CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

Models for Energy Optimization of Robotic Cells

Bc. Matěj Petr

May 2019

Supervisor: doc. Ing. Přemysl Šůcha Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Petr** Jméno: **Matěj** Osobní číslo: **420283**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Modely pro minimalizaci spotřeby robotických buněk

Název diplomové práce anglicky:

Models for Energy Optimization of Robotic Cells

Pokyny pro vypracování:

This thesis addresses optimization of energy consumption of robotic cells. The aim is to propose a detailed mathematical model of this problem, come up with a suitable estimation of energy profiles [1] and test both in the SW tool Process Simulate. The particular objectives of the thesis are:
1) Review the existing works in the domain and analyze mathematical model described in [1,2].
2) Analyze the representation of robotic cells in Process Simulate and propose a suitable mathematical model formulated as integer linear programming.
3) Using data measured in Process Simulate, devise a data driven approach to estimate the energy profile of given a movement for a given robot.
4) Implement the model and the estimator of energy profiles in Process Simulate.

Seznam doporučené literatury:

[1] Libor Bukata, Přemysl Šůcha, Zdeněk Hanzálek, Optimizing energy consumption of robotic cells by a Branch & Bound algorithm, Computers & Operations Research, Volume 102, 2019, Pages 52-66.
[2] Libor Bukata, Přemysl Šůcha, Zdeněk Hanzálek, Pavel Burget. Energy Optimization of Robotic Cells In: IEEE Transactions on Industrial Informatics. 2017, vol. 13, no. 1, 92-102
[3] Michele Gadaleta, Marcello Pellicciari, Giovanni Berselli. Optimization of the Energy Consumption of Industrial Robots for Automatic Code Generation. To appear in Robotics and Computer-Integrated Manufacturing. 2019.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Přemysl Šůcha, Ph.D., optimalizace CIIRC

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **29.01.2019** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **20.09.2020**

doc. Ing. Přemysl Šůcha, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Acknowledgement / Declaration

I wish to express my sincere gratitude to my thesis supervisor doc. Ing. Přemysl Šůcha Ph.D. for all his help, advice, time, and comments.

I also wish to express my thanks to Blumenbecker Prag s.r.o for providing me with robotic cells on which I could run the experiments.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 22. 5. 2019

.....

Abstrakt / Abstract

Optimalizace spotřeby energií je důležitým problémem dneška. Zejména v průmyslu, kde jsou často využívány robotické buňky. Tato práce se zaměřuje na optimalizaci spotřeby robotických buněk. Práce definuje problém jako smíšené celočíselné lineární programování (MILP). Problém je definován s ohledem na možnosti, které poskytuje program Process Simulate od firmy Siemens. V práci je popsán algoritmus řešící tento problém. Algoritmus potřebuje informace o spotřebě energií, proto práce přináší dva způsoby jejího získávání. První je založen na získání informace z ovladače Kuka-Krc. Druhý je založený na datech využívající informace ze standardního ovladače. Experimenty ukázaly, že je možné ušetřit až 8 % spotřebované energie pro časově optimální trajektorie jen úpravami jejich parametrů. Ještě větší úspory lze dosáhnout relaxací doby cyklu.

Klíčová slova: optimalizace spotřeby energií, smíšené celočíselné lineární programování, robotické buňky.

Překlad titulu: Modely pro minimalizaci spotřeby robotických buněk

Energy optimisation is becoming an important problem nowadays. Especially in the industry where robotic cells are often used. This thesis focuses on energy optimisation for the robotic cells. Namely, it provides a definition of the problem as mixed integer linear programming (MILP). The problem is defined with regards to options that are provided by the software tool Process Simulate by Siemens. An algorithm solving the problem is described. Since the algorithm needs information about energy consumption, two ways of obtaining it are provided. One is based on getting the information from Kuka-Krc controller. The other is data-driven based on the information from the default controller. The experiments showed that it is possible to save up to 8 % of energy consumption for time-optimal trajectories just by adjusting their parameters. By relaxing the cycle time, we can provide even bigger saving.

Keywords: energy optimisation, mixed integer linear programming (MILP), robotic cells.

