure 13.29: Subsumption Architecture

Mobot-System

Hierarchical structure of competence level of the robot Mobot:

1. Collision avoidance
2. Random movement in environment
3. Exploration of environment
4. Mapping and path planning
5. Capture changes in local environment
6. Detect known objects and deduct effects on object oriented tasks
7. Create plan and execute it
8. Deduct effects on environments and resulting modifications in plans

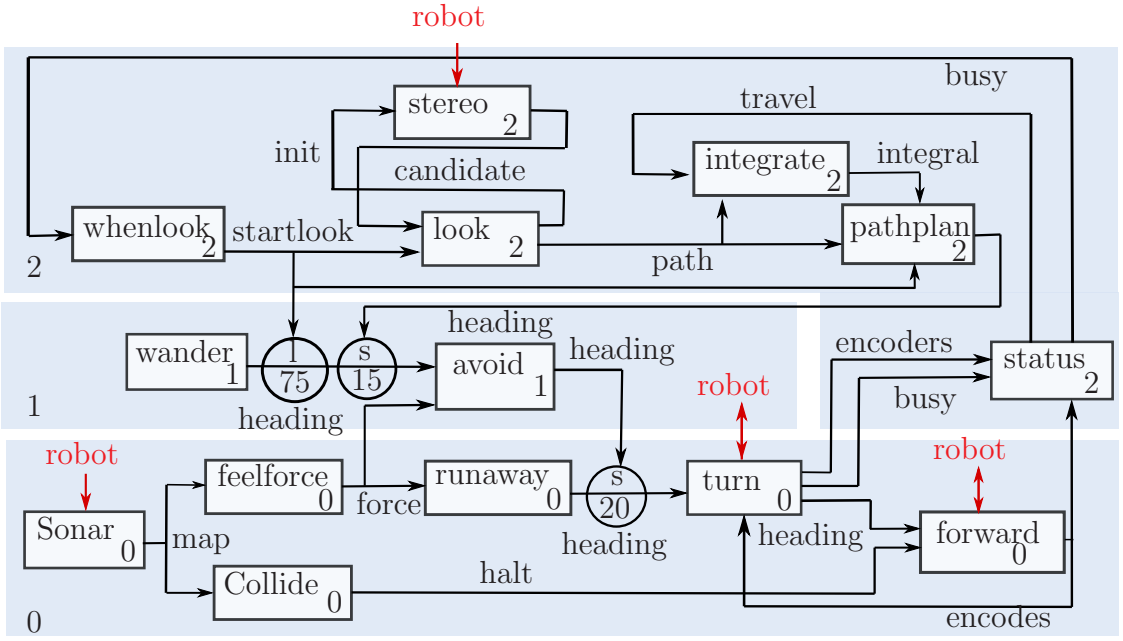


Figure 13.30: Subsumption Architecture: Mobot-System

13.10 Distributed Reactive Architectures

Distributed Reactive Architectures are a set of independent subsystems with identical (competing) behavior patterns. The coordination takes place via a behavior pattern and the prioritization takes place via an arbitration process, such as the decaying activation. *Multi-agent systems* are an example of such an architecture.

Their characteristics are self-organization, negotiation, associative memorization of information, and recognition of certain patterns.

13.10.1 CoMRoS: Multi-Agent-Robot-Architecture

CoMRoS: COoperative Mobile RObots Stuttgart

The goals are:

- Universal architecture for autonomous mobile robots
- Predictable/Computable distributed negotiations
- Reusability
- Dynamic configuration
- Robustness
- Safety

The scenarios are:

- Driving in formation
- Cooperative transport
- RoboCup

Figure 13.31 shows the division in three levels:

Strategic Level: Complete planning of actors

Tactic Level: Context aware planning, control of execution, contains assignments from strategical level

Reflex Level: Process with mainly reactive behaviors, partially directly connected to Hardware

Each level contains multiple software components (elementary-agents or autonomous cycles).

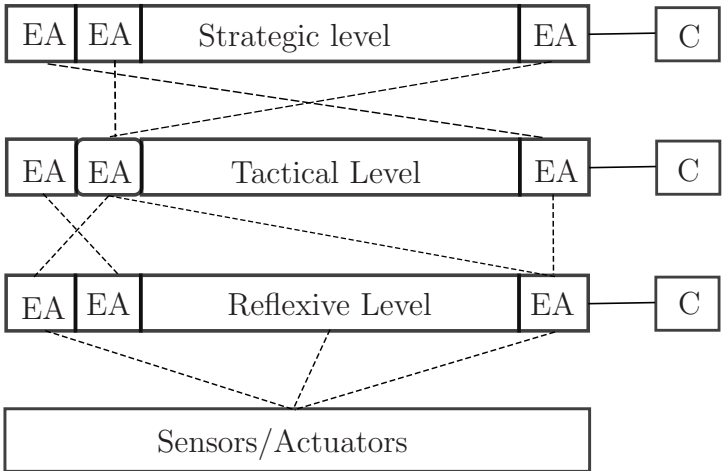


Figure 13.31: CoMRoS: Multi Agent Robot Architecture

Elementary agents (EA) are autonomous and can be divided into a brain (decision unit) and a body (sensor-actor cycle). (see figure 13.32).

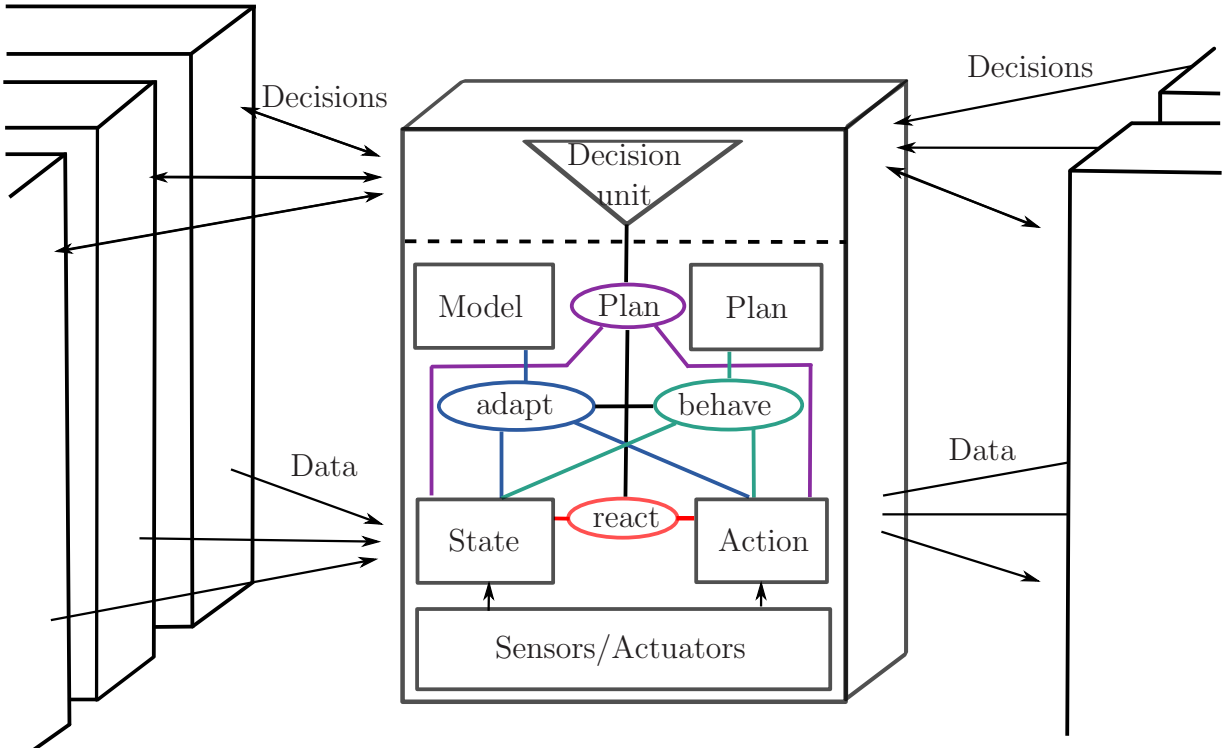


Figure 13.32: CoMRoS: Elementray Agent

The function and structure of an EA:

- Sensor-actor-cycle
- Connection of sensors and actors define aitonomous areas of EA

Decision unit: The decision unit controls this abstract control cycle. Furthermore, it leads the negotiations between the agents through interaction between the decision units based on a dynamically configurable networks.

The nodes of the sensor-actuator-cycle are:

- World model
- Plans
- Actions
- States

Node connection by four cycles

Reactions: Feedback between sensors and actors

Behaviors: Defines the typical behavior of agents

Adaption: Updates the structure of the model

Planer: Creates new plans for agents

14. Robot Programming

14.1 Programming of Industrial Robots

Programming is a key aspect to any robotic system. It relays vital information to an industrial robot making it capable of functioning and carrying out specific tasks. Robotic programming was once viewed as daunting as it required complex lines of code. Fortunately, advancements in robotics has led to the development of more intuitive programming methods that are user friendly to even the most novice of operators.

The operating system must be real-time capable and have an interface to the robot controller. The programming language is a special, robot-specific standard language, such as VAL. There are libraries for standard languages (RCCL for C, ...). Robot-oriented routines are special data types such as matrices, kinematics/dynamics routines, movement commands (Cartesian, joint space) and effector commands. Task-oriented routines are the knowledge base with the environmental model, the rule base for the breakdown of tasks, planning algorithms and the division of complex tasks.

Today the most common programming methods for industrial robots are online programming offline programming, and teaching by demonstration.

14.2 Online Programming

Programming the robot when it is online or in an operational mode is called *online programming*[1]. The two types of online programming which will be discussed in the following are teach pendant programming and lead through programming.

14.2.1 Teach Pendants

Teach pendants are the most common methods of industrial robot programming. Over 90% of robots are programmed with teach pendants. Pendants (also caled teach boxes) are handheld devices connected to the robot via cables. Teach pendants containins several

[1] `https://www.youtube.com/watch?v=EA6pWwNI_wg&ab_channel=Veritasium`

buttons, switches, or a touchscreen, keyboard and are typically part of a robot's control system. They all have typically an emergency stop button since the robot is operating with the technician in the work envelope. More modern teach pendants also can be wireless. Figure 14.1 shows an example of teach pendant[2].

Figure 14.1: Teach pendant

Many of the newer FANUC teach pendants feature a touch screen. Programming through teach pendants involves an operator inputting commands through the pendant's keyboard. Many robot applications can be programed with pendants. The operator steps the robot through the program point by point while recording the coordinates for each point. The technician has the ability to send the robot to a desire position and record the point. The technician may also change the speed since a low speed as usually required for carefull positioning or while test running through a new or modified routine. Once the technician has recorded and tested the program, the teach pendant is disconnected and the robot can operate the stored program at full speed. Teach pendant programming is offten associated with control path and point to point control.

A FANUC R-2000ib can be programmed to weld car frames. While a FANUC LR Mate 200id is capable of locating parts to be placed on a conveyor due to the instructions relayed to it through an operator's input using a teach pendant. Images 14.2[3] and 14.3[4] show a FANUC R-2000ib and a FANUC LR Mate 200id respectively.

[2] https://www.researchgate.net/figure/An-example-teach-pendant-Kuka-LWR-controller-Such-interfaces-fig3_282234684

[3] https://www.robots.com/robots/fanuc-r-2000ib-165f

[4] https://www.fanuc.eu/ch/de/roboter/roboterfilter-seite/lrmate-serie/lrmate-200id-14l

Figure 14.2: FANUC R-2000ib

Figure 14.3: FANUC LR Mate 200id

Teach pendants provide many advantages when it comes to robot programming. some of advantages are:

- Most traditional industrial robots come with a teach pendant, which makes them familiar to technicians.
- They allow precise positioning, as the robot can be programmed using numerical coordinates, in either world coordinates, robot coordinates or another coordinate system.
- Teach pendants are great for simple movements, such as painting in a straight line or over a large flat surface.
- The vast majority of industrial robots come with teach pendants making it easy to get robots up and running without needing to purchase or integrate additional programming software.

The disadvantages af teach pendant method are:

- Disruptive to the whole system due to robot downtime. The robot must be put into teach mode and all operations using the robot halted until it has been programmed.
- Requires training to learn and program.
- Might be difficult for skilled crafts people who are unfamiliar with programming.

14.2.2 Lead Through Methods

Lead through methods involve programming robots through demonstration associated with continious path control. During this process an operator physically moves a robot through a desired task. This method also calls lead through the nose. Lead through programming

is best for detailed applications as it simplifies programming by eliminating the need to write several lines of complex code. It is also faster than other programming methods but is not suited for simple or straightforward applications. This method has declined in popularity with traditional industrial robots as many have become too large or heavy to physically manipulate their robotic arms. However, this method has gained popularity for programming collaborative robots. The FANUC CR-35iA is one example of a collaborative robot that uses demonstration programming. Figure 14.4[5] shows a FANUC-CR35iA.

Figure 14.4: FANUC CR-35iA

During the lead-through programming, the user manually moves the robot arm with the tool using his hands. The method makes it possible for non-programmers to control and program robots. However, it still lacks many of the capabilities of traditional robot programming. During lead-through programming, it is desired to make the robot compliant in one or more directions to enable a user to move the robot by hand to a desired position while recording each step.

WO2010/088959 discloses a method for programming an industrial robot by lead-through. The robot controller is switched into a floating control mode before the programming begins. The robot is programmed by means of lead-through at the same time as the robot controller is in the floating control mode. When the robot is in the floating control mode, the stiffness of the manipulator is reduced in one or more Cartesian directions and orientations. By switching the robot into the floating control mode the manipulator is made compliant and easy to move by hand. The stiffness is reduced so that the manipulator becomes compliant, but not resilient. When the robot is in the floating control mode, it is easy for the user to move the robot by pushing or pulling the robot to a desired position, and when the user stops the movement, the manipulator will stay in the desired position, since there is no resilience in the floating control mode.

During lead-through programming, the robot controller usually is passively controlled. When the robot is passively controlled, the stiffness of the manipulator is reduced in all

[5] `https://www.fanuc.eu/at/de/roboter/roboterfilter-seite/kollaborierende-roboter/collaborative-cr35ia`

Cartesian directions and orientations and by that the manipulator is made fully compliant. With fully compliant is meant that the manipulator is made compliant in all possible directions and orientations, which enables the user to lead the robot arm around freely in any direction and orientation. Further, the manipulator will stay in the desired position when the user stops the movement of the robot arm during passive lead-through. This is accomplished by controlling the robot to compensate for gravity forces acting on the robot arm. A passively controlled robot makes it easy and fast for the user to program large and sweeping motions during lead-through programming. When the user needs the robot to make large sweeping motions he grasps the robot arm firmly and leads the arm freely to the desired position. A disadvantage with the passive control of the robot during lead-through programming is that it is difficult to program more precise movements, such as linear movements.

This problem can be solved by actively controlling the movements of the robot during lead-through programming. For example, the movements of the robot can be restricted so that the manipulator only moves in one direction.

US2015/0081098 discloses a method for active lead-through programming of a robot having a force sensor. The sensor detects a guidance force applied to the robot arm by an operator. The movements of the robot are controlled using force control in such manner that a pre-specified reference point associated with the robot arm is moved only in a selected direction as a result of movement of the robot arm by the operator, during manually-guided adjustment of the position and orientation of the robot arm. This method is suitable for programming linear movements.

When programming an industrial robot the user requires the robot to be both fast and precise. These criteria can be easily met during traditional programming of the robot using coded commands, as the user can type the exact position and determined speeds. However, when using lead-through programming it can be difficult to be fast as well as precise.

The advantages are:

- Easy to use
- Little or no programming skill is required.
- Speed of programming is high

The disadvantages are:

- Required production line shutdown
- The technician should be inside of work envelope while the robot is operating which exposes him to great risk of damage

See also [Henriksson 20]

14.3 Offline Programming (OLP)

Offline programming (OLP) software is becoming an increasingly essential tool for manufacturing professionals charged with planning new robot work cells; and it's being used for much more than developing the robot program. OLP software helps with planning and optimizing the work cell design, virtually commissioning the robot, and accelerating the time to production. It uses 3D CAD data to create a virtual model of the robot and work cell, and simulate the processes and workflows inside and outside the cell – a powerful tool for engineers and planners to evaluate trade-offs and make better decisions. OLP software provides a meaningful return on investment for many types of automation projects, by saving time, improving productivity, and helping manufacturers to identify opportunities for cost savings.

There are many approaches and strategies to OLP, and we're not advocating for one approach over another. The process we're describing is adapted from [Pan 12].

14.3.1 Steps of the OLP Process

1. **Generation of Models**[6]

   The first step to OLP is to create a virtual model of the work cell. This involves creating or obtaining 3D CAD models of the equipment, work pieces, enclosure, tools, and other resources and fixtures that will be in the work cell; and importing them into your OLP software. There may be extra steps in order to simulate resources and processes, depending on the OLP software being used. Accuracy of the models and process related information used is critical to generating a reliable simulation of the process and error-free offline program for the robots.