/ Contents

Tables / Figures

Chapter 1
Introduction

The energy consumption is continuously growing. In the United States, energy consumption almost tripled since the 1950s [1]. One of the most significant contributors to this growth is the industrial sector. According to the US Energy Information Administration, in 2017, the industry energy consumption accounted for about a third of total energy consumption in the US [1]. In the Czech Republic, in 2018, it was almost 30 % whereas in the European Union it was 25 % [2].

Naturally, there is a growing interest in energy efficiency in the industry sector. Two main reasons motivate this interest. The first one is economical: for example, plants producing 1000 vehicles a day can consume several hundred GWh of electricity per year [3]. Even saving 1 % of the energy consumed (i.e. thousands of MWh per year) could save hundreds of thousands of Euros a year. The other is environmental. Many international organisations, such as the EU, try to implement environment-friendly policies.

For example, the European Union in 2009 passed legislation (sometimes called the 2020 climate & energy package) that sets several targets including 20 % improvement in energy efficiency [4]. One of the EU projects aiming at the energy efficiency improvement was the now-completed AREUS project (Automation and Robotics for European Sustainable Manufacturing). In the project, several partners from industry and research facilities teamed up to devise new technologies capable of reducing energy consumption and environmental impact of robotic cells and industrial robotics in general. Some of the goals were "[t]he development of new Eco-efficient simulation tools for the design and development of robotic processes, evaluating and also optimizing energy consumption" or "[t]he development of new Eco-efficient optimization tools for the optimization of the overall energy consumption without affecting final productivity" [5].

1.1 Motivation

Industrial robots play an important and ever-growing role in many industry sectors thanks to the possibilities they offer, such as increasing speed, precision, or repeatability of production processes. The downside of the use of industrial robots is increased electricity consumption and consequently, the increased cost of production. For example, in the automotive industry, about 8 % of electricity consumption in production is attributed to industrial robots [6]. The biggest contributors to this consumption are the robotic cells themselves. It has been shown on robotic cells with six robots in the welding shop of Škoda Auto that robot drive consumes 51 % of all consumed energy [7].

Generally, the companies focus more on cycle time optimisation rather than on energy optimisation. However, these two things are not necessarily mutually exclusive; often the energy consumption can be decreased without increasing the cycle time. During the production cycle, there can be many operations that use a maximal speed for a robot, even if it is not necessary for keeping the desired cycle time. Such operations have then very high consumption and may not even have the optimal trajectories with

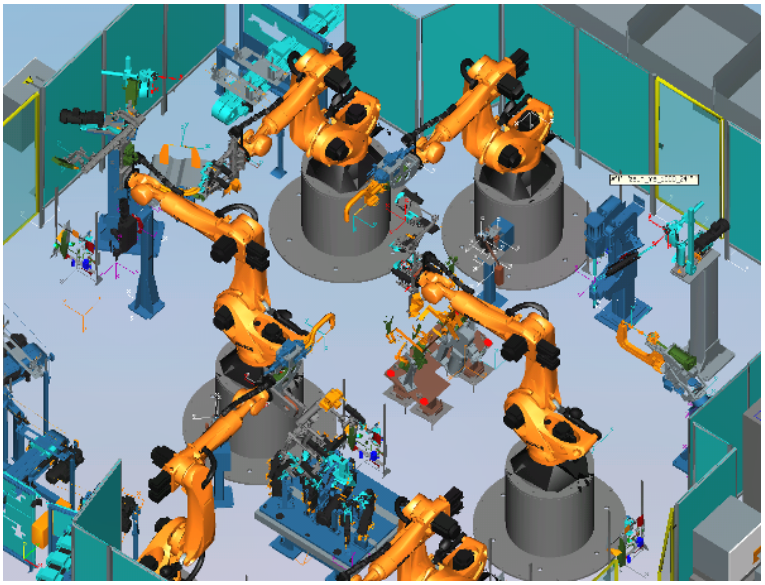

Figure 1.1. A typical robotic cell [8].

respect to energy consumption. That is because the robotic cells are complex systems where one cannot simply optimise all individual trajectories and operations separately to achieve optimal results. The designer of the robotic cells has to take into account also possible collisions between different parts of the cells.