   The creation of the simulation is very complex. Compared to this, the optimization of motion sequences is cheaper. There is a risk of incorrect or incomplete simulation (Both kinematic and dynamic parameters must be as exact as possible, otherwise damage to robots is possible)

[6] `https://www.visualcomponents.com/resources/blog/steps-of-the-olp-process`

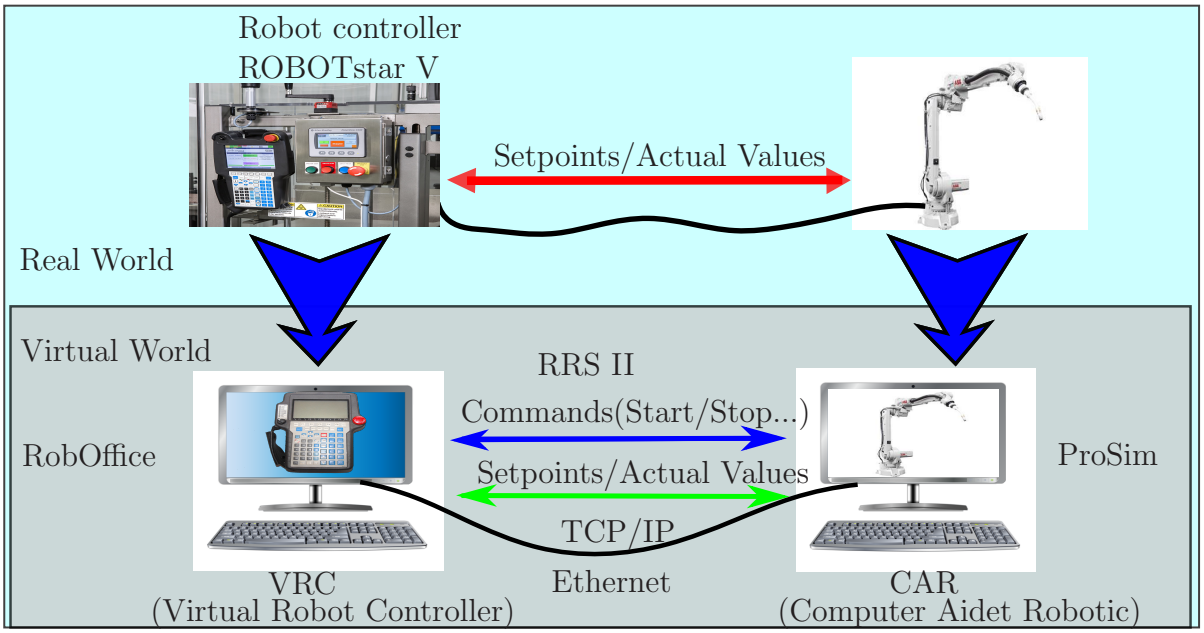


Figure 14.5: Robot Simulation: Realistic Robot Simulation

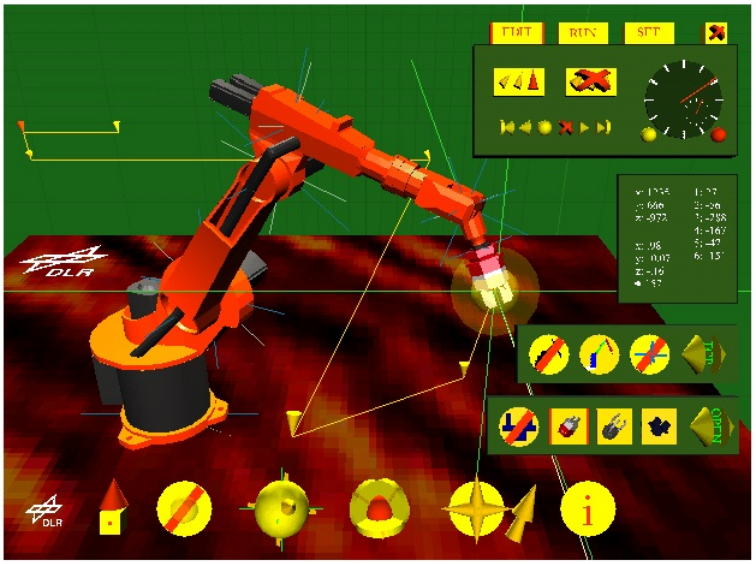


Figure 14.6: Robot Simulation: VRML 2.0 DLR and KUKA

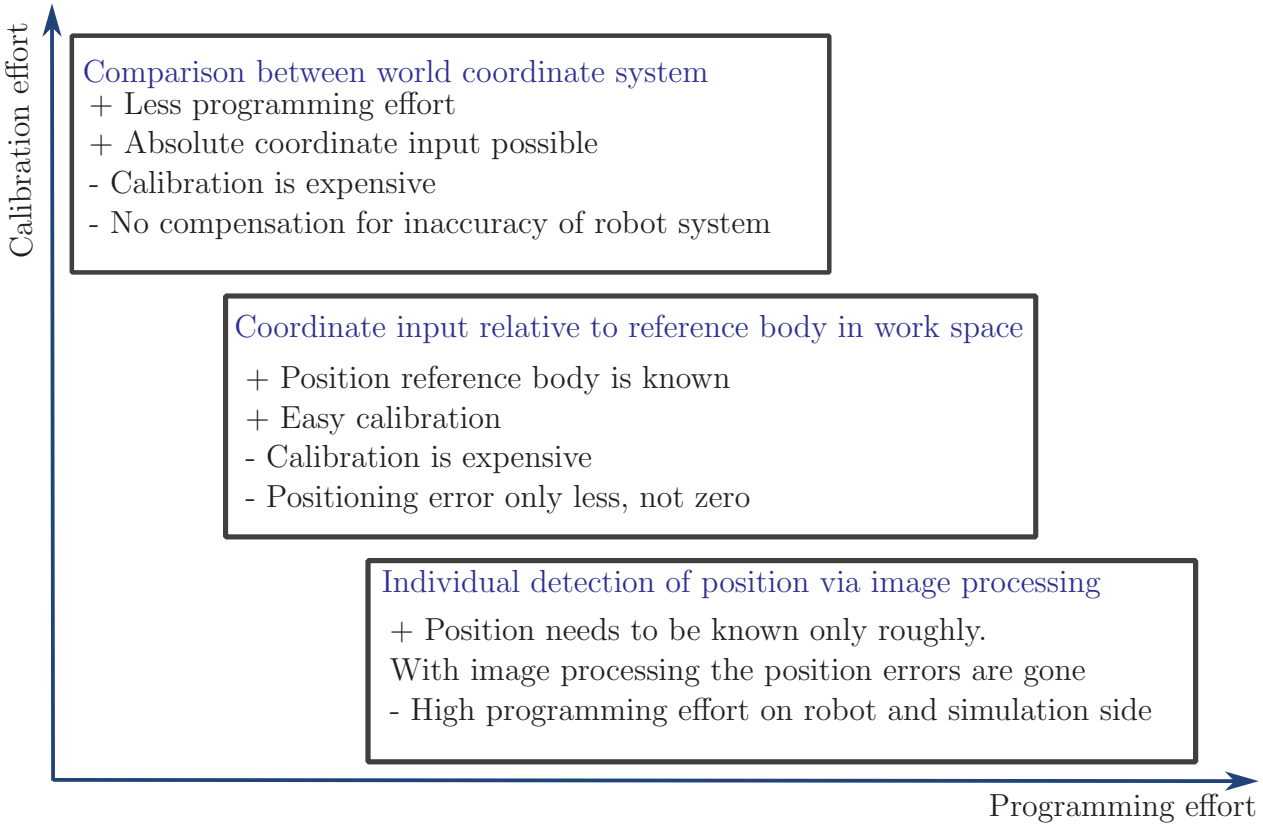


Figure 14.7: Comparison Between Simulation And Robot

2. **Tool Path Generation**

   Tool path generation involves extracting robot positions from 3D CAD data with a specific tool center point – the point in relation to which all robot positioning is defined. Many OLP software packages can do this automatically, and have built-in functions to automatically generate paths from features of the CAD models, such as corners, edges, or other geometry features.

3. **Process Optimization**

   Process optimization incorporates trajectory planning, process planning, and tooling design. It's an iterative design loop with a number of factors and trade offs that need to be considered, so simulation helps significantly with this process.

   - **Trajectory planning** involves determining the best route for the robot to make, from Point A to Point B. While it might sound simple, this isn't a trivial task; and it's not always about planning the fastest or shortest route. Robot work cells are typically designed in compact configurations, so motions and trajectories have to be planned carefully to avoid unintended interactions between the robot and other objects in the cell. There are several factors that need to be considered, such as motion type, joint configuration / speed / acceleration, reachability, and collision detection and avoidance.
   - **Process planning** involves planning the processes and workflow in the work cell. The major constraints for this step are budget, productivity, and quality, but

there are several factors and trade offs that need to be considered. This step includes layout design, resource selection (including robots and other equipment), maintenance considerations, and sequence optimization.

- **Tooling design** involves the selection, modification, and placement of tools. This includes robotic end of arm tooling and other tools that touch the work piece, such as positioner faceplates and clamps.

4. **Post Processing**

   After the robot program has been verified in the simulation environment, it needs to be implemented to the real robot. But first, the program must be converted into the language of the target robot. This step is called post processing. Post processors; the drivers that convert the robot programs; need to be created to perform this conversion. Post processors are specific to the robot brand, application, and other customer specific requirements for safety, usability, performance, etc. It's rare to develop post processors from scratch; it requires lots of work and there are many companies that develop/sell them commercially; but they still require some customization for your specific application and setup. As a gross estimate, about 80% of the commands are the same for each post processor, but what changes is the local customization for each application, customer specific requirements, and special macro commands that have to be called at the beginning or end of a program.

5. **Initial Calibration**

   Initial calibration involves calibrating errors between the work cell and virtual model, and updating the virtual model to match. The goal is to ensure the offline program is running at 100%, with no unplanned operator intervention. Calibration is performed using a tool center point on the shop floor. Depending on the application, it's also possible to calibrate without using a tool center point; as an industrial robot can be used as its own coordinate measuring machine to determine the relative positions of critical components in the work cell.

   It's not always necessary to perform initial calibration offline; it can also be done in the production environment. For example, in robotic spot welding, there are usually around 10 to 20 points that need to be programmed in the robot. It would be much faster to have the operator calibrate the robot versus doing it offline. Similarly, in arc welding, there's lots of variation between work pieces, and robots use vision sensors to compensate for work piece errors.

14.3.2 Robot Teaching with Visual Components

Some software products used for OLP, like Visual Components, have features that also make the OLP process very simple. With Visual Components Premium, you can create programs that define the actions and routines to be performed by an industrial robot. This can be done by virtually teaching the robot positions, or by using the curve-teaching tool to generate paths. You'll probably use a combination of these methods in developing your robot program. A new feature we introduced in Visual Components 4.0 is the curve-teaching tool, which greatly simplifies robot path teaching. In this section, we'll introduce you to the curve-teaching tool and how you can use it to teach a robot a path of

positions. Figures 14.8 and 14.9 show the development of programs with two-dimensional or higher-dimensioned structures. Elements used are graphics, diagrams, icons, animations.

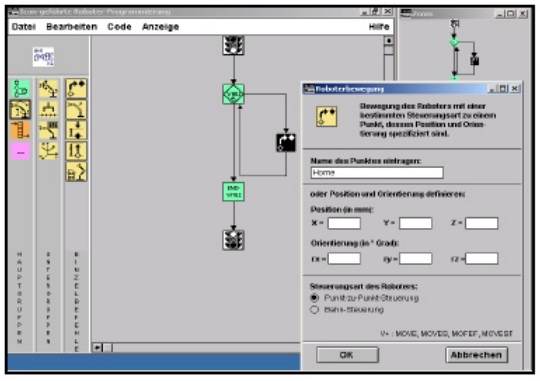


Figure 14.8: Graphics Robot Programming Prototype of the PAK

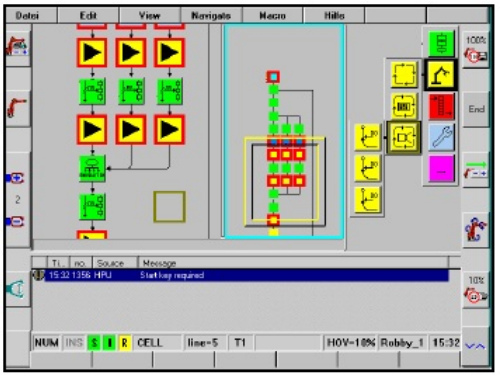


Figure 14.9: Implementation of the PAK Prototype for use with the KUKA handheld control unit

Path Satement A *Path statement* in Visual Components describes a sequence of positions for a robot to follow to complete a given path. Depending on the application, such as deburring, sealing or arc welding, a robot program may have tens or even hundreds of positions that need to be defined. As you might imagine, creating a robot program for one of these applications can take a lot of time, especially if you have to teach all of the positions.

Curve-teaching Tool The *curve-teaching tool* simplifies this by automating the robot path teaching process. It analyzes object geometries, makes path predictions, and suggests robot paths. It then automatically generates the statements in your robot program code[7].

OLP is not just developing a robot program; it's a process for designing and planning an entire robot work cell. OLP software helps manufacturers to design better production solutions by allowing them to evaluate and visualize the many requirements, constraints, and trade offs in the OLP process.

The advantages of OLP Combined with Simulation are:

- The robot can be programmed even before it is set up
- Design and engineering flaws can be identified early on. Changes can be made on the computer if necessary, avoiding the need for costly on-site modifications.
- Extensive changes to robot applications are often much easier to implement through offline programming than through direct modification of the robot.
- In the 3D computer environment, every part of the robot environment can viewed from all sides. In real life, certain perspectives are often concealed or hard to access.

The disadvantages af OLP are:

[7] `https://www.visualcomponents.com/resources/blog/teaching-robots-visual-components/`

- Virtual models will (probably) never be able to represent the real world with 100% accuracy. Programs may still need to be altered after they are applied to the real robot.
- Might take longer overall. Although offline programming reduces the downtime of the robot, it means that someone has to spend extra time developing the simulation, as well as testing it on the robot.
- Can sometimes end up wasting time sorting out simulator issues instead of solving production challenges. This could be related to the quality of the simulator[8] .

14.4 Teaching by Demonstration

Teaching by demonstration (and more specific methods like Kinetiq teaching) offers an intuitive addition to the classic teach pendant. These methods involve moving the robot around, either by manipulation a force sensor or a joystick attached to the robot wrist just above the end effector. As with the teach pendant, the operator stores each position in the robot computer. Many collaborative robots have incorporated this programming method into their robots, as it is easy for operators to get started immediately using the robot with their applications.

The advantages of Teaching by Demonstration are:

- Quicker than traditional teach pendants. It removes the need for multiple button pressing, allowing the operator to simply move the robot to the desired position.
- More intuitive than both traditional teach pendants and simulation programs, as the task is programmed in almost the same way a human operator would perform it. This makes it simple for operators to learn. Generally, this method requires no knowledge of programming concepts or being familiar with 3D CAD environments (as simulation does).
- Very good for detailed tasks which would require many lines of code to achieve the same effect, such as welding or painting of intricate shapes.

The disadvantages of Teaching by Demonstration are:

- As with traditional a teach pendant, this method uses the physical robot for programming. This means that it does not reduce downtime, as much as offline programming.
- Moving the robot to precise coordinates is not as straightforward as with the other methods. This is especially true with some joystick based systems, where there is no way of entering a numerical value. Kinetiq teaching combines these features by allowing for the entering of exact numerical coordinates along with positioning based coordinates.

[8] `https://www.blumenbecker.com/industrial-automation/industrial-robotics/offline-programming`

- Not so good for tasks which are algorithmic in nature. For example, if a robot had to paint a flat surface by moving horizontally along the surface, then move down an inch, move horizontally in the opposite direction, etc. Moving the robot by hand would be arduous and inaccurate for such a task [9].

14.5 ROS: Robot System Operation

Robot Operating System (ROS or ros) is an open-source robotics middleware suite. Although ROS is not an operating system but a collection of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management.

Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor data, control, state, planning, actuator, and other messages. Despite the importance of reactivity and low latency in robot control, ROS itself is not a real-time OS (RTOS). It is possible, however, to integrate ROS with real-time code. The lack of support for real-time systems has been addressed in the creation of ROS 2, a major revision of the ROS API which will take advantage of modern libraries and technologies for core ROS functionality and add support for real-time code and embedded hardware. See figure 14.10.

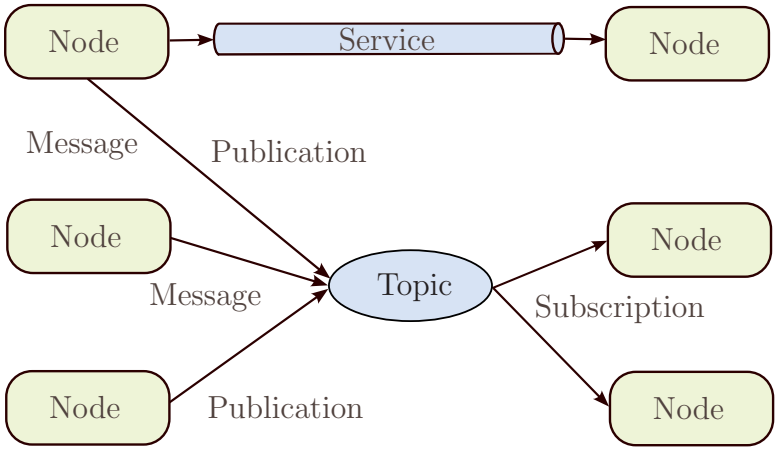


Figure 14.10: ROS

Software in the ROS Ecosystem can be separated into three groups:

- language-and platform-independent tools used for building and distributing ROS-based software.
- ROS client library implementations such as roscpp, rospy, and roslisp;

[9] `https://blog.robotiq.com/what-are-the-different-programming-methods-for-robots`

- packages containing application-related code which uses one or more ROS client libraries.

Both the language-independent tools and the main client libraries (C++, Python, and Lisp) are released under the terms of the BSD license, and as such are open-source software and free for both commercial and research use. The majority of other packages are licensed under a variety of open-source licenses. These other packages implement commonly used functionality and applications such as hardware drivers, robot models, datatypes, planning, perception, simultaneous localization and mapping, simulation tools, and other algorithms.

The main ROS client libraries are geared toward a Unix-like system, primarily because of their dependence on large collections of open-source software dependencies. For these client libraries, Ubuntu Linux is listed as Supported, while other variants such as Fedora Linux, macOS, and Microsoft Windows are designated experimental and are supported by the community. The native Java ROS client library, rosjava, however, does not share these limitations and has enabled ROS-based software to be written for the Android OS. rosjava has also enabled ROS to be integrated into an officially supported MATLAB toolbox which can be used on Linux, macOS, and Microsoft Windows. A JavaScript client library, roslibjs has also been developed which enables integration of software into a ROS system via any standards-compliant web browser[10].

14.6 Finroc

Finroc is a modular C++/Java framework for robot control systems. It has been developed at the Robotics Research Lab of TU Kaiserslautern, Germany since 2008. Frameworks have fundamental impact on software quality of robot control systems as well as development effort. Taking many existing approaches into account, the design of Finroc evolved around the question: What can be done in a framework in order to support or even guarantee relevant quality attributes of robot control systems? It is hardly possible to implement all the identified features at once. Thus, the initial design focused on areas that are hard to add or change later. The main developers used MCA2 before developing Finroc and learned to appreciate many of its qualities. Thus, application style in Finroc is similar.

Prominent features of Finroc are:

- Efficient (zero-copy), lock-free, real-time implementation
- Scales up to thousands of components
- Can be run without an operating system
- Intra-process runtime construction
- Slim and highly modular framework core (see below - or our IEEE IRC 2017 paper for the master plan)
- Separate, native C++11 and Java versions

[10] `https://en.wikipedia.org/wiki/Robot_Operating_System`

14.6.1 Finroc Application Structure and Decomposition in a Nutshell

Similar to many other robotic frameworks (MCA2 in particular), Finroc applications are constructed from interconnected components. Similar to MCA2, the structural elements in Finroc applications are modules (basic components), groups (composite components) and parts (executables/processes).

Modules are the basic application building blocks. Their interfaces are a set of ports. Often, these are data ports: Output ports are used to publish data, whereas Input Ports receive data. Edges are used to connect two ports. Modules with data ports connected by edges form a kind of data flow graph - which can be visualized with the finstruct tool. Edges can connect components running on different systems just as well as components running inside the same process. This allows to easily create distributed systems. Apart from data ports there are also rpc ports: Server ports provide an interface with remote procedure calls that client ports can connect to.

Finroc applications may consist of thousands of components. In order to maintain a clear application structure, modules can be placed in groups that have their own interface. Using groups, an application hierarchy is established. Modules are typically assigned to thread containers. The thread inside the thread container will trigger any periodic update tasks continuously with a certain cycle time. It is, however, also possible to react to asynchronous events.

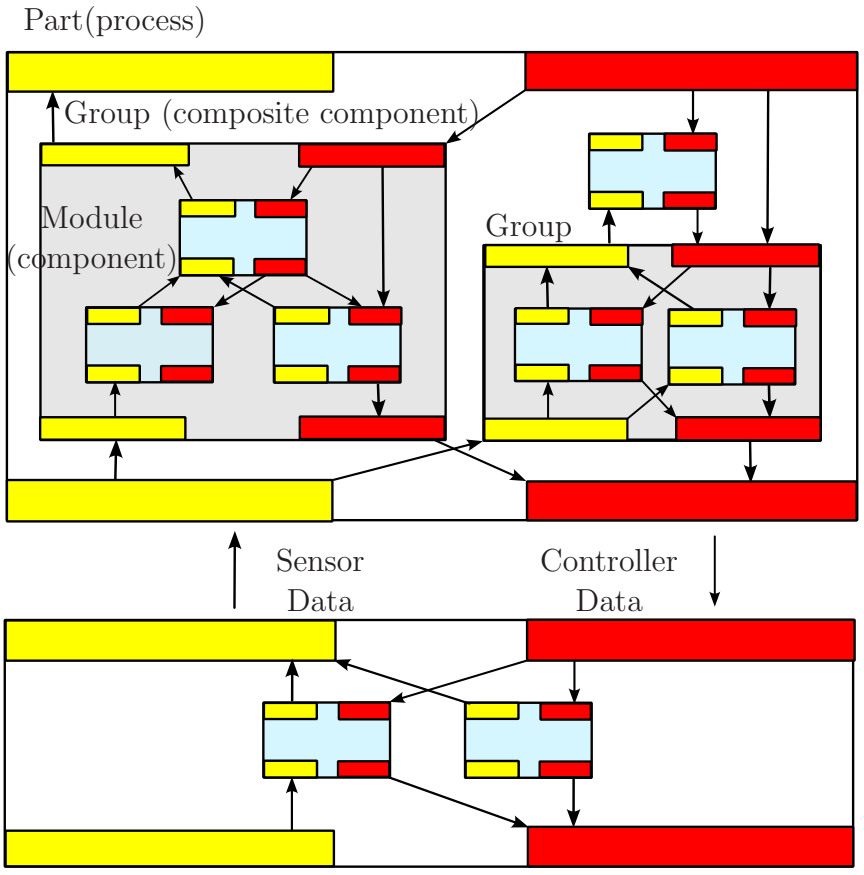


Figure 14.11: Finroc

Similar to the MCA2 framework, Finroc distinguishes between sensor and controller data (see figure 14.11). Sensor data (yellow) flows upwards in application visualization. Controller data (red) flows downwards. Distinguishing between sensor and controller data in this way, supports clear application visualization and structure.

Finroc supports different component types. The SenseControlModule and SenseControlGroup are the respective component types from MCA2, extended by service ports. Apart from that, there are plain modules and groups or e.g. ib2c behaviours. See [Reichardt 17] and [Reichardt 13].

15. Summary and Example in Application

Motivation

After automation has been becoming a standard in production factories it grows more and more also for mobile machines. So, there are already some solutions for agriculture machines at the market. For example, the *John Deere* autonomous guidance systems can be mentioned. This is a system that can control tractors over a field steered by a computer system: To do that GPS is used to measure the position of the vehicle and computers on board of the machine do the controlling. Besides these, the trucks on the roads get an increasing amount of complex assistance systems, too.

There is not such a big step to bring the innovations to machines on construction site. Here the computer technique can be used to support the workers with assistant systems. For example, it is possible to improve the controlling of an excavator arm. Digging is normally a tasks that needs a lot of experience because of the unfamiliar kinematic and the huge amount of degrees of freedom that has to be controlled of such a backhoe. This can be improved by letting solve the calculations of the kinematic done by a computer system. With the results, an assistant system can allow to control the position of the bucket directly by a joystick and the hydraulic cylinders do not have to be controlled individually. Furthermore, a construction side is mostly a well defined place that is separated from other environments. This separation is mostly strict on a construction side. This is a good condition for further automation because the area is well defined and the dangerous working ranges of an autonomic machine can be strictly closed for people. So, further automation can make a construction side less dangerous for humans.

15.1 Bachloe Loader Platform

A *backhoe loader* is with its combination of a front loader, a tractor like vehicle and a backhoe the *Swiss Army Knife* of the construction machines. So, for first steps in automation this is the perfect system. With this work an approach is given for a closed loop control for the movements of the backhoe of such a vehicle: The basis for automation is always the ability to control the machine.

Figure 15.1: Backhoe Loader

Figure 15.1 shows a machine of type John Deere 410J TMC backhoe loader. As the automation of them whole system would be a task that exceeds the range of this work a lot, the approach given here is limited to the rear backhoe of the vehicle, which should be controlled along a given trajectory. To be able to solve this task, the machine which is available for this work is equipped with angle encoders that can measure the main angle positions. Additionally, the hydraulic system which moves the backhoe arm can be controlled via a *Controller Area Network* (CAN) bus. It also includes Laser scaner for rotation and a shovel camera along with a GPS-System.

A good control algorithm can only be given if the system is described mathematically in an adequate way. Therefore, the main part of this work is the modelling of the system. That includes both the kinematic and the hydraulics. With the kinematic the structure of the backhoe mechanic will be described. This will be done mathematically with equations that contains the relations between the angles, length and positions of the different parts. Here problems may occur because of the closed kinematic loops that are given by the way the hydraulic cylinders are connected with the rest of the backhoe parts. In contrast to well known robot arms which are mostly equipped with directly driven joints, the movements in the main joints are driven indirectly by the hydraulic system. This results in a parallel kinematic that causes additional problems, because the arm should not only be brought to some static targets but also it should move with a given velocity. In addition, the relations between the velocities have to be modelled. This includes the relation between the velocity of the *Tool Center Point* (TCP) located on the middle tooth of the bucket and the angle velocities as soon as the velocities of the actuators which are the hydraulic cylinders. Vice versa, the resulting TCP velocity which is induced by the hydraulic movements has to be modelled. The problems that occur because of the parallel mechanism influence this modelling part, too.

The hydraulic movement is controlled by the hydraulic flow into and out of them. This again depends on the opening of connected valves which will be controlled by the CAN messages. Because of that it is necessary to describe also the hydraulic system including the hydraulic cylinders and the valves to get an understanding of the control chain. With this information a control algorithm can be formulated that can be used to move the bucket along a given trajectory which was the original task.

Figure 15.2: Digging a Trench on Real Machine

15.2 The Mechanical System

15.2.1 DH Parameters

To describe the kinematics of the backhoe, the Denavit-Hartenberg Convention has been chosen. This is a common technique to find the necessary parameters that describe the kinematics of a robot completely.

One coordinate system for every joint will be defined. The z-axis of the coordinate system is along the moving axis of the joint and the x-axis is the cross product of the z-axis and the z-axis from the coordinate system of the previous joint. After all, the y-axis will be defined to build a right handed coordinate system.

To transform one coordinate system into the next one, a consistent transformation exist that can be defined with four parameters. A first transformation is a rotation with angle θ_n around the z-axis to bring the x-axis parallel to the x-axis of the next coordinate system. Then, the coordinate system will be translated with the length dn along the z-axis to bring the x-axes together. The next transformation is a translation with length an along the x-axis so that finally the origins are at the same place. In the end, the coordinate system will be rotated with angle α_n around the x-axis to bring the z-axes together. After all, the transformations results in several parameters that describe the kinematics, the so called *DH Parameters*.

To be able to do that the kinematic of the manipulator has to be described first. This is rather difficult because the arm has a parallel kinematic. The main segments are moved from hydraulic cylinders, whose are positioned parallel to this segments. To be able to give proper relations between the two chains the parallel kinematic can be cut. This is also necessary because the drives are included in the second chain and control the angle

variations in the first one. So, the distinction is necessary to be able to give a relation between the deflection of the hydraulic cylinders and the main angle configuration. In the following the first part of the kinematic which follows the main segments will be called *Main Segment Chain* and the other one which is defined along the hydraulic actuators will be called *Actuator Chain*. Both of the chains are independently enough to calculate the pose of the TCP because they are parallel. But here the definition of both is necessary because the actuators are included in the Actuator Chain while the measured angles in the system are parts of the Main Segment Chain.

Main Segment Chain For the identification of the main backhoe configuration it is necessary to define a DH definition for the main parts of the backhoe. With this set of values, it is possible to describe the actual position of all parts of a backhoe. Furthermore, these values are used to calculate the forward and inverse kinematic of which explanations can be found in sections 15.2.2 and 15.2.3. In table 15.1 the parameters that are used to describe the kinematic of the *Main Segment Chain* according the DH convention can be found. It includes the main angles θ_{sw}, θ_{bo}, θ_{cr} and θ_{bu} of the backhoe arm as well as the segment lengths a_{sw}, a_{bo},a_{cr} and a_{bu}. The lengths of the segments are fixed so their values are also given in table 15.1.

	θ	d	a(mm)	α
J_0				
J_{sw}	θ_{sw}	0	a_{sw} = 465	90°
J_{bo}	θ_{bo}	0	a_{bo} = 2829.86	0
J_{cr}	θ_{cr}	0	a_{cr} = 2144.85	0
J_{bu}	θ_{bu}	0	a_{bu} = 945.065	90°

Table 15.1: Main DH Parameter of the backhoe

All of these given values have been measured in the CAD model provided by John Deere. That is also valid for the other values of the lengths and angles that are given for the actuator chain in the next paragraph. So, the parameters in the table without a given value are variable. They describe the actual displacement of the joints. For the backhoe they are θ_{sw}, θ_{bo}, θ_{cr} and θ_{bu}, the angles of the main joints. On the real backhoe loader these angles will be measured from angle encoders, so the values of these parameters are available to the control system. In figure 15.3, the locations of the parameters are shown.

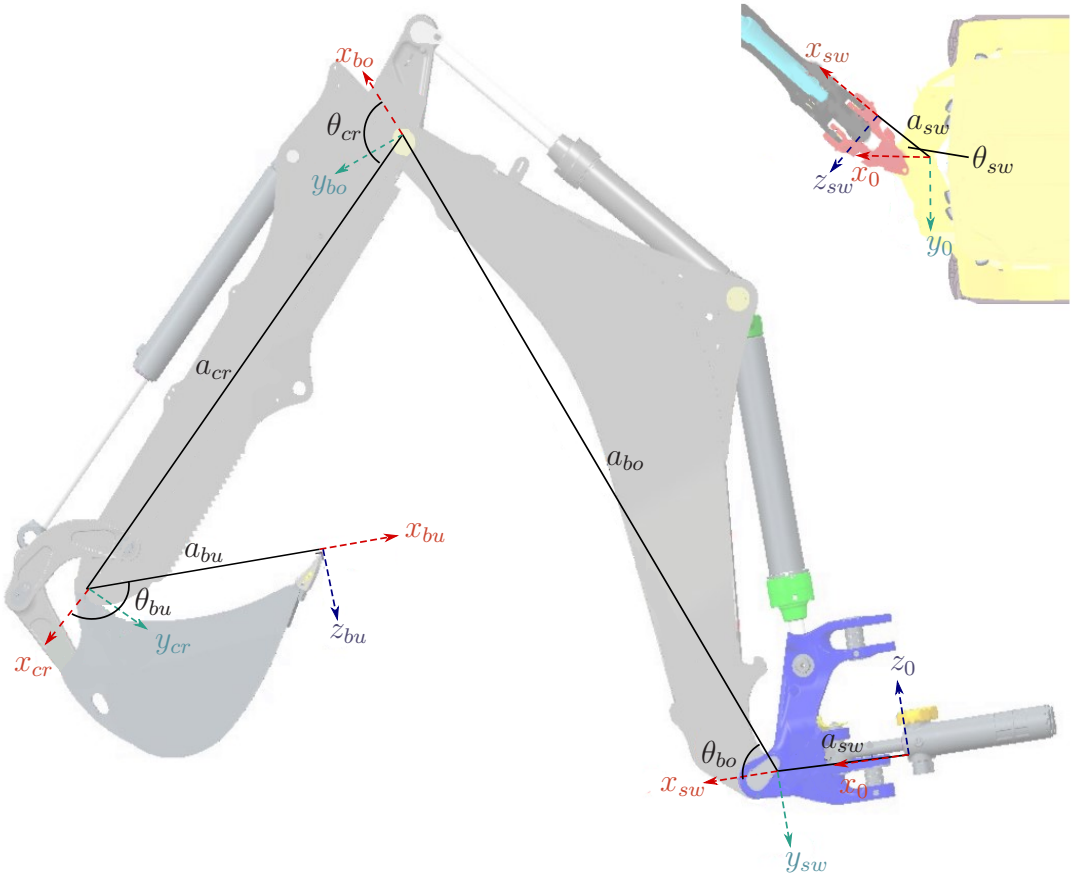


Figure 15.3: Backhoe main DH Parameters

Actuator Chain

To be able to give a description of the other moveable parts, parameter definitions for the remaining joints and links are given, too. They describe mainly the deflection of the hydraulic cylinders and the angular displacement of the revolute joints surrounding them. So, there is a set of parameters for every available hydraulic cylinder which are grouped into boom, crowd and bucket parameters.

Each set describes a part of the *Actuator Chain* and gives a relation between the deflection of a hydraulic cylinder and the position of the appropriated backhoe part. So, there is a transformation from the coordinate system of the previous main joint to the one of the next over the kinematic path that contains the hydraulic cylinder. This is done for every hydraulic cylinder and therefore for every main joint. For example, the set of the boom describes the transformation ${}^{sw}_{bo}T$ from boom to crowd joint over the boom hydraulic cylinder.

From the given transformations it is possible to formulate equations for the relation between the backhoe main angles and the deflection of the hydraulic cylinders. In table 15.2 the parameters for the boom are given and the parameters for the crowd can be found in table 15.3.
The assignments for the DH parameters are given in figures 15.4 and 15.5.

Joint	θ	d(mm)	a(mm)	α
J_{sw}				
J_{bo2}	$\theta_{bo2} = -121.422°$	0	$a_{bo2} = 472.283$	0
J_{bo3}	θ_{bo3}	0	0	90°
J_{bo4}	0	$d_{bo4} = 1372.5 + (0; 800)$	0	−90°
J_{bo5}	θ_{bo5}	0	$a_{bo5} = 1478.3$	0
J_{bo}	$\delta_{bo} = -35.698°$	0	0	0

Table 15.2: DH Parameter of boom hydraulics

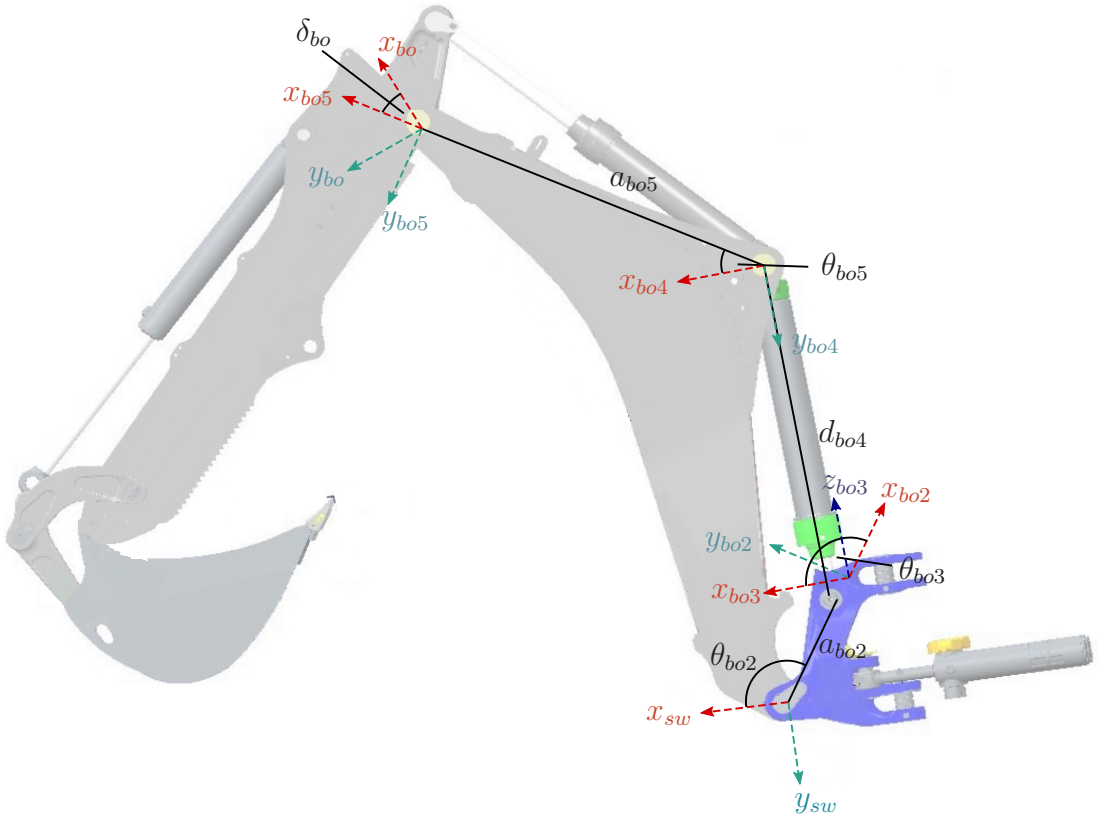


Figure 15.4: Boom Hydraulic DH Parameters

Joint	θ	d(mm)	a(mm)	α
J_{bo}				
J_{cr2}	$\theta_{cr2} = -144.302°$	0	$a_{cr2} = 1478.3$	0
J_{cr3}	θ_{cr3}	0	0	90°
J_{cr4}	0	$d_{cr4} = 1122.7 + (0; 700)$	0	−90°
J_{cr5}	θ_{cr5}	0	$a_{cr5} = 2530.25$	0
J_{cr}	$\delta_{cr} = -4.3423°$	0	0	0

Table 15.3: DH Parameter of crowd hydraulics

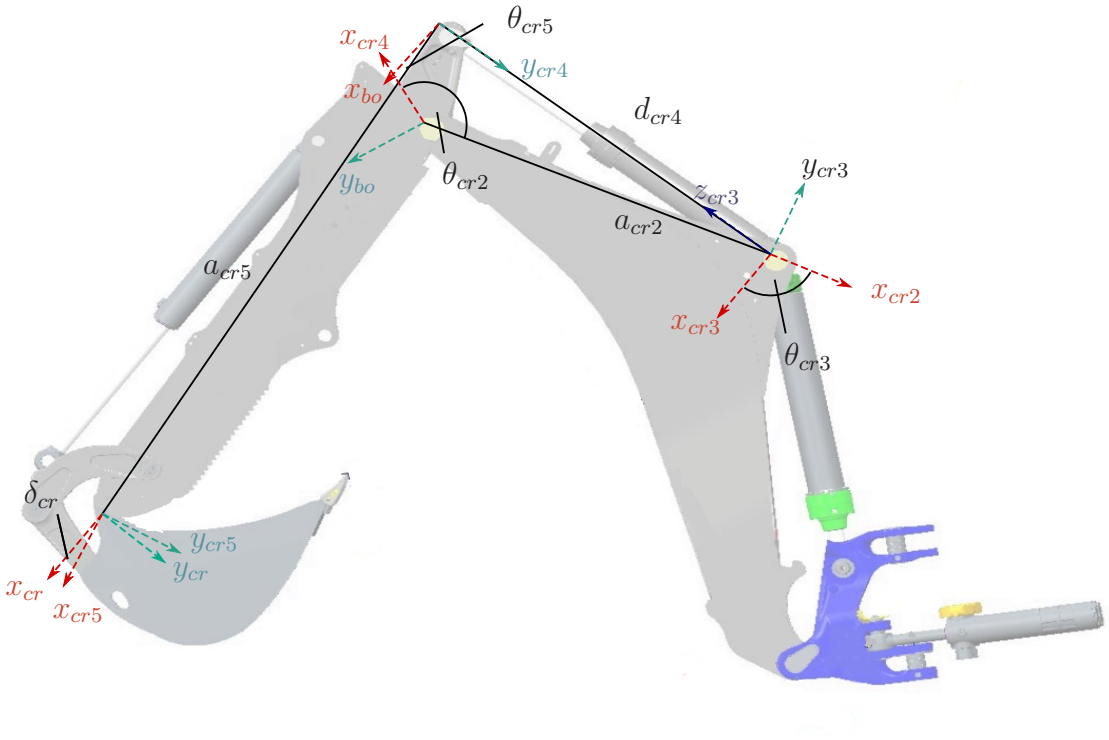


Figure 15.5: Crowd Hydraulic DH Parameters

Moreover, here it can be seen that the structure of the DH Parameters for the boom and crowd hydraulic are the same, only the values differ. Owing to that fact, it is possible to adapt calculations from one of these parts to the other. Only the values have to be changed while the equations can be reused.