If a collision only occurs when using a particular, high speed of the robot, there are two possible solutions. One is to keep the speed and change the trajectory which can be longer or more complicated and therefore more energy consuming. The other possibility is to follow the original trajectory and lower the speed by bounding the maximum speed. This approach can lead to smaller energy consumption if the speed is not too low. If the robot moves too slowly, energy consumption can, paradoxically, increase because the robot has to deal with the effect of gravity.

Finding which operations should decrease its speed and how to achieve the optimisation is not an easy task, especially for a human designer. The relation between velocity and energy consumption of a single trajectory is not linear. Therefore, optimising even one operation is not trivial. The real robotic cells usually contain many robots that can interact or collide with each other. As can be seen, the optimisation of robotic cells is not a problem that can be solved by hand. It is a problem that requires advanced optimisation algorithms and software capable of precise simulations.

As a consequence of increasing demand for energy efficiency and the need for sophisticated tools which could solve the problem, the field of study of optimisation of robotic cells is getting more attention nowadays.

1.2 Related work

Optimisation of robotic cells is an important field of study. Most papers tackling optimisation of robotic cells deal with time optimisation, or more precisely with increasing throughput, which is the inverse cycle time. For example, Dawande [9] provides a survey about available results about cyclic production. The field of energy optimisation is not as deep.

Carabin et al. [10] summarised state of the art of existing techniques in energy optimisation. The authors focused both on hardware and software aspect of energy optimisa-

tion. As for the software aspect, they emphasised trajectory optimisation and operation scheduling. Meike [6], similarly, provided an overview of energy saving methods which focuses on hardware changes.

Riazi et al. [11] presented the energy optimisation of a multi-robot system by optimising the way the robot moves along the existing path. It is achieved by optimisation of the non-linear model with respect to physical and time constraints. The optimisation criterion is a minimisation of "a weighted sum of the squared joints' accelerations for the trajectories". The authors claim that the presented algorithms can save up to 45 % of total energy consumption.

Riazi et al. [12] in the follow-up paper discussed suitable cost functions, including pseudo power or mechanical jerk minimisation. This procedure, which again preserves cycle time and path, reduces up to 30 % of energy consumption, and moreover, it reduces peak-power by up to 60 %. Both [11] and [12] were a result of the already mentioned AREUS project.

As shown, it is possible to decrease energy consumption without increasing the cycle time; nevertheless, in some cases, it might be more advantageous to allow for relaxation of the cycle time to save more energy. Especially in cases where the robot can execute an operation more slowly, e.g. because it has to wait for the completion of other robotic operations. Meike et al. [13] claim that if an operation can be done in 50 % extra time, it can save up to 20 % of energy per cycle time by moving slower. Pellicciari et al. [14] presented a method of decreasing energy consumption by "time-scaling" that is by slowing down the operations and by reducing idle time. They achieved a 12.1 % reduction of energy loss on a single robot reference trajectory. Meike et al. [15] tested an approach where they time-scaled the last movement. They noticed that energy consumption is not inversely proportional to the speed of the last operation (i.e. duration time). At the lowest speeds (below 30 % of maximum speed) the energy consumption rises due to factors such as the growing consumption of mechanical brakes. The function describing energy consumption of robot at different time-scaling ratios is called Energy Signature (or Energy Profile in [8]).

Gadaleta et al. [16] expanded the Energy Signature concept and replace time-scaling with a method of scaling maximum velocity and acceleration parameters. The authors tested their approach on real robots, and they found out a decrease in consumption of up to 30 %. Also, the results show that velocity and acceleration scaling can provide better results than simple time-scaling.

The above-mentioned trajectory optimisation is an example of so-called local optimisation. The local optimisation focuses only on trajectory optimisation with respect to the physical limits of the robots and obstacle avoidance. This approach is more common in literature. In contrast to that stands the global optimisation that considers the whole robotic cell. This approach is needed to reach the full potential of optimisation.

Such optimisation appears in the pioneering work of Wigström et al. [17]. The authors formulated the nonlinear mathematical model. The model includes constraints ensuring a prespecified event sequence within a task, the maximum cycle time, and that activities acting in the same resource cannot happen at the same time. However, this model does not consider alternative positions of robots or energy saving modes. Moreover, it is limited to cases where the cell works only on one workpiece during the cycle time.