As shown in the images above, the bucket is not driven by the hydraulic cylinder directly. Instead an additional linkage to move the bucket exists. Due to that fact a further set of parameters is necessary. So, two of them are given here. While the first one describes the kinematic of the bucket according to the other parts, the second one describes an additional kinematic chain for the linkage that also depends on the deflection of the hydraulic cylinder. In the end, the bucket movement depends on the combination of these two chains. Only with the presence of both of them the bucket position becomes explicit. With the parameters, it is possible to give equations for the relation between the deflection of the hydraulic cylinder and the main bucket joint angle θ_{bu}. The parameters that are used in the kinematic chains of the bucket are described in table 15.4 and 15.5. The figures 15.6 and 15.7 show the relation between the parameters and the real backhoe parts.

Joint	θ	d(mm)	a(mm)	α
J_{cr}				
J_{bu2}	$\theta_{bu2} = -169.791°$	0	$a_{bu2} = 1864.53$	0
J_{bu3}	θ_{bu3}	0	0	90°
J_{bu4}	0	$d_{bu4} = 1145.5 + (0; 800)$	0	−90°
J_{bu5}	θ_{bu5}	0	$a_{bu5} = 476.56$	0
J_{bu6}	θ_{bu6}	0	$a_{bu6} = 1090.59$	0
J_{bu}	$\delta_{bu} = -22.1181°$	0	0	0

Table 15.4: DH parameter for bucket hydraulics

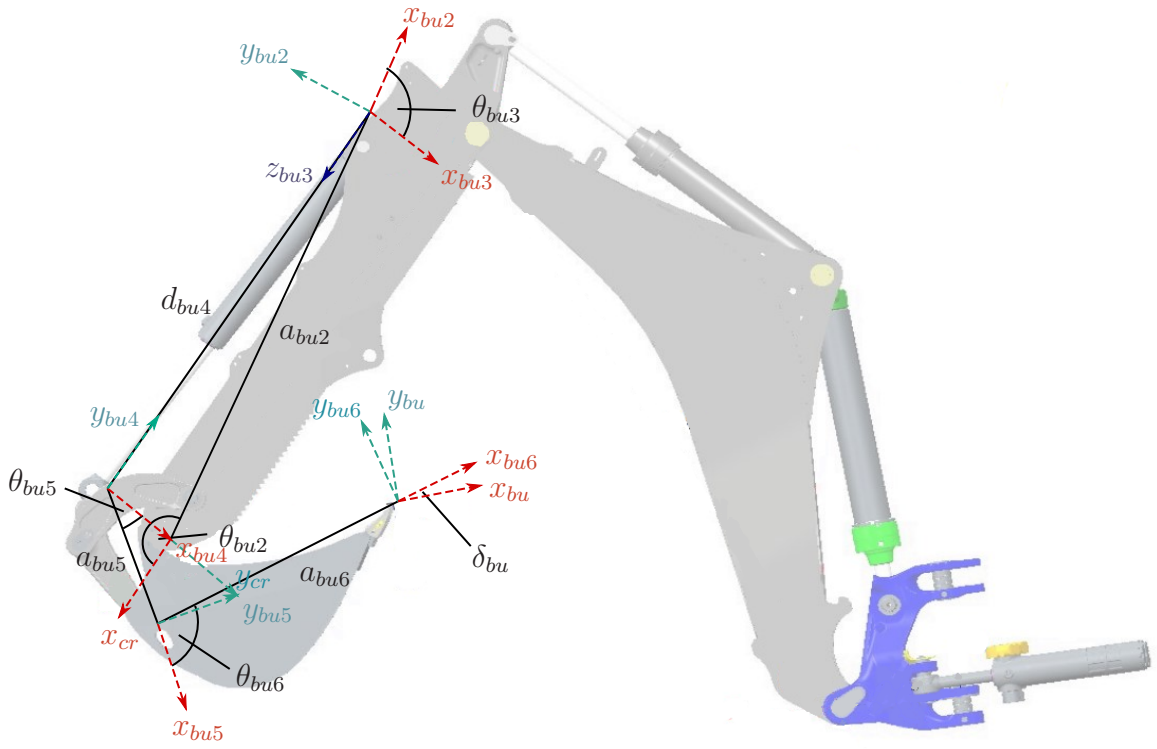


Figure 15.6: Bucket hydraulic DH Parameters

Joint	θ	d(mm)	a(mm)	α
J_{cr}				
$J_{bu2'}$	$\theta_{bu2'} = -178.478°$	0	$a_{bu2'} = 237.472$	0
$J_{bu3'}$	$\theta_{bu3'}$	0	$a_{bu3'} = 480$	0
$J_{bu4'}$	$\theta_{bu4'}$	0	$a_{bu4'} = 380$	0
$J_{bu5'}$	$\theta_{bu5'}$	0	$a_{bu5'} = 476.56$	0
J_{bu}	$\delta_{bu'} = -22.1181°$	0	0	0

Table 15.5: DH parameter for the bucket gear

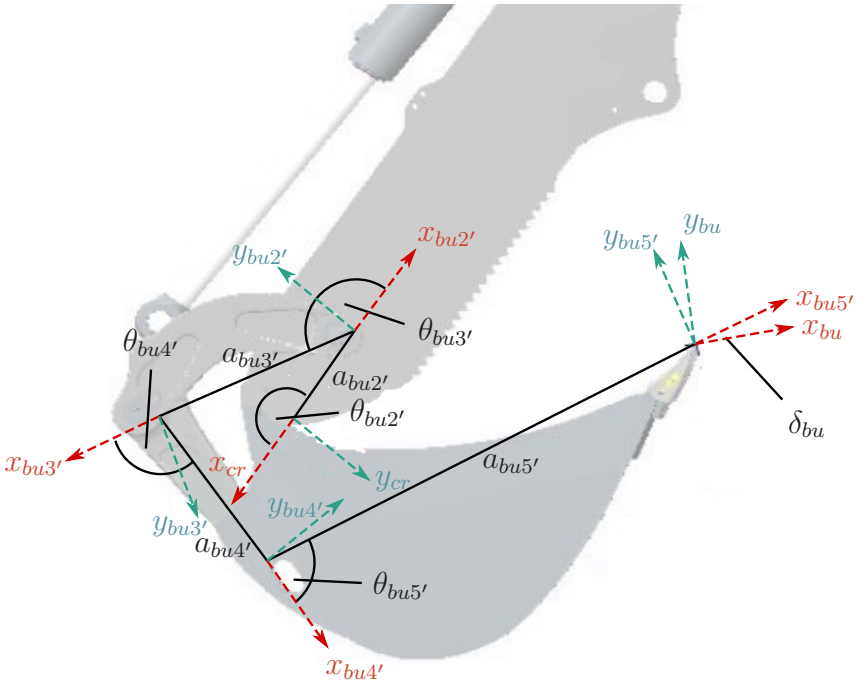


Figure 15.7: Bucket linkage DH Parameters

15.2.2 Forward Kinematic

On the basis of the DH Parameters it is possible to calculate the forward kinematic, which results in a definition of an equation for the pose of the $TCP : P_{bu} = (x, y, z, \phi, \theta, \psi)^T$. This pose is defined at the outer end of the middle tooth of the bucket. The vector $(x, y, z)^T$ describes the translation to this point from the base coordinate system. Furthermore, the orientation of the TCP is given by (ϕ, θ, ψ). Here the roll angle ϕ describes the rotation around the x-axis, the yaw angle θ the rotation around the y-axis and the pitch angle ψ the rotation around the z-axis of the base coordinate system.

A first transformation ${}^{0}_{sw}A$ describes the transition between the base coordinate system which will be spanned by the axes x_0, y_0 and z_0 as shown in figure 15.3. Furthermore, it defines the pose of the swing described by the swing coordinate system spanned by x_{sw},y_{sw} and z_{sw}.

$$ {}^{0}_{sw}A = \begin{bmatrix} \cos(\theta_{sw}) & 0 & -\sin(\theta_{sw}) & a_{sw}.\cos(\theta_{sw}) \\ \sin(\theta_{sw}) & 0 & \cos(\theta_{sw}) & a_{sw}.\sin(\theta_{sw}) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15.1} $$

To change the basis from swing to boom coordinate system, the transformation matrix sw ${}^{sw}_{bo}A$ is applied. It consists of a rotation R_{zsw} around the z_{sw} axis with angel θ_{bo} and a following translation T_{xbo} along the new x-axis with length a_{bo}.

$$
\begin{aligned}
{}^{sw}_{bo}A &= R_{zsw}(\theta_{bo}).T_{xbo}(a_{bo}) \\
&= \begin{bmatrix} \cos(\theta_{bo}) & -\sin(\theta_{bo}) & 0 & a_{bo}.\cos(\theta_{bo}) \\ \sin(\theta_{bo}) & \cos(\theta_{bo}) & 0 & a_{bo}.\sin(\theta_{bo}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{15.2}
$$

The next transformation ${}^{bo}_{cr}A$ transfers the boom coordinate system (x_{bo}, y_{bo}, z_{bo}) to the crowd (x_{cr}, y_{cr}, z_{cr}).

$$
\begin{aligned}
{}^{bo}_{cr}A &= R_{zbo}(\theta_{cr}).T_{xcr}(a_{cr}) \\
&= \begin{bmatrix} \cos(\theta_{cr}) & -\sin(\theta_{cr}) & 0 & a_{cr}.\cos(\theta_{cr}) \\ \sin(\theta_{cr}) & \cos(\theta_{cr}) & 0 & a_{cr}.\sin(\theta_{cr}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{15.3}
$$

For the final transformation to the bucket coordinate system (x_{bu}, y_{bu}, z_{bu}), the transformation given by the matrix ${}^{cr}_{bu}A$ can be used. This is a rotation R_{zcr} around the z_{cr} axis with θ_{bu}, a translation T_{xbu} along x_{bu} axis about a_{bu} and finally a rotation of 90° around x_{bu} to bring it into the same axis arrangement as the base coordinate system.

$$
\begin{aligned}
{}^{cr}_{bu}A &= R_{z3}(\theta_{bu}).T_{xTCP}(a_{bu}).R_{xTCP}(90°) \\
&= \begin{bmatrix} \cos(\theta_{bu}) & 0 & -\sin(\theta_{bu}) & a_{bu}.\cos(\theta_{bu}) \\ \sin(\theta_{bu}) & 0 & \cos(\theta_{bu}) & a_{bu}.\sin(\theta_{bu}) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{15.4}
$$

The multiplication of all these transformation matrices results in a unique matrix ${}^{0}_{bu}A$ that represents the transition from the base coordinate system to the TCP at the bucket. It contains the configuration of the main backhoe angles and the link length of the main parts.

$$
\begin{aligned}
{}^{0}_{bu}A &= {}^{0}_{sw}A.{}^{sw}_{ou}A.{}^{bo}_{cr}A.{}^{cr}_{bu}A \\
&= \begin{bmatrix} C_{bo.cr.bu}.C_{sw} & -S_{sw} & S_{bo.cr.bu}.C_{sw} & C_{sw}.(a_{sw} + a_{cr}.C_{bo.cr}) + a_{bo}.C_{bo.cr.bu} \\ C_{bo.cr.bu}.S_{sw} & C_{sw} & S_{bo.cr.bu}.S_{sw} & S_{sw}.(a_{sw} + a_{cr}.C_{bo.cr}) + a_{bo}.C_{bo.cr.bu} \\ S_{bo.cr.bu} & 0 & S_{bo.cr.bu}.C_{sw} & -a_{cr}.S_{bo.cr} - a_{bo}.S_{bo.cr.bu} \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{15.5}
$$

such that:
$S_{bo.cr.bu} = \sin(\theta_{bo} + \theta_{cr} + \theta_{bu})$ and $C_{bo.cr.bu} = \cos(\theta_{bo} + \theta_{cr} + \theta_{bu})$.
It is also possible to give a matrix ${}^{0}_{bu}T$ for the direct transformation from the base coordinate system to the TCP. This is the result from a translation $(x, y, z)^T$ to this point and a rotation which is defined from the roll angle ϕ, the pitch angle θ and the yaw angle ψ.

In this definition of the orientation, ψ describes the rotation around the x-axis of the basecoordinate system, θ the rotation around y-axis and ψ around the z-axis.

$$
{}^{0}_{bu}T = \begin{bmatrix} C\theta.C\psi & C\psi.S\phi.S\theta - C\phi.S\psi & S\phi.S\psi + C\phi.C\psi.S\theta & x \\ C\theta.S\psi & C\phi.C\psi + S\phi.S\theta.S\psi & C\phi.S\theta.S\psi - C\psi.S\phi & y \\ -S\theta & C\theta.S\phi & C\theta.C\phi & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15.6}
$$

Because there is no joint to buckle, the bucket sideways around the global x-axis, the roll angle ϕ, is always zero. By using this to simplify ${}^{0}_{bu}T$ it results in an easier matrix because $\sin(0) = 0$ and $\cos(0) = 1$.

$$
{}^{0}_{bu}T = \begin{bmatrix} \cos(\theta)\cos(\psi) & -\sin(\psi) & \cos(\psi)\sin(\theta) & x \\ \cos(\theta)\sin(\psi) & \cos(\psi)+ & \sin(\theta)\sin(\psi) & y \\ -\sin(\theta) & 0 & \cos(\theta) & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15.7}
$$

To determine the forward kinematic the direct transformation ${}^{0}_{bu}T$ and the kinematic transformation ${}^{0}_{bu}A$ will be equalized. That is possible because the both describe the same pose.

$$
{}^{0}_{bu}T = {}^{0}_{bu}A \tag{15.8}
$$

The components of these two matrices can be compared directly. So, the equations to get the TCP pose $(x, y, z, \phi, \theta, \psi)^T$ can be read off.

$$
\begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \cos(\theta_{sw}) + (a_{sw} + a_{cr}.\cos(\theta_{bo} + \theta_{cr}) + a_{bu}.\cos(\theta_{bo} + \theta_{cr} + \theta_{bu})) \\ \sin(\theta_{sw}) + (a_{sw} + a_{cr}.\cos(\theta_{bo} + \theta_{cr}) + a_{bu}.\cos(\theta_{bo} + \theta_{cr} + \theta_{bu})) \\ -a_{cr}.\sin(\theta_{bo} + \theta_{cr}) - a_{bo}.\sin(\theta_{bo}) - a_{bu}.\sin(\theta_{cr} + \theta_{bu}) \\ 0 \\ \theta_{bo} + \theta_{cr} + \theta_{bu} \\ \theta_{sw} \end{bmatrix} \tag{15.9}
$$

With these equations, a relation between the backhoe main DH parameters and the TCP pose in the base coordinate system is defined. The forward kinematic problem is solved.

15.2.3 Inverse Kinematic

The inverse kinematic gives a solution for the problem that the pose of the *bu* is given and the main angles of the backhoe are not. With the equations described here, it is possible to calculate the backhoe angles θ_{sw}, θ_{bo}, θ_{cr} and θ_{bu} from the pose $P_{bu} = (x, y, z, \phi, \theta, \psi)^T$.

This is useful if a target position is given and the angles have to be set to bring the bu in this pose. To solve this problem, equations for θ_{sw}, θ_{bo}, θ_{cr} and θ_{bu} have to be found from the equalization of ${}^{0}_{bu}T$ and ${}^{0}_{bu}A$.
From the forward kinematic the configuration for the swing angle θ_{sw} is directly given.

$$\theta_{sw} = \psi \tag{15.10}$$

All the other angles occur as movements in one plane. Because of that fact, the equations might be more easy if ${}^{0}_{bu}T$ and ${}^{0}_{bu}A$ will be transformed into the swing coordinate system. This can be done by multiplying both with ${}^{0}_{sw}A^{-1}$

$${}^{0}_{sw}A^{-1}.{}^{0}_{bu}T = {}^{0}_{sw}A^{-1}.{}^{0}_{bu}A \tag{15.11}$$

$$\begin{bmatrix} \cos(\psi-\theta_{sw}).\cos(\theta) & -\sin(\psi-\theta_{sw}) & \cos(\psi-\theta_{sw}).\sin(\theta) & x.\cos(\theta_{sw}) - a_{sw} + y.\sin(\theta_{sw}) \\ \sin(\theta) & 0 & \cos(\theta) & -z \\ \sin(\psi-\theta_{sw}).\cos(\theta) & -\cos(\psi-\theta_{sw}) & \sin(\psi-\theta_{sw}).\sin(\theta) & y.\cos(\theta_{sw}) - x.\sin(\theta_{sw}) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} C_{bo.cr.bu} & 0 & S_{bo.cr.bu} & a_{cr}.C_{bo.cr} + a_{bo}.C_{bo} + a_{bu}.C_{bo.cr.bu} \\ S_{bo.cr.bu} & 0 & -C_{bo.cr.bu} & a_{cr}.S_{bo.cr} + a_{bo}.S_{bo} + a_{bu}.S_{bo.cr.bu} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15.12}$$

From the forward kinematics the facts $\psi = \theta_{sw}$ and $\theta = \theta_{bo} + \theta_{cr} + \theta_{bu}$ are known. So this can be inserted into the matrices:

$$\begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & x.\cos(\psi) - a_{sw} + y.\sin(\psi) \\ \sin(\theta) & 0 & -\cos(\theta) & -z \\ 0 & 1 & 0 & y.\cos(\psi) - x.\sin(\psi) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & a_{cr}.\cos(\theta_{bo}+\theta_{cr}) + a_{bo}.\cos(\theta_{bo}) + a_{bu}\cos(\theta) \\ \sin(\theta) & 0 & -\cos(\theta) & a_{cr}.\sin(\theta_{bo}+\theta_{cr}) + a_{bo}.\sin(\theta_{bo}) + a_{bu}.\sin(\theta) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15.13}$$

In addition, a direct transformation ${}^{sw}_{bu}T$ from the swing coordinate system to the bu can be formulated. This consists of the transformation from the origin of that coordinate system to the $bu({}_{sw}x, {}_{sw}y, {}_{sw}z)^T$ and the orientation of the bucket in this frame which is given by θ because the whole movement will be executed in the $x-y$ plane of the swing coordinate system:

$${}^{sw}_{bu}T = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & {}_{sw}x \\ \sin(\theta) & 0 & -\cos(\theta) & {}_{sw}y \\ 0 & 1 & 0 & {}_{sw}z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15.14}$$

This transformation describes the same as the one given by equation 15.13 So it can be set equal to them. Because the rotation is already the same, it can be reduced to the translation $({}_{sw}x, {}_{sw}y, {}_{sw}z)^T$.

$$\begin{bmatrix} x.\cos(\psi) - a_{sw} + y.sin(\psi) \\ -z \\ y.\cos(\psi) - x.\sin(\psi) \end{bmatrix} = \begin{bmatrix} {}_{sw}x \\ {}_{sw}y \\ {}_{sw}z \end{bmatrix} = \begin{bmatrix} a_{cr}.\cos(\theta_{bo} + \theta_{cr}) + a_{bo}.\cos(\theta_{bo}) + a_{bu}.\cos(\theta) \\ a_{cr}.\sin(\theta_{bo} + \theta_{cr}) + a_{bo}.\sin(\theta_{bo}) + a_{bu}.\sin(\theta) \\ 0 \end{bmatrix} \tag{15.15}$$

It can be seen that $({}_{sw}x, {}_{sw}y, {}_{sw}z)^T$ can be directly calculated from a given pose and the available static parameters. To calculate the main angles, it is necessary to solve the right side of equation 15.15. The calculation in lengthy and is ommited.

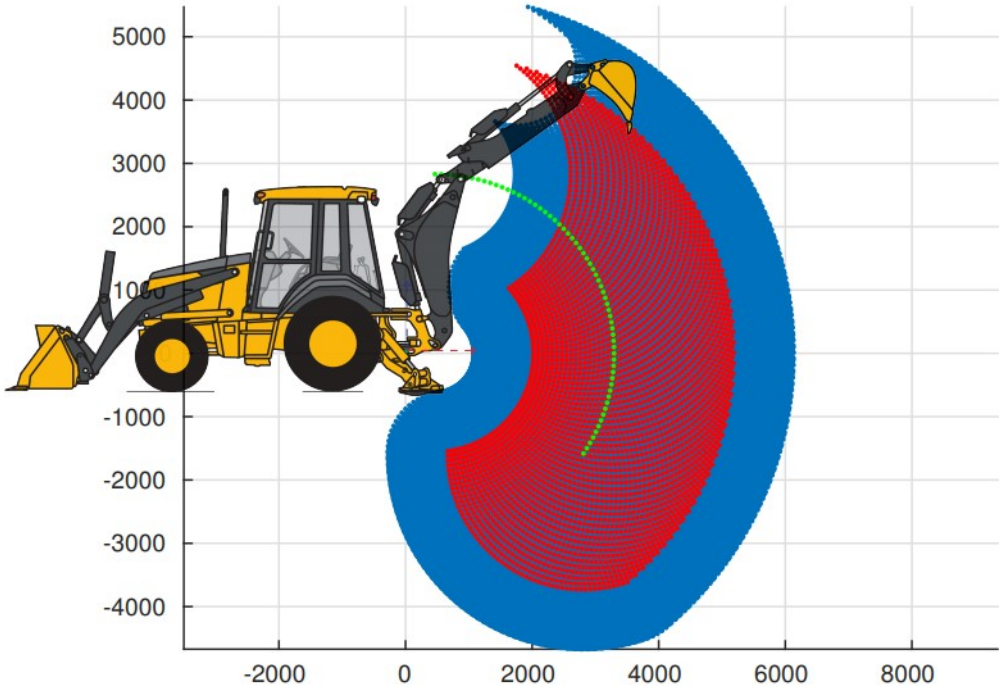


Figure 15.8: Backhoe arm workspace.(green=boom position, red=crowd position, blue=TCP position).

15.3 Control Architecture

For building up a software system for controlling a mobile robot it is necessary to use a propper architecture as a suitable skeleton for including the used algorithms. It organizes the different parts of the system and manages the communication and dataflows between the modules. To adress the excavation functions some additional extension are added to implement perception and control inside of this domain. Furthermore a planning and acting layer will be added which can use the functions of all the domains and include them into its task execution strategies. Which gives the possibilty of providing a sensor

set configuration as well as drive drain configuration of a particular robot and all of the modules will adapt to the given properties. The functionality is organized in different semantic groups, also introduced as layers. In the following a short introduction into this layers is given.

Hardware Interface

Within the hardware interface layer implementations are given to give the control approach access to the hardware. It contains driver modules and communication interfaces exchanging data with the different devices used by the system. This can include interchanging vehicle data using a bus system as Controller Area Network (CAN) or reading images from attached camera or other sensor systems. The implementation of this layer is highly dependent on the actual hardware used by the system and has to be adapted accordingly. If simulation is used for development usually this is the only part which should be exchanged for switching between real machine and simulated one. For the connection to the upper layers of the architecture the hardware interface is designed for providing a standardized interface to different kind of vehicles and sensor sets.

Perception

By the perception group the actual sensor processing will be implemented. It uses the data from the sensor system attached to the vehicle to generate a model of the current state of the environment and the machine. A main part of this layer is the localization of the vehicle in the world. For this sensor fusion methods are used to create a feasible representation for the position and orientation in the world using different localization sensors as *Inertial Measurement Unit* (IMU) or *Global Navigation Satellite System* (GNSS). Additionally, environment models for different aspects are generated and provided as virtual sensor data. This could be for example a representation of recognized obstacles or blocked areas in the surrounding. Also this is done by using fused sensor data out of different sources as laser scanners and camera systems. All of the fusion in the perception layer is done based on the quality of the provided data evaluated according its measurement accuracy.

Knowledge

From the knowledge group external sources of environmental information are provided to the software system. One of the main properties distinguish the knowledge data from the one determined by perception is the static charactre of the data. An example for this are global maps fetched from *OpenStreetMap* containing information about streets, buildings, altitudes and other landmarks.

Mapping

In the mapping layer, there are different kinds of maps generated based on the environment models generated by the perception. Each of theses maps can consider another aspect of the state of the environment. For example they can contain information about the slope or the heights of the appropriate spatial positions. Additionally, traversibility and obstacle maps are generated which can be used for the planning and control of the driving tasks. Furthermore they are usefull for the planning of the mounting positions of the excavator. In the mapping layer even the static information from the knowledge block and the perception are combined for creating dynamically adapted maps of the environment inside of the global context. It is the connection interface between the sensing and knowledge part and the control system.

Navigator

By the functionality modules inside of the navigator layer a planning will be created for the driving tasks. There the environment representations created of the mapping and knowldge part is used to create navigation plans. As a result of the naviagtion planning sufficient pathes will be created to reach the given target positions. Additionally, in this layer the actual driving activity can be chosen and the navigation will be done accordingly. This could be following an object or reaching a given target position for example. According to the chosen driving action and the given target configuration, the navigator generates the trajectory and target pose settings needed by the pilot.

Pilot

With the pilot the actual driving functions are implemented. In this layer the vehicle will be controlled according to the target parameters given by the navigator. One of the main task of the pilot is to adapt the driving according the present environment conditions. This includes obstacle avoiding and adjustments according the conditions of the driven path. For this the data from the on-board sensor system is used together with and reactive behavior based vehicle control mechanism. By the pilot a generalized control vector will be generated containing velocity and steering target configurations.

Low-Level

In the low-level implementation the control vector given by the pilot will be continue processed. This includes additional safety layers preventing the vehicle from performing movements with an high risk. The control commands are reactive adjusted according to sensor measurements. For example the vehicle can be stopped to prevent collisions from here. See figure 15.9

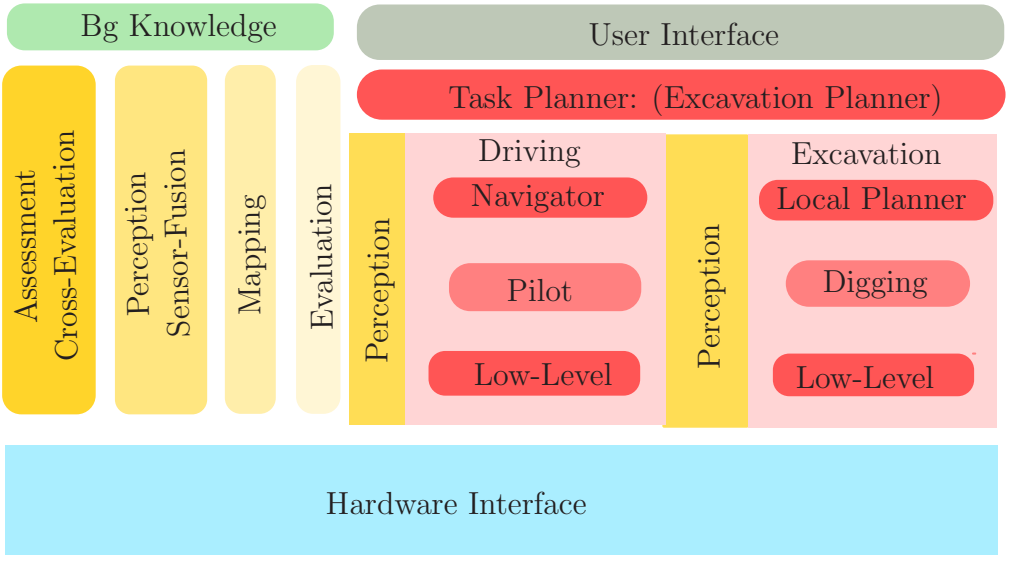


Figure 15.9: Reaction Architecutre with extension for the excavation domains

15.3.1 Maps

For enabeling the software system to deal with excavation tasks descriptions of the environment and the work to be done has to be added. This will be done by using different

maps to store height information about the environment. Usually a description of the initial shape of the landscape will be available based on public data or initial measurements.These static maps will be stored in the knowledge section of the architecutre. Additionally, while the digging process the environment will be changed and it is necessary to stay in track with this process. Accordingly, dynamic adapted maps are introduced to storing the current progress of the digging process.

Knowledge: Static Maps

To give the autonomous excavation system an initial idea from the environment it is possible to use previous sampled data. For the excavation planning the most important information is the surface height in the area where the digging work should be done, the working area. It is necessary that for each location inside of the working area an height value is available to do a propper planning. This is reached by using an elevation grid map providing the information which could be fileld by using information from several sources. For example the heights could be previous measured by a drone flying over the area or a land surveyor could create measurements in the classical way. Additionally, it is also possible to use publicly available data as they are for example provided from some land registry offices or online map services as Open Street Map. Usually each of these data source will provide the information in a different format which has to be converted into the elevation grid map as it is used by the present approach. This grid map will divide the working area in equal sized squares identifiable uniquely with their center position given as two-dimensional coordinate in a global reference frame. To convert the data from the given representation for each of the grid cells an height value has to be found from the appropriate data source. This could be done easily from a given three dimensional representation as it will be provided by scans from drones for example. If external mapping data provides height information this can also be converted to the gird map by using interpolation techinques. The created elevation map will be stored in the knowledge part and offered to the other modules of the software system.

Mapping: Dynamic adapted map

For tracking the excavation process a dynamic adapted height map will be needed. It should be a representation of the current surface height of the working area and should be updated according to the changes the digging process will do in the environment. Similar to the static elevation map of the knowledge layer a grid aspect map is used here storing a height value for each of its spatial squared areas. To stay in track with the changes it is necessary to recognize them by using the on-board capabilities of the technical system used. This could be using a set of sensors observing the working area as laser scanners or cameras. But it is also possible to just track the digging work, especially the movements of the bucket through the soil. It is even possible to combine both of the solutions. For this grid aspect maps are used together with operations to combine them creating a dynamic adapted representation of the working area. For creating the map all of the used positions are given in a global coordinate frame with a defined center point inside of the area of excavation. Further algorithms inside of technical system can use the continously updated current surface map for their calculation as a representation of the current surface. Therefore it will be provided by the global mapping part of the software architecture.

15.3.2 Excavation Domain: Perception and Control

As in the initial design of REACTiON only driving funtionallities are adressed, it is necessary to do some extensions for executing excavation actions. This is done by adding a new domain block which contains the digging functions. Here the local digging is addressed, accordingly the functionality implemented in this part of the software stack is responsible for the digging process on the local mounting positions where the excavator itself stay fixed at a position. Within the digging domain some additional perception requirements are needed with a stronger focus on the working area and the excavation tools. Accordingly, the excavation work inside of the local working area of the machine should be monitored as well as the current state of the bucket filling to stay on track with the task. This should be done in an additional domain specific perception layer inside of the semantic group of the excavation domain. Addtionally, on the control level additional functions are added for handling excavation tasks. So in the excavation domain a toolset in terms of implementation of digging actions is provided. They use the excavation tools of the machine to fulfill different subtasks insided of the excavation domain. Besides the higher level work planner additional local digging planners are also added here for determining digging parameters as start positions for the next iteration. As with the addtional digging tools provided by the machine for sure even additional hardware abstractions are necessary managing the access to the excavator arm. All of this is organized in sublayers responsible for the planning and control functions of the digging domain.

Perception

For the excavation domain a specialized additional perception layer is introduced. It is used as a semantic group for holding digging specific environment and machine state perception functionalities. This will include the surveillance of the excavation area and monitoring the progress of the excavation task. For the recognition of the current landscape shape usually environment sensors as camera systems or laser scanners are used. As an excavator uses its arm to make changes in this environment often the backhoe occlude parts of the observed area depending on the actual sensor position. To generate a proper representation of the working area surface these occlusions has to be considered while the perception process.

Another specific task for the perception will be the monitoring of the bucket filling determining measurements of the current soil volume inside. With these the modeling of the task progress can be enhanced additional this information could be used for monitoring loading processes. For example by aggregating the bucket charges the payload on a truck can be determined and the digging process could be adapted accordingly.

Besides the environment also the state of the machine can be observed. Hereby, the current lifting height as well as the orientation in space will be determined and used for a control system. Furthermore detectors for recognizing machine displacements introduced by the digging progress can be included. Such displacements will happen for example if the soil is quite stiff and the bucket can not easily go into it. With detecting such situations the digging control can react to these events. In general better possibilities for adaptation inside of the control system will be enabled with better perception methods.

Control

In the control block of the excavation domain, functionality is organized to control the actual digging actions using the excavation hardware. It is addtionally organized in

sublayers for the different aspects insided of the control hierarchy. On top level there is the local digging planner determining the parameters for the hierarchy level below. This is the actual control layer for the digging actions, it provides implementation for the different actions used insided of the local digging cycle. It generates general control commands for the excavation hardware as they can be used by the low-level hardware access layer on the lowest level of the control hierarchy.

Local Planning

Even on the local digging level additional planning of the excavation action parameters is needed. This is done in the local planning layer of the excavation domain. Here the parameters are determined needed for the local digging cycle. Accordingly, for each of the iterations of the local cycle a starting position to start the trenching from will be calculated from a set of rating functions. These will give a rating for each possible position inside of the local working area regarding its suitability to start the next digging cylce from. Furthermore a maximum length as well as a maximum depth are figured out according to the instructed task which are used to limit the bucket movements into the target pit. Each of these functions will be calculated continously to being highly adaptive and permiting the digging process to react to the unpredictable conditions of the soil. In the present approach the digging movements are not controlled on a very strict predifined path. Accordingly a prediction of the complete range of each digging iteration is not given and a new calculation of the best starting position is needed for every repetition. With the parameters determined by the planning functions the actual digging activities can be configured. They are implemented inside of the digging layer, explained in the following.

Digging

For the actual excavation actions the digging application layer is introduced. Inside of this semantic group a set of functions controlling the digging operation is implemented. They are organized as a control hierarchy implementing the different parts of the digging operations based on reusable pritmitives. These action networks can be activated and configured according the requirements of the current digging task and its present state. With this functionality is implemented for creating trenches with a given configuration. It uses basic primitives designed according the possibilities given by the general kinematic configuration of an excvator arms and the requirements of the digging functions. This leads into a set of fundamental operation implementations reusable in the different bigger action units of the excavation work. With these the implementation of the phases of the local digging cycle are done. Together they build up a highly adaptable reactive control network generating a target velociy vector for the bucket movement. Additional to the basic controls there are network parts implemented used for adaptation. They are responsible to prevent the excavation actions from bursting the pit borders while the digging phases and also from hitting obstacles in the phases of transfering the empty or filled bucket to its target positions. In this hierarchy level of the control software the calculation of the control value is as far as possible independent of the actual detailed kinematic configuration of the excavator leading into the possibility of using the same approach with different kinds of machines even in various size categories. Within the further low-level layers the velocity control vector has to be fragmented and transformed into the actual commands for the hardware.

Low-Level

In the low level control layer of the excavation domain the given velocity vector determined by the application layer will be translated into the actual command values for the hydraulic system. Using them the necessary hydraulic velocities can be calculated to generate the bucket movement as requested by the digging functions. This is highly dependent on the actual kinematic configuration as provided by the present machine. Based on the hardware interface available further transformations has to be done to generate the necessary control commands. For the excavators used for the experiments in the scope of this works joystick commands are generated replacing the input humans would do on a regular machine. To use the approach on different machines mainly the implementation of the low level layer together with the actual hardware abstraction layer has to be exchanged.

15.4 Common Goals

One of the most popular machines on construction sites is the excavator due to its wide applications. They can be used for different excavation tasks, such as loading piles to trucks, digging trenches or whole excavation pits, and transportation tasks. Some of the jobs are pretty repetitive and intensive in time as creating large excavation pits. Besides the usage for construction work, excavators are operating in mining applications, too. Here, they are mainly used to load blasted rock materials from piles to trucks or belt conveyors. In contrast to wheel loaders, excavators have some advantages for these loading tasks as they have substantial breakout forces. Together with the typical job properties, this large field of application will turn excavators into good candidates for applying automation functionality. Furthermore, there are some applications in which human drivers should not be on the machine at all. This is always the case if it becomes dangerous, as it will be in hazardous environments like nuclear power plants or stiff mountainsides. In the following years, more and more nuclear power plants have to be broken down and also the waste of them has to be managed. Additionally, some natural resources will become rarer, and they will be searched and won even in more dangerous mining scenarios, for example, in high mountain areas. For these tasks, teleoperated or even fully autonomous machines have to be used.