Bukata et al. [8] expanded [17]; they came up with an extended mathematical model which considers various robot speeds, positions, power-saving modes, and alternative orders of operations, and it allows to consider multiple workpieces in the robotic cell at

once. This model can be turned to a MILP. Since such a model is suitable only for small instances, the authors provided a heuristic algorithm capable of solving larger instances. They showed on a real robotic cell from Škoda Auto that the energy consumption could be possibly reduced by about 20 %, the biggest influence on the decrease being speed optimisation.

The authors returned to this problem in another work [18] where they used the same model but solved the problem with a more efficient branch and bound algorithm, which finds similar or better solutions.

1.3 Contribution

The problem of energy optimisation is described in detail with specific emphasis on its connection to Process Simulate framework. The models from works of Bukata et al. [8] and [18] were analysed. Based on these works, a new model for energy optimisation was proposed. Furthermore, two types of energy optimisation were implemented. One is based on measuring energy consumption with Kuka-Krc controller, and the other is based on estimating the energy consumption from data obtained by the default controller. Both of the optimisation types were tested on several robotic cells, including a real robotic cell provided by Blumenbecker Prag s.r.o.

The experimental results showed that it is possible to decrease the energy consumption for the time-optimal cycle time. The decrease was about 8 %. It also showed that relaxation of cycle time could provide even better results. For one cell, relaxation of cycle time allowed to save more than 16 % of energy consumption in comparison with consumption of time-optimal solution.

1.4 Outline

This work could be divided into three parts. The first part that provides a general introduction could be found in this chapter.

Then, there is a theoretical part. In Chapter 2, the problem is formally defined with regards to the options provided by Process Simulate. Based on the problem statement, linear programs describing time and energy optimisation are defined in Chapter 3.

Afterwards, there is a practical part. Chapter 4 discusses possible data-driven approaches to estimating energy consumption. In Chapter 5, there are shown experiments with energy optimisation.

The thesis is finally concluded in Chapter 6, where we also provide some suggestions on how the things mentioned in this thesis could be further improved upon.

Chapter 2
Problem statement

This chapter presents the optimisation problem for robotic cells tailored to the framework Process Simulate provides. At the end of the chapter, there are examples of how the problem defined in Process Simulate corresponds to the theoretical problem formalisation as defined in this chapter.

2.1 Problem Statement

The optimisation problem for robotic cells can be defined as follows. We define a set of robots $R = \{r_1, \ldots, r_n\}$ and a graph $G = (V, \overline{E}, CR)$, $\overline{E} = E \cup L$, describing the production process, where V is a set of vertices, and $\overline{E}$ is a set of oriented graph edges, defined as the union of edges E and links L, and CR is a collision set.

Each vertex represents a non-empty ordered set of Process Simulate operations associated with exactly one robot. The Process Simulate operations, hereafter in this chapter called just operations, are defined by points (3D coordinates) the robot should approach. These points can be grouped into a compound operation. The simulator decides the trajectory the robot undertakes between the points. In addition to this, the robot can perform certain actions such as welding upon reaching the point. Note that Process Simulate distinguishes between points by their zone. The zone says how the robot should approach the point, e.g. for the coarse zone the robot needs to pass in some distance from the points whereas fine zone dictates that the robot moves precisely to the point. The last operation of all vertex sets is always fine. Then for each vertex, we have a minimum and maximum duration, $\underline{dm}_i$, $\overline{dm}_i$. The only exceptions to this are so-called dummy vertices. These vertices have no real operation attached and are used to ensure that a cycle of each robot starts at 0 s regardless of when the first real operation starts. Each robot has exactly one dummy vertex. Also, all vertices with the possible exception of dummy vertices have a minimum mandatory wait after the final operation, $\underline{dw}_i$. This type of wait will be called a *post-wait* from now on.