15.5 Problems and Solutions

In excavation tasks, the problem space is usually an open and continuous world with uncertain state variables. Additionally, the effects of the applied action can not be modeled entirely, and it is not always possible to predict the resulting world state after applying an action. There are some Strong interaction with environment. For example, soil conditions are often unpredictable, and sometimes there are some obstacles. Accordingly, the forces needed to generate desired digging movements can not be determined correctly, and the digging action has a different result as expected.

To solve the specified excavation tasks using an autonomous system, a deviation of the job on multiple levels is required. The fragmentation of the task starts from the partitioning of the excavation volume on top down to the execution of the different digging states with the low-level control system's skills. By this deviation, the job should be solved by using a provided set of reactive primitive skills. These also are identified and developed according to the requirements of the excavation tasks. It is solved by a hierarchical

task planning method using multiple stages of different kinds of planners and execution strategies. Usually, in hierarchical task planning, the problem will be described in a very structural formal language to use general planners to find a solution. A set of primitive tasks will be calculated, which can be directly executed by an implemented action. Applied in the determined order, they will establish the goal state. For the excavation tasks, a specialized solution will be introduced, containing different planning approaches on each stage of the hierarchy to calculate the execution configurations and determine the needed actions. On the lowest level, a couple of primitive skills are identified needed to perform the excavation tasks. They are implemented as a couple of reactive control systems providing their functionality. These are brought together using a control network and a couple of state machines, implementing the general execution cycle. Usually, to control a robot end-effector along a given trajectory, a complete determination of its forces is required. As the properties of the dug material can be very unpredictable, it is pretty challenging to examine the needed forces. Tthis problem can be addressed by using adaptive control methods together with an iterative planning strategy. With the planner in each digging iteration, a new configuration for executing the actions will be determined according to the particular world state. Furthermore, even while the execution adaptions in control are performed reacting to changing conditions

To solve the excavation task it will be divided into a hierarchy of subtasks. With the partitioning of the excavation area into smaller parts which can be handled without replacement of the machine the first step townwards the hierarchy is allready done. On each of this local excvation positions a local subtask has to be solved using a local excavation cycle. Even this subtask will be additionally devided into smaller pieces and a multi level task hierarchy will be created. In each level the task will be more devided and the contained subtasks are more abstract. Additionally the reusability increases with the task definition in each abstraction level. With the hierarchy the task will be connected to the machine properties. So the task implementations on lower levels fit more to the actual present functions the excavator provides while the upper levels fits more to the intial task structure. So in the most upper level the local trenching is defined which will be split into the phases. This will be connected to the required machine functions throught multiple hierarchy layers defining different control laws.

15.6 Planning Algorithm

15.6.1 Optimal Working Area Partition

Often the size of the excavation volume exceeds the range of an excavator if it will stay mounted at one place. This happens if the working area is larger as it can be reached with the excavator arm. Caused by this it is necessary to divide the excavation area into smaller pieces which do not exceed the range of the arm, each. If these are determined the machine can move between different mounting positions from which it can do its digging work inside of the smaller areas which can be reached by the arm. Such a partition should be done as most optimal as possible. Which leads into the need of finding a set of small area pieces which covers the whole working area. While an optimal solution would be to find the smallest available set of such pieces. It is the problem of finding the smallest set of subsets of a universe set whose union will cover the whole universe. With defining the working area as universe it is possible to map the initial problem of dividing the working

area to this problem as described later in detail. As the *Set Cover Problem* is NP-complete there exist no solution to find the optimal covering set. But in literature are multiple approximation algorithms described which are able to calculate use full solutions. As finding solutions for such problems is an interesting field there are always some better approximation algorithms published.

In the following sections the Set Cover Problem and approximation algorithms are described in detail. Additionally a description is given how the actual problem of partitioning of the working area is mapped to it.

15.6.2 Set Covering Problem

NP-complete problems do not have a solution which can be calculated in polynomial time but for some of them it is possible to find nearly optimal approximations. Such problems are also called optimization problems and the algorithms which can be used to find solutions are approximation algorithms. If a good approach for handling an optimization problem is found, it is possible to give a measurement for how close a calculated solution is to the optimal one. A class of $p(n)$–approximation algorithms can be defined. With C_A is the cost for the solution found by the algorithm and C_{opt} is the optimal cost the following equation holds:

$$\frac{1}{p(n)} C_{opt} \leq C_A \leq p(n) C_{opt} \tag{15.16}$$

Given is a finite range space S which consists of a finite set U, the universe, and a set of subsets of U which is called the ranges R. Additionally with a cost function $c : S \to \mathbb{R}_+ a$ mapping for a given range to a value defining its cost for including it to the solution set is given. With the set cover problem a subset of the ranges should be found which covers the full universe U which has minimal costs.

$$S = (U, \mathcal{R}) \tag{15.17}$$

$$\forall R \in \mathcal{R} : R \subseteq U \tag{15.18}$$

A solution set $\mathcal{L}$ for the set cover problem contains a subset of the ranges for which every element of the universe is an element of one of the solution sets in $\mathcal{L}$.

$$\mathcal{L} = \{\mathcal{X} \subseteq \mathcal{R} | \forall u \in U : \exists X \in \mathcal{X} : u \in X\} \tag{15.19}$$

It is possible to calculate the cost for applying the found solution with by using the following function:

$$C(\mathcal{L}) = \sum_{R \in \mathcal{L}} c(R) \tag{15.20}$$

There is a special case if the cost of using a range for the solution is equal for each of them ($c(R) = 1$), it will reduce the problem to finding the set of a minimal number of ranges covering the universe. As the set cover problem is NP-hard, there exist no approach for finding the optimal solution set in polynomial time. But approximation algorithms exists which delivers nearly optimal solutions, for example the Greedy Set Cover algorithm.

15.6.3 Excavation Volume Partitioning

As initially mentioned an effective excavation work needs a proper division of the labor. Starting from a large excavation volume describing the whole digging work which has to be done, a partitioning of this should be calculated to be able to apply appropriate actions. The large volume has to be divided in pieces which can be reached by the excavators arm. Accordingly each of the generated parts should not exceed the configuration space of the machine. Or saying the other way around the excavation volume should be covered by pieces representing the configuration space. This leads into the idea of mapping the issue to the *Set Cover Problem*. As the universe is defined as the set which should be covered by the ranges, the excavation area will be mapped to this set. This means the positions which are inside of the digging area are used to build the universe set. Formally the set containing all positions which are in the digging area will be defined as:

$$E = \{p | p \quad \text{is in excavation area}\} \tag{15.21}$$

For the definition of the ranges shapes according the available digging actions or at leastvthe configuration space has to be generated. As they will describe a local activity the covered cells of such a shape are dependent on the excavators mounting pose. Accordingly, to build up the range set this shape, which also should be called digging primitive, will be moved to all of the possible mounting poses available. This poses not only include the position of the machine, it also includes its orientation.

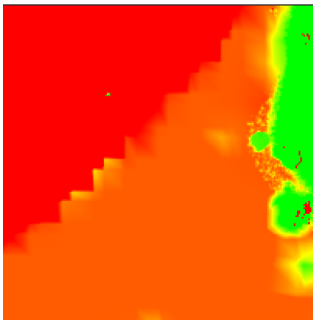


Figure 15.10: Environment map

Figure 15.11: Excavation Target

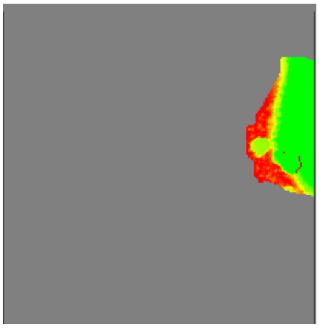


Figure 15.12: Excavation Map

Figure 15.10 to 15.12 shows a visualization of exacvation maps which is resulted from an Experiment. With the different colors the height values of the corresponding grid cells are represented. Red areas describe a very low height while green ones are the highest. The quarry level on which the work should be done is shown in orange, as the pile should be removed its area in the target map (figure 15.11) is given in orange, too. In the maps describing the actual excavation task parts are shown in gray, these are outside of the working area and the surface does not matter for the result. In a first step candidates for mounting the machine to execute the digging work are examined. This is done based on a map of accessible positions which has to be calculated before. An example for this is shown in figure 7.3a, which is a grid map representation of the positions which are safe to mount the excavator. There the grid cells which are in green color are accessible but the

red colored cells can not be used. To determine the candidates from which the excavation area can be reached a reachability sphere is used with a radius according the range of the excavator defined by its kinematics. By using the given inputs the stand candidates will be calculated and stored into a new grid aspect map as shown in figure 15.14. There, the usable stand candidates are marked with a red color. These can be used to determine the range sets together with the reachable excavation points in the next step.

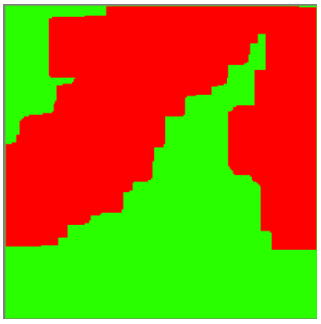


Figure 15.13: Accessible Positions

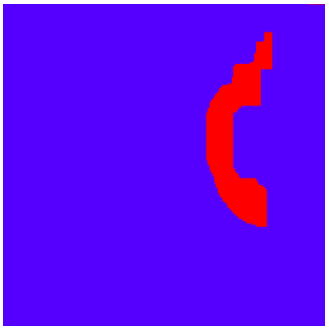


Figure 15.14: Stand Candidates

15.6.4 Digging Starting Point Determination using Rating Functions

For determining the starting position for each of the iterations of the local digging cycles rating functions are used. It describes the two rating functions, *Difference Rating* and *Border Rating* which are added together and multiplied with the *Workspace Rating* and *Distance Rating*. In the present approach the Planning Algorithm Distance Rating, Difference Rating and Border Rating are used as they are defined in the work of Schmidt. Instead of using the Workspace Rating the rated positions are shrinked to the reachable configuration space as well as the range of the previous planned local action by multiplicating them with the others as additional appropriate rating functions.

Hence, first the difference between the desired shape and the current surface is calculated to get the remaining height differences for the excavation work. To prepare the difference map for the further rating functions it will be shrinked to the negative cells only, to consider only this areas where soil has to be removed. This is done as filling-up is not in the scope of this work and even should be done in further working steps using a different set of functionality. After this the differences will be multiplied with -1 to make them positive, followed by a normalization with the lowest and biggest height differences to bring them to the range between zero and one. Based on this the first rating function, which is the Difference Rating, can be caluclated. This function will rate excavation areas which contain more soil to be removed higher as areas with less height differences. Additionally the remaining soil in the neighborhood of each position are considered to push the digging start into areas with bigger workload. See also [Groll 17].

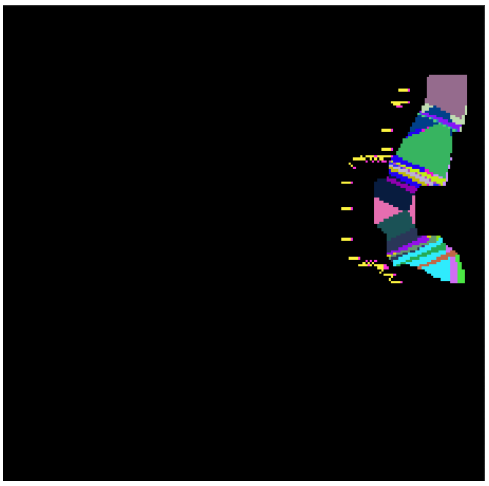


Figure 15.15: Partitioning Result

16. Literature For the Lecture & Basics For the Lecture

Generall

Mechanics of Terrestrial Locomotion	[Zimmermann 09]	
Geometric fundamentals of robotics	[Selig 05]	
Control Prolems in Robotics and Automation	[Siciliano 98]	
Fundamentals of Robotic Mechanical Systems	[Angeles 13]	InfoBib
Taschenbuch Robotik-Montage-Handhabung	[Hesse 16]	InfoBib
Intelligent robotics and applications	[Caihua 08]	InfoBib
Handbook of robotics	[Siciliano 08]	InfoBib
Robot Vision	[Sommer 08]	InfoBib
Autonome mobile Systeme	[Berns 07]	InfoBib
Handbuch Robotik	[Haun 13]	InfoBib
Introduction to robotics	[Craig 05]	InfoBib
Introduction to autonomous mobile robots	[Siegwart 11]	InfoBib
Mobile Robotik	[Nehmzow 12]	InfoBib
Parallel robots	[Merlet 05]	InfoBib
Visuell basierte Robotik	[Sommer 96]	InfoBib
Informationsverarbeitung in der Robotik	[Dillmann 13]	InfoBib

Used works

Roboter - Geschichte, Technik, Entwicklung	[Ichbiah 05]	Chapter 1
Robotik: Programmierung intelligenter Roboter	[Siegert 13]	Chapter 2
Kinematik und Robotik	[Husty 97]	Chapters 3+
Theory of Applied Robotics: Kinematics, Dynamics, and Control	[Jazar 10]	Chapters 3-6
Introduction to Robotics	[Craig 05]	Chapters 3-9
Industrieroboter	[Weber 22]	Chapter 10
Robotergreifer	[Hesse 05]	Chapter 11
Planning and Execution of Excavation Tasks	[Groll 22]	Chapter 15

Conferences

Autonome Mobile Systeme
International Confernce on Robots and Systems
International Confernce on Robotics and Automation (ICRA)
IEEE robotics & automation
IEEE transactions on robotics
International onference on Intelligent Robots and Systems (IRUS)

A. Notations and Fundamentals for the Modeling of Robot Systems

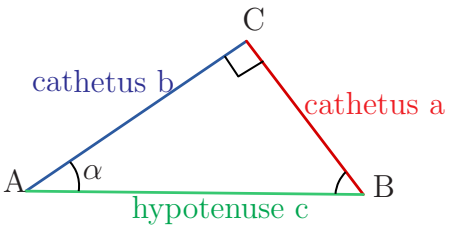
A.1 Notations

- Scalars are described with lowercase letters (for example: s)
- Matrices are described with capital letters (for example: A)
- vectors are marked with arrows (for example: $\vec{u}$)
- Identifiers for scalars, vectors or points are given as an index at the bottom right (for example: $\vec{u}_1$)
- If sinus or cosinus appear as components of the matrices, they are specified as follows:
  $\cos(\theta_1) = C\theta_1 = C_1$ and $\sin(\theta_1) = S\theta_1 = S_1$
  $\cos(\theta_1 + \theta_2 + ... + \theta_n) = C_{123...n}$ and $\sin(\theta_1 + \theta_2 + ... + \theta_n) = C_{123...n}$
- Coordinate systems (frames) are described with capital letters or numbers (for example: B)
- If a vector refers to a specific frame, the frame identifier is given at the top left (for example: ${}^{B}\vec{u}$)
- If a matrix is transformed from a reference frame B into a basic frame A, this will be specified at the bottom or top left (for example: ${}^{A}_{B}R$)

A.2 Trigonometric Functions

The trigonometric functions sin, cos, tan, and cot are defined as the ratios of corresponding sides in a right triangle (between 0° and 90°).

$$\sin\alpha = \frac{\text{opposite}}{\text{hypotenuse}} = \frac{a}{c}$$
$$\cos\alpha = \frac{\text{adjacent}}{\text{hypotenuse}} = \frac{b}{c}$$
$$\tan\alpha = \frac{\text{opposite}}{\text{adjacent}} = \frac{\frac{a}{c}}{\frac{b}{c}} = \frac{a}{b} = \frac{\sin\alpha}{\cos\alpha}$$
$$\cot\alpha = \frac{\text{adjacent}}{\text{opposite}} = \frac{\frac{b}{c}}{\frac{a}{c}} = \frac{b}{a} = \frac{\cos\alpha}{\sin\alpha}$$



For an arbitrary angles we define them for the unit circle. The length of the hypotenuse is $\sqrt{x^2+y^2} = 1$, or $x^2 + y^2 = 1$. Then, $\sin(\alpha) = y$, and $\cos(\alpha) = x$.

Radian of an angle α is defined by the length of the arc of unit circle corresponding to α. See figure A.1

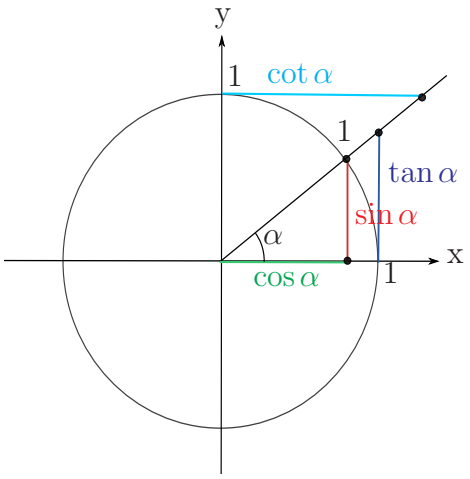


Figure A.1: Trigonometric Functions

The *radian* of an angle α (in degrees) is the length of the arc that encloses the angle with the unit circle (radius r = 1).

A.2.1 Properties of sin and cos

	sin	cos
Domain	$-\infty < x < \infty$	$-\infty < x < \infty$
Codomain	$-1 \le \sin x \le +1$	$-1 \le \cos x \le +1$
Period	2π	2π
Symmetry	odd	Even
Roots	$x_k = k \cdot \pi$	$x_k = \frac{\pi}{2} + k \cdot \pi$
Maxima	$x_k = \frac{\pi}{2} + k \cdot 2\pi$	$x_k = k \cdot 2\pi$
Minima	$x_k = \frac{3\pi}{2} + k \cdot 2\pi$	$x_k = \pi + k \cdot 2\pi$
Transformations	$\sin(90° - \alpha) = \cos\alpha$	$\cos(90° - \alpha) = \sin\alpha$

A.2.2 Values of sin and cos

	sin	cos
0°	0	1
30°	$\frac{1}{2}$	$\frac{1}{2}\sqrt{3}$
45°	$\frac{1}{2}\sqrt{2}$	$\frac{1}{2}\sqrt{2}$
60°	$\frac{1}{2}\sqrt{3}$	$\frac{1}{2}$
90°	1	0

A.2.3 Some additional Formula regariding to sin and cos

- $\sin(x_1 \pm x_2) = \sin x_1 . \cos x_2 \pm \cos x_1 . \sin x_2$
- $\cos(x_1 \pm x_2) = \cos x_1 . \cos x_2 \mp \sin x_1 . \sin x_2$
- $\tan(x_1 \pm x_2) = \frac{\tan x_1 \pm tan x_2}{1 \mp \tan x_1 . \tan x_2}$

A.2.4 Graphs of sin and cos

Figures A.2 and A.3 show a typical picture of sin and cos.

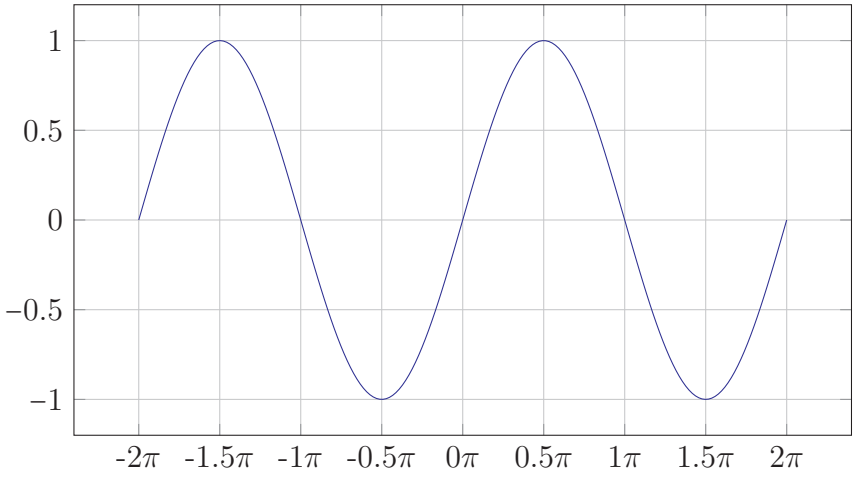


Figure A.2: sin(x)

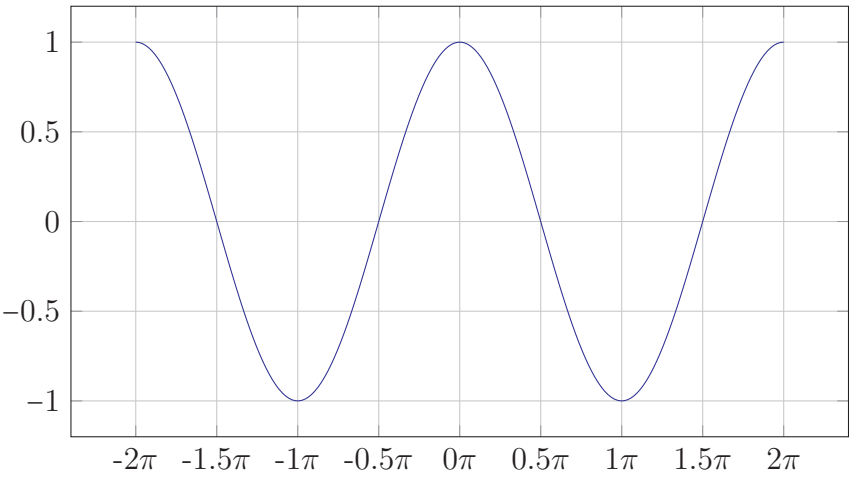


Figure A.3: cos(x)

A.2.5 Cosine Rule/Sine Rule

Cosine Rule:

The relationships between the three sides (a, b, c) of a plane triangle (see A.4) and the cosine of one of the three angles are:

$$a^2 = b^2 + c^2 - 2 \cdot b \cdot c \cdot \cos(A)$$
$$b^2 = a^2 + c^2 - 2 \cdot a \cdot c \cdot \cos(B)$$
$$c^2 = a^2 + b^2 - 2 \cdot a \cdot b \cdot \cos(C)$$

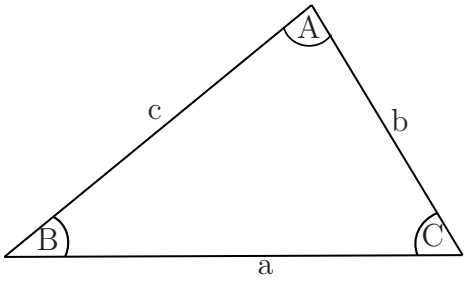


Figure A.4: Cosine rule

Sine Rule:

The relationship between the three sides (a, b, c) of a plane triangle (see A.4) and the sine of one of the three angles is:

$$\frac{\sin A}{a} = \frac{\sin B}{b} = \frac{\sin C}{c}$$

A.3 3D-Coordinate Systems

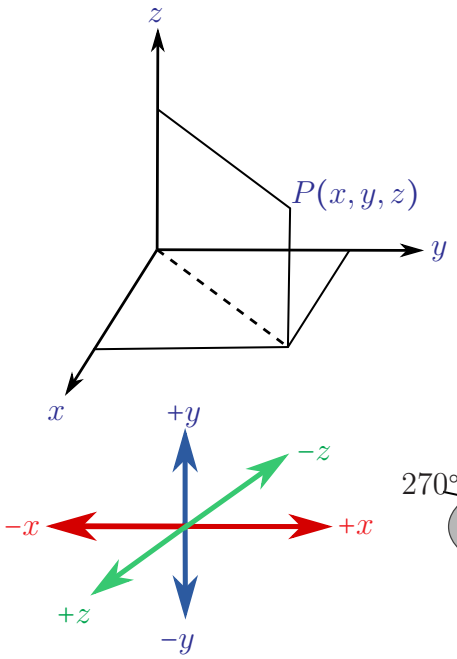


Figure A.5: Cartesian coordinates

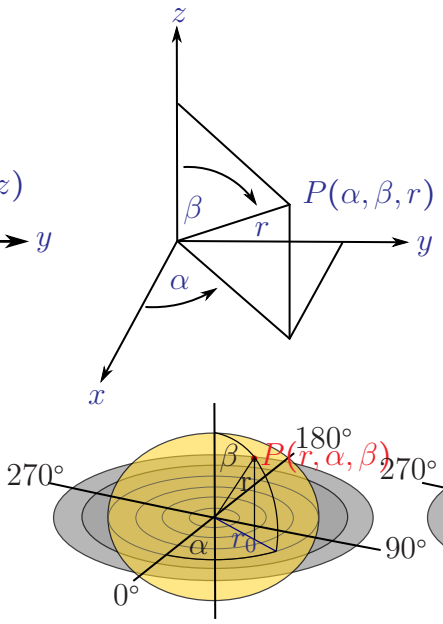


Figure A.6: Spherical coordinates

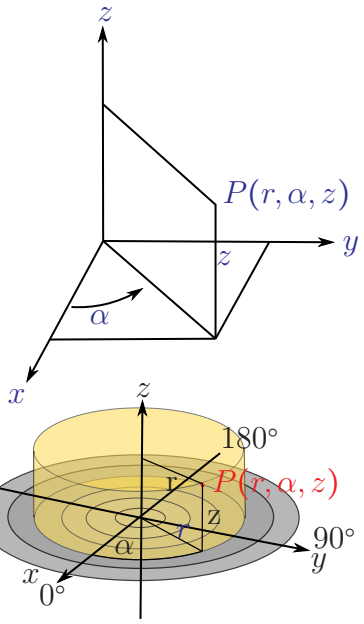
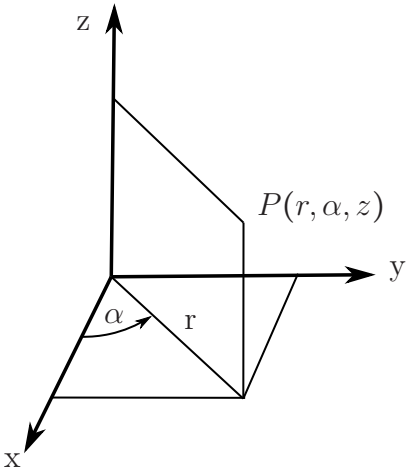


Figure A.7: Cylindrical coordinates

A.3.1 Transformation of Coordinate Systems

Coordinate systems can be transformed to each other based on the following formulas:



Cartesian coordinates→Cylindrical coordinates

(x,y,z)→(r,α,z)

$$r = \sqrt{x^2 + y^2}$$

$$\tan(\alpha) = \frac{y}{x}$$

$$z = z$$

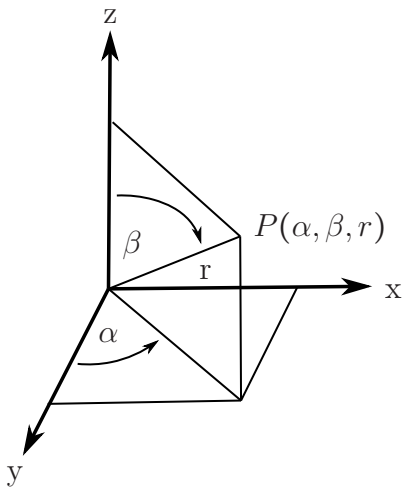
Cylindrical coordinates→Cartesian coordinates

(r,α,z)→(x,y,z)

$$x = r \cdot \cos(\alpha)$$

$$y = r \cdot \sin(\alpha)$$

$$z = z$$



Cartesian coordinates→Spherical coordinates

(x,y,z)→(r,α,β)

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\cos(\beta) = \frac{z}{\sqrt{x^2 + y^2 + z^2}}$$

$$\tan(\alpha) = \frac{y}{x}$$

Spherical coordinates→Cartesian coordinates

(r,α,β)→(x,y,z)

$$x = r \cdot \sin(\beta) \cdot \cos(\alpha)$$

$$y = r \cdot \sin(\beta) \cdot \sin(\alpha)$$

$$z = r \cdot \cos(\beta)$$

A.4 Vector Calculation

A.4.1 Scalar Product

Let us $\vec{a}$ and $\vec{b}$ be two n-dimential vectors, then their *Scalar Product* is defined as following:

$$\vec{a}.\vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + ...a_n b_n$$

or equivalently,

$$\vec{a}.\vec{b} = |a|.|b|.\cos\theta$$

which θ is the smallest angle between a and b.
If two vectors are orthogonal, then their scalar product is 0. Commutative and distributive properties hold, but associative does not hold. With respect to scalars, it is:

$$n(\vec{a}.\vec{b}) = (n.\vec{a}).\vec{b} = \vec{a}.(n.\vec{b})$$

It holds:

- $\vec{a}.\vec{b} = |\vec{a}|^2$
- $\vec{e_x}.\vec{e_x} = \vec{e_y}.\vec{e_y} = \vec{e_z}.\vec{e_z} = 1$
- $\vec{e_x}.\vec{e_y} = \vec{e_y}.\vec{e_z} = \vec{e_z}.\vec{e_x} = 0$

A.4.2 Cross Product/Vector Product

Cross product $\vec{a} \times \vec{b}$ in spaces is a Vector, which is perpendicular to $\vec{a}$ and $\vec{b}$ and therefore normal to the plane containing them.

For $\mathbb{R}^3$, it is defined as:

$$\vec{a} \times \vec{b} = |a|.|b|. \sin\theta.\vec{e}$$

such that θ is the angle between the vectors, ans $\vec{e}$ is perpendicular unit vector.

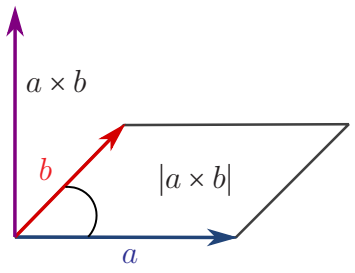


Figure A.8: Vector Product

Cross product can be computed componentwise for $\mathbb{R}^3$:

$$\vec{a} \times \vec{b} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{bmatrix}$$

Magnitude of the cross product is equal to the area of the parallelogram

$$A_P = |\vec{a} \times \vec{b}| = |\vec{a}| \cdot |\vec{b}|. \sin\theta$$

For parallel vectors the cross product is 0. It holds:

$$\vec{a} \times \vec{a} = \vec{0}$$

Distributive und anticommutative property hold. $|\vec{a} \times \vec{b}|$ can be computed as following:

$$|\vec{a} \times \vec{b}| = det \begin{bmatrix} \vec{e}_1 & a_1 & b_1 \\ \vec{e}_2 & a_2 & b_2 \\ \vec{e}_3 & a_3 & b_3 \end{bmatrix}$$

A.4.3 Triple Product

Triple Product is a combination of cross and scalar product.

$$V_{\vec{a},\vec{b},\vec{c}} = (\vec{a} \times \vec{b}).\vec{c} = (\vec{b} \times \vec{c}).\vec{a} = (\vec{c} \times \vec{a}).\vec{b}$$
$$= det \begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{bmatrix}$$

Magnitude of triple product is signed volume (V) of the prism defined by the three vectors. For right handed coordinate systems $V > 0$, and for left handed coordinate systems $V < 0$. For linear dependent vectors it is 0, and also anticommutative property holds.

A.5 Matrices and their Properties

A.5.1 Determinant

Determinant of a $n \times n$-Matrix (Laplace´s formula for $i-th$ row) is defined as following:

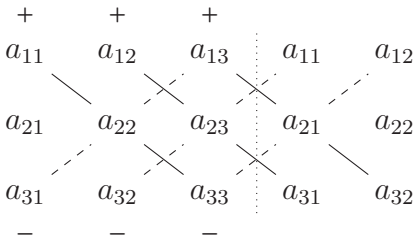
$$detA = \sum_{i=1}^{n} (-1)^{i+j} a_{ij} detA_{ij}$$

Rule of thump for 2×2 matrices (rule of sarrus) is:

$$\det \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = a_{11}.a_{22} - a_{12}.a_{21}$$

Rule of thump for 3×3-marices (rule of Sarrus) is:

$$\det \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}$$



Example:

Expanding the determinant along row 1:

$$\det \begin{bmatrix} 0 & 3 & 2 \\ 1 & 2 & 3 \\ 2 & 1 & 1 \end{bmatrix} = 0.\det \begin{bmatrix} 2 & 3 \\ 1 & 1 \end{bmatrix} - 3.det \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix} + 2.\det \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} = -3. - 5 + 2. - 3 = 15 - 6 = 9$$

A.5.2 Properties of Determinants

- $detA = detA^T$
- $det\lambda A = \lambda^n.detA$
- $detA^{-1} = \frac{1}{detA}$ for $detA \neq 0$
- Determinant is 0, if
  - all elements of a row/column are 0.
  - two rows are linearly dependent.
- Exchanging two rows changes the sign of the determinant.

A.5.3 Properties of the Eigenvalues

- The *Trace* of a matrix is the sum of all eigenvalues:

$$tr(A) = \sum_{i=1}^{n} \lambda_i \tag{A.5.1}$$

- The determinant of a matrix is the product of all eigenvalues:

$$det(A) = \prod_{i=1}^{n} \lambda_i \tag{A.5.2}$$

- The eigenvectors belonging to different eigenvalues are linearly independent.

A.5.4 Pseudo-Inverse of Matrices

For each $m \times n$ matrix A, Pseudo-Inverse of A is defined as a $n \times m$ matrix A^+ satisfying all of the following four criteria, known as the Moore–Penrose conditions:

- $AA^+A = A$
  AA^+ does not need to be the general identity matrix, but it maps all column vectors of A to themselves.
- $A^+AA^+ = A^+$
  A^+ acts like a weak inverse.
- $(AA^+)^T = AA^+$
  AA^+ is Hermitian.
- $(A^+A)^T = A^+A$
  A^+A ia also Hermitian.

Basic Properties

- The pseudo-inverse exists and is unique.
- The pseudo-inverse of a zero matrix is its transpose.
- If A is invertible, then its pseudo-inverse is its inverse:

$$A^+ = A^{-1}$$

- The pseudo-inverse of the pseudo-inverse is the original matrix:

$$(A^+)^+ = A$$

- The pseudo-inverse of a scalar multiple of A is the reciprocal multiple of A^+:

$$(\lambda A)^+ = \frac{1}{\lambda} A^+, \quad \text{for} \quad \lambda \neq 0$$

A.6 Complex Numbers

A *Complex number*, is a number in the form of $a + ib$, which a and b are real numbers and i is the imaginary unit with $i = \sqrt{-1}$, and $(\pm i)^2 = -1$

Let us $C_1 = a + bi$ and $C_2 = c + di$ be two complex numbers. Basic mathematical operations are defined as following:

- **Adding:** $C_1 + C_2 = (a + ib) + (c + id) = (a + c) + i(b + d)$
- **Subtracting:** $C_1 - C_2 = (a + ib) - (c + id) = (a - c) + i(b - d)$
- **Multiplying:** $C_1 \times C_2 = (a + ib) \times (c + id) = ac + iad + ibc + i^2bd = (ac - bd) + i(ad + bc)$
- **complex conugate of C_1:** $a - ib$ (see figure A.9)
- **Inverse of C_1:** $\frac{1}{C_1}\frac{1}{(a+ib)} = \frac{1}{(a+ib)} \times \frac{a-ib}{a-ib} = \frac{a-ib}{a^2+b^2}$

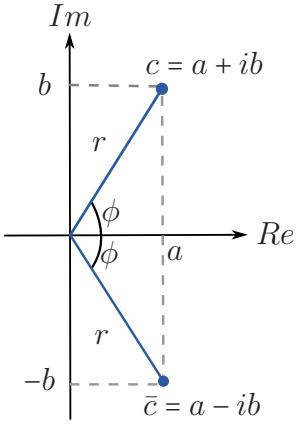


Figure A.9: Complex Numbers

A.6.1 Historical Aspects

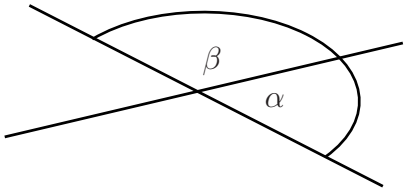
Each complex number $a + bi$ is viewable as a point in a plane and its transfer to space leads to Quaternions.
Word *Quaternion* comes from the latin Word quaternio. Remember that multiplication with triple is not possible, but with quadruples. The basic rules of multiplication is: $i^2 = j^2 = k^2 = ijk = -1$. Then,

$$Q = r_1 + i \cdot r_2 + j \cdot r_3 + k \cdot r_4$$

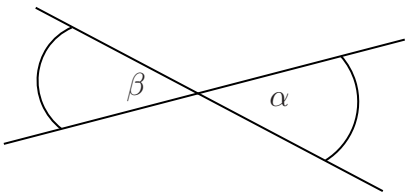
A.7 Angles

Types of Angles:

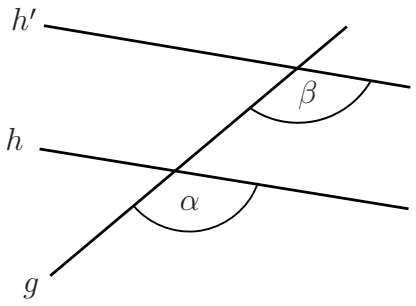
Supplementary Angles: Two angles whose measures add up to 180 degrees. Supplementary angles can be placed so that they form a straight line



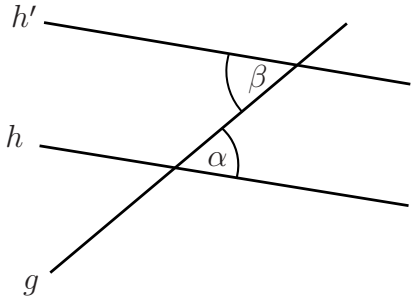
Vertical Angles: Two angles formed by intersecting lines. They cannot be adjacent but are always equal in measure.



Corresponding angles: Corresponding angles are formed where a line known as an intersecting traversal crosses through a pair of straight lines. Corresponding angles are the pair angles that are found in the same relative position on different intersections.



Alternate Angles: Alternate angles are angles that are in opposite positions relative to a transversal intersecting two lines.



A.8 Jacobi-Matrix

Let $f : \mathbb{R}^n \to \mathbb{R}$ be the total derivation of y, $\vec{y} = f(\vec{x})$ for $x, y \in \mathbb{R}$.

$$\begin{aligned}
\vec{y}_1 &= f_1(x_1, x_2, \ldots, x_n) \\
\vec{y}_2 &= f_2(x_1, x_2, \ldots, x_n) \\
&\vdots \\
\vec{y}_n &= f_n(x_1, x_2, \ldots, x_n)
\end{aligned}$$

$$\begin{aligned}
dy_1 &= \frac{df_1}{dx_1}dx_1 + \frac{df_1}{dx_2}dx_2 + \cdots + \frac{df_1}{dx_n}dx_n \\
dy_2 &= \frac{df_2}{dx_1}dx_1 + \frac{df_2}{dx_2}dx_2 + \cdots + \frac{df_2}{dx_n}dx_n \\
&\vdots \\
dy_n &= \frac{df_n}{dx_1}dx_1 + \frac{df_n}{dx_2}dx_2 + \cdots + \frac{df_n}{dx_n}dx_n
\end{aligned}$$

or equivalently,

$$\begin{bmatrix} dy_1 \\ dy_2 \\ \vdots \\ dy_n \end{bmatrix} = \begin{bmatrix} \frac{df_1}{dx_1} & \frac{df_1}{dx_2} & \cdots & \frac{df_2}{dx_n} \\ \frac{df_2}{dx_1} & \frac{df_2}{dx_2} & \cdots & \frac{df_1}{dx_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{df_n}{dx_1} & \frac{df_n}{dx_2} & \cdots & \frac{df_n}{dx_n} \end{bmatrix} = \begin{bmatrix} dx_1 \\ dx_2 \\ \vdots \\ dx_n \end{bmatrix}$$

Alternatively, $d\vec{y} = df(\vec{x}) = \frac{df(\vec{x})}{d\vec{x}} dx$ with a Jacobi-matrix $J(\vec{x}) = \frac{df(\vec{x})}{d\vec{x}}$.

A.9 Geometry in Space

- A straight line in space can be described by:

$$g := \overrightarrow{a} + \lambda \overrightarrow{b}$$

with the support vector $\overrightarrow{a}$ and direction vector $\overrightarrow{b}$

- The distance d between point x and line g is calculated from:

$$d = \frac{|\overrightarrow{b} \times (\overrightarrow{x} - \overrightarrow{a})|}{|\overrightarrow{b}|}$$

- The distance between two straight lines g_1, g_2 is calculated with:

$$d = \frac{|\overrightarrow{b_1} \times (\overrightarrow{a_2} - \overrightarrow{a_1})|}{|\overrightarrow{b_1}|}$$

- The intersection angle of two straight lines g_1, g_2 is:

$$\varphi = \arccos\left(\frac{\overrightarrow{b_1} \cdot \overrightarrow{b_2}}{|\overrightarrow{b_1}| \cdot |\overrightarrow{b_2}|}\right)$$

- A level in space can be described by:

$$E := \overrightarrow{a} + \lambda \overrightarrow{b} + \mu \overrightarrow{c}$$

with $\overrightarrow{a}$ as a base and $\overrightarrow{b}, \overrightarrow{c}$ as direction vectors

alternativly:

$$E := \overrightarrow{n} \cdot (\overrightarrow{a_1} - \overrightarrow{a_2}) = 0$$

with $\overrightarrow{a_2}$ as the position vector of the base, $\overrightarrow{a_1}$ as the position vector of the current point and $\overrightarrow{n}$ as the normal vector of $\overrightarrow{a_1} - \overrightarrow{a_2}$

- The distance between a point x and a plane E is calculated as follows:

$$d = \frac{|\overrightarrow{n} \cdot (\overrightarrow{x} - \overrightarrow{a_2})|}{|\overrightarrow{n}|}$$

- The distance between a straight line g and a plane E is calculated as follows:

$$d = \frac{|\overrightarrow{n} \cdot (\overrightarrow{a_g} - \overrightarrow{a_{E2}})|}{|\overrightarrow{n}|}$$

with $E := \overrightarrow{n} \cdot (\overrightarrow{a_{E1}} - \overrightarrow{a_{E2}})$ and $g := \overrightarrow{a_g} + \lambda \overrightarrow{b_g}$

A.10 Physics Background

A.10.1 Translation

Due to inertia, bodies move uniformly without the application of force without changing direction and the following applies:

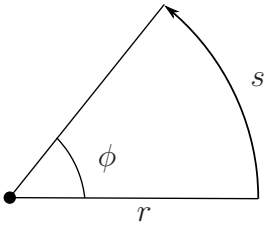
$\vec{v}(t) = \frac{d\vec{s}(t)}{dt}$	Velocity
$\vec{s}(t) = \int_0^t \vec{v}(t)$	Position vector if the start is at the origin at time t=0
$\vec{a}(t) = \frac{d\vec{v}(t)}{dt}$	acceleration
$\vec{v}(t) = \int_0^t \vec{a}(t)$	Speed when $\vec{v}(0) = \vec{0}$
$\vec{v}(t) = \vec{a}t$	if $\vec{s}(t_0) = \vec{v}(t_0) = \vec{0}$ and $\vec{a}(t_0) = const$

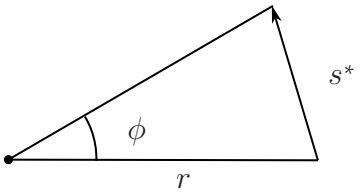
In free fall (acceleration due to gravity $g = 9.81 m/s^2$) the following applies:
$v = gt$, $s = \frac{1}{2}gt^2$, $v = \sqrt{2gs}$ with s = Height of fall

A.10.2 Rotation

The following applies to rotational movements:

$\varphi = \frac{s}{r}$ Rotation angle (s in radians)





$\varphi = \frac{s^*}{r}$	Approximation for small angles
$\omega(t) = \frac{d\varphi(t)}{dt}$	Angular velocity
$\alpha(t) = \frac{d\omega(t)}{dt}$	Angular acceleration
$\omega = 2\pi f = \frac{2\pi}{T}$	for $\omega = const$, f =Angular frequency, T =Orbital time
$\lvert\vec{v}(t)\rvert = r\frac{d\varphi(t)}{dt}$	Tangential speed with uniform circular motion
$\vec{a}(t) = \frac{d\vec{v}(t)}{dt}$	Radial acceleration (points to the center of rotation)

A.10.3 Forces

The following applies to forces:

$\vec{F} = m \cdot \vec{a}$	Newton´s second law (Basic equation of mechanics)
$G = m \cdot g$	Weight ($1kg \approx 9.81N$)
$\vec{F}_Z = -m \cdot \vec{a}_r = -m \cdot \vec{r} \cdot \omega^2$	Centrifugal force of F_Z in rotation

Coriolis Force

Coriolis force reflects radial moving bodies in a rotating frame of reference.

- Straight path B_1 with respect to a rest frame.
- Curved path B_2 with respect to a rotating frame.
- Therefore a force needs to be applied to keep a straight path.

See figure A.10

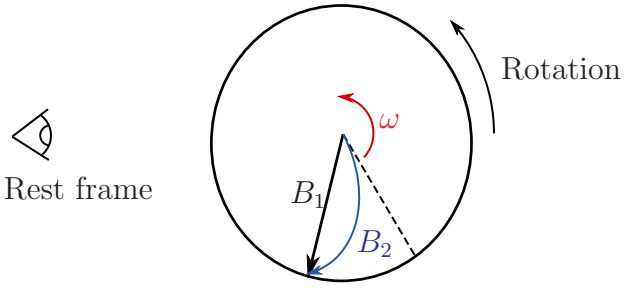


Figure A.10: Forces

A.10.4 Torque and Moment of Inertia

The following applies to torques and moments of inertia:

$\vec{M} = \vec{r} \times \vec{F}$	Torque on body with lever arm $\vec{r}$ and force $\vec{F}$
$M = F \cdot r \cdot \sin\gamma$	Equation for magnitude of torque
$dJ = r^2 dm$	Moment of inertia for a point of mass with mass dm
$J = \int_{Volumen} r^2 dm$	Moment of inertia for a body

With the distance between point of mass and axis r, and mass distribution J relative to rotational axis.
See figure A.11

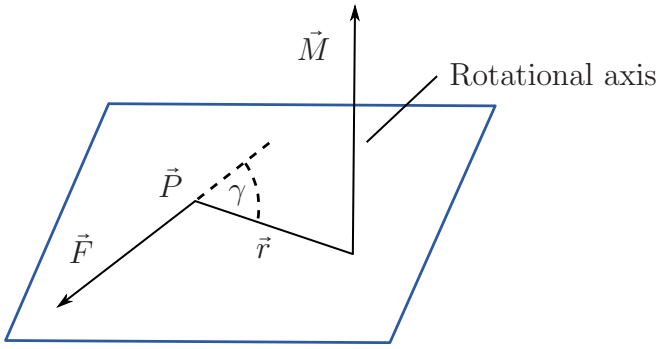


Figure A.11: Torque

Tensor

Tensor is inertia w.r.t x-y-z-system in homogeneous coordinates.

$$M = \int \vec{r}.\vec{r}^T dm = \begin{bmatrix} \int x^2 dm & \int yxdm & \int xzdm & \int xdm \\ \int xydm & \int y^2 dm & \int yzdm & \int ydm \\ \int xzdm & \int yzdm & \int z^2 dm & \int zdm \\ \int xdm & \int ydm & \int zdm & \int dm \end{bmatrix}$$

A.10.5 Newton's Second Rule

Newton's Second Rule describes the relationship between the acceleration of an object $\frac{dv}{dt}$ with mass M, and the force F acting on that object. Then:

$$
\begin{aligned}
M\frac{dv}{dt} &= \sum F_i \\
\sum F_i &= Ma_i \\
\sum r_i \times F_i &= \frac{dJ_{c.m.}}{dt} \\
J_{c.m.} &= \sum r_i \times m_i(\omega \times r_i) \\
r_i &= x_i x + y_i y + z_i z \\
\omega &= \omega_x x + \omega_y y + \omega_z z \\
J_x &= \sum m_i(y_i^2 + z_i^2)\omega_x - \sum m_i x_i y_i \omega_y - \sum m_i x_i z_i \omega_z \\
J_y &= -\sum m_i x_i y_i \omega_y + \sum m_i(z_i^2 + x_i^2)\omega_y - \sum m_i y_i z_i \omega_z \\
J_z &= -\sum m_i x_i z_i \omega_z - \sum m_i y_i z_i \omega_z + \sum m_i(x_i^2 + y_i^2)\omega_z \\
J_{c.m.} &= \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = I\omega \\
I_{xx} &= \sum m_i(y_i^2 + z_i^2)\omega_x \\
I_{yy} &= \sum m_i(z_i^2 + x_i^2)\omega_y \\
I_{zz} &= \sum m_i(x_i^2 + y_i^2)\omega_z \\
I_{xy} &= I_{yx} = -\sum m_i x_i y_i \omega_y \\
I_{xz} &= I_{zx} = -\sum m_i x_i z_i \omega_z \\
I_{yz} &= I_{zy} = -\sum m_i y_i z_i \omega_z
\end{aligned}
$$

Where M is the mass, $a = \frac{dv}{dt}$ is the acceleration and F_i is the external force, ω is an angular velocity vector.

A.10.6 Modelling the Movement of Vehicles

1. Convert all forces F_i to earth-bound coordinates and update the estimate of the speed of the vehicle's center of gravity:

$$v_{t+\Delta t} = v_t + (\sum F_i / M)\Delta t.$$

2. Apply the speed to the center of the mass of the vehicle to determine the new position at time $t + \Delta t$:

$$s_{t+\Delta t} = s_t + v_{t+\Delta t}\Delta t.$$

3. Update the angular speed of the vehicle by integrating the current value of the angular acceleration:

$$\omega_{t+\Delta t} = \omega_t + I^{-1}(\sum r_i \times F_i - \omega \times (I \times \omega))\Delta t.$$

   Note that the forces F_i must be transformed into a vehicle-bound coordinate system

4. Update the orientation of the vehicle, based on the orientation of its center of gravity, by the value $\omega_{t+\Delta t}\Delta t$. The current implementation is based on how the simulation represents rotation. If the rotation of the vehicle is represented with a rotation matrix, $R_{t+\Delta t} = R\Omega$ applies with

$$\Omega = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

Note that $R_{t+\Delta t}$ must be renormalized after the calculation.

A.10.7 Work/Energy

The following applies to work:

$A = \vec{F} \cdot \vec{s}$	Work=force · way
$A = F \cdot s \cdot \cos\delta$	Absolute equation for the force
$A = \int_{(s)} \vec{F}_s d\vec{s}$	Work, if the force is not constant (force $\vec{F}_s$ in the direction of $d\vec{s}$)

The following applies to energy:

$E_{pot} = m \cdot g \cdot h$	Potential energy (mass m, height h)
$E_{kin} = \frac{1}{2} \cdot m \cdot v^2$	Kinetic energy
$E_{rot} = \frac{1}{2} \cdot m \cdot v^2 = \frac{1}{2} \cdot m \cdot r^2 \cdot \omega^2 = \frac{1}{2} \cdot J \cdot \omega^2$	Kinetic energy (rotating body)

After falling from the height h, the body has the kinetic energy:

$$E_{pot} = m \cdot g \cdot h = m \cdot \frac{v^2}{2 \cdot g} \cdot g = \frac{1}{2} \cdot m \cdot v^2$$

A.10.8 Units

Size	Sign	Unit	Name
Path/Distance	s	m	Meter
Time	t	s	Seconds
Speed	v	$\frac{m}{s}$	
Acceleration	a	$\frac{m}{s^2}$	
Angle Speed	ω	$\frac{rad}{s}$	
Angle of Rotation	φ	rad	
Force	$\vec{F}$	$N = \frac{kg\ m}{s^2}$	Newton
Moment of Inertia	J	$kg\ m^2$	
Work	A	$J = \frac{kg\ m^2}{s^2}$	Joule

Bibliography

[Albus 89] J. S. Albus, R. Lumia, J. Fiala, A. J. Wavering et al, "NASREM–The NASA/NBS Standard Reference Model for Telerobot Control System Architecture", 1989.

[Angeles 13] J. Angeles, *Fundamentals of Robotic Mechanical Systems Theory, Methods, and Algorithms*, Springer, 2013.

[Berns 07] K. Berns, T. Luksch, Eds., *Autonome Mobile Systeme 2007*, ser. Informatik aktuell. Kaiserslautern: Springer, October 18–19 2007.

[Caihua 08] X. Caihua, ICIRA, *Intelligent robotics and applications*, Springer, 2008.

[Christaller 02] T. Christaller, M. Decker, M. J. Gilsbach, G. Hirzinger, K. W. Lauterbach, E. Schweighofer, G. Schweitzer, D. Sturma, "Perspektiven für menschliches Handeln in der zukünftigen Gesellschaft", in *Robotik*, vol. 14, Springer. 2002.

[Craig 05] J. J. Craig, *Introduction to robotics: mechanics and control*, Pearson Educacion, 2005.

[Dillmann 13] R. Dillmann, M. Huck, *Informationsverarbeitung in der Robotik*, Springer-Verlag, 2013.

[Dillmann 95] R. Dillmann, M. Kaiser, F. Wallner, P. Weckesser, "PRIAMOS: An advanced mobile system for service, inspection, and surveillance tasks", in *Modelling and Planning for Sensor Based Intelligent Robot Systems*, World Scientific, 1995, pp. 362–383.

[Groll 17] T. Groll, S. Hemer, T. Ropertz, K. Berns, "A behavior-based architecture for excavation tasks", in *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, vol. 34, IAARC Publications. 2017.

[Groll 22] T. Groll, "Planning and Execution of Excavation Tasks", Dissertation, Technische Universität Kaiserslautern, 2022.

[Groover 87] M. P. Groover, *Robotik umfassend*, McGraw-Hill, 1987.

[Haun 13] M. Haun, *Handbuch Robotik: Programmieren und Einsatz intelligenter Roboter*, Springer-Verlag, 2013.

[Henriksson 20] A. Henriksson, A. A. MUNOZ, E. Vartiainen, J. Blom, M. Ralph, "Method For Controlling An Industrial Robot During Lead-Through Programming Of The Robot And An Industrial Robot", 2020.

[Hesse 05] S. Hesse, G. Monkman, R. Steinmann, H. Schunk, *Robotergreifer*, Hanser Fachbuchverlag, 2005.

[Hesse 16] S. Hesse, V. Malisa, *Taschenbuch Robotik-Montage-Handhabung*, Carl Hanser Verlag GmbH Co KG, 2016.

[Husty 97] M. Husty, A. Karger, H. Sachs, W. Steinhilper, "Robotik", in *Kinematik und Robotik*, Springer, 1997, pp. 377–524.

[Ichbiah 05] D. Ichbiah, *Roboter: Geschichte - Technik - Entwicklung*, Knesebeck, 2005.

[Jazar 10] R. N. Jazar, *Theory of applied robotics: kinematics, dynamics, and control*, Springer., 2010.

[Merlet 05] J.-P. Merlet, *Parallel robots*, Springer Science & Business Media, 2005, vol. 128.

[Nehmzow 12] U. Nehmzow, *Mobile robotics: a practical introduction*, Springer Science & Business Media, 2012.

[Pan 12] Z. Pan, J. Polden, N. Larkin, S. Van Duin, J. Norrish, "Recent progress on programming methods for industrial robots", *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 2, pp. 87–94, 2012.

[Reichardt 13] M. Reichardt, T. Föhst, K. Berns, "On software quality-motivated design of a real-time framework for complex robot control systems", *Electronic Communications of the EASST*, vol. 60, 2013.

[Reichardt 17] M. Reichardt, S. Schütz, K. Berns, "One fits more-on highly modular quality-driven design of robotic frameworks and middleware", *Journal of Software Engineering for Robotics (JOSER)*, vol. 8, pp. 141–153, 2017.

[Selig 05] J. M. Selig, *Geometric fundamentals of robotics*, Springer, 2005, vol. 128.

[Siciliano 08] B. Siciliano, O. Khatib, T. Kröger, *Springer handbook of robotics*, Springer, 2008, vol. 200.

[Siciliano 98] B. Siciliano, K. P. Valavanis, *Control problems in robotics and automation*, Springer, 1998.

[Siegert 13] H.-J. Siegert, S. Bocionek, *Robotik: Programmierung intelligenter Roboter*, Springer-Verlag, 2013.

[Siegwart 11] R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza, *Introduction to autonomous mobile robots*, MIT press, 2011.

[Sommer 08] G. Sommer, R. Klette, *Robot Vision*, Springer, 2008, vol. 4931.

[Sommer 96] G. Sommer, J. Pauli, K. Daniilidis, *Visuell Basierte Robotik: Stand und Perspektiven Einer Technologie.*, Universität Kiel. Institut für Informatik und Praktische Mathematik, 1996.

[Weber 22] W. Weber, H. Koch, *Industrieroboter: Methoden der Steuerung und Regelung*, Carl Hanser Verlag GmbH Co KG, 2022.

[Zimmermann 09] K. Zimmermann, I. Zeidis, C. Behn, *Mechanics of terrestrial locomotion: with a focus on non-pedal motion systems*, Springer Science & Business Media, 2009.

Index