Certain vertices can also have a wait before the first movement, dn_i, for simplicity we shall call it *prior wait*. This prior wait is introduced mostly to help with collision resolutions. In Figure 2.1, we can see an illustration of the use of prior wait. The pictured Gantt diagram represents a robotic cell with three robots, Robot A, Robot B and Robot C. The first robot has one vertex $A1$, that is five seconds long. The second one has a vertex $B1$ with duration one second and a vertex $B2$ which has a two-second duration, a one-second post-wait and two-second prior wait. Robot C has two vertices: the first one that lasts five seconds and then waits two seconds and the second one that lasts three seconds. There is a dependency between vertices $A1$ and $B2$ that says that vertex $B2$ can start after the vertex $A1$ finished. The length of the link (i.e. time-lag between vertices of two different robots) in the picture (that is to say an offset or time lag between two vertices) has a length of two seconds. Generally, it can take any non-negative real value.

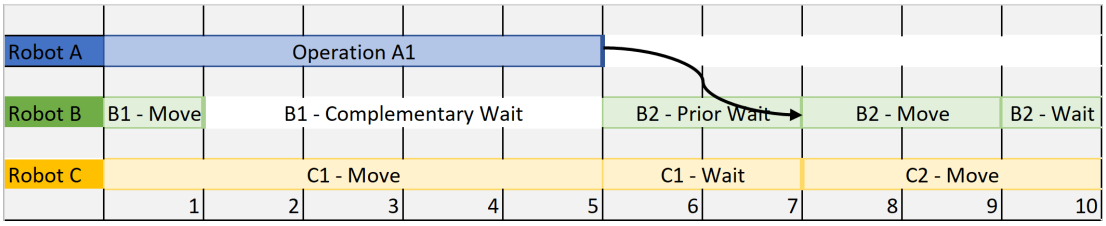

Figure 2.1. An illustration of the use of prior wait.

Let us also assume that all depicted vertices represent a compound operation. Please note that in general, it does not have to be true. One compound operation in Process Simulate can correspond to multiple vertices.

In Figure 2.1, vertex $B2$ has to wait two seconds after vertex $A1$ is finished before it can start (this is the prior wait). The wait can be induced either by setting the link length to two seconds or by adding a two-second wait on the compound operation $B1$. Now let us imagine that there is a collision between robots B and C happening at time 7.5 s. Now the collision can be resolved in two ways. The first one is to try to delay vertex $C2$ by adding a wait time after the vertex $C1$. The other possibility is to start the vertex $B2$ at an earlier time. That can be done by decreasing the prior wait to zero.

Figure 2.1 can also be used for explanation of a different concept. We can see that after vertex $B1$ finishes the robot B must idly wait at least four seconds until Robot A finishes vertex $A1$. Similarly, the robot A stays idle after it finishes its vertex until the end of the operation cycle. We call this wait, which is not induced by any commands, a *complementary wait* dc_i. It is especially important in energy optimisation because a robot consumes energy even if it seemingly does nothing.

As for the oriented graph edges $\overline{E}$, they can either be edges E, that is an ordered pair of vertices of one robot defining their order. Alternatively, they can be links L, i.e. ordered pairs of vertices of two different robots which define time-lag between them. Henceforth the word "edge" shall refer only to the edges from set E. Each edge (i, j) has constant h_{ij} associated with it which denotes whether the edge starts a new cycle or not. Each robot can only have one such an edge. Each link (i, j) has a constant o_{ij} which is an offset defining how much time after completion of vertex i, the vertex j must wait before it can start executing.

We also consider a collision resolution set CR, which contains a quadruple (v_i, v_j, op_i, op_j), where v_i and v_j are vertices which are in the collision and thus cannot be executed at the same time, and op_i, op_j are their respective operation phases. We differ between three operation phases: wait before a vertex (prior wait) N, movement M, wait after a vertex (post-wait, complementary wait) W. The first phase can occur for example when a robot waits for the execution of its first vertex, the second one occurs when a robot executes a vertex, and the last one happens for example when a robot finishes a vertex and waits for a start of the next one. There can be nine types of collisions based on the operation phases of the colliding vertices, $op_i \times op_j = \{N, M, W\} \times \{N, M, W\}$. In general, collision resolution set can be empty.

The notion of the collision resolution was briefly touched above during explanation of prior wait. Here it will be defined more formally. Figure 2.2 shows an example of a collision between robots A and B. As can be seen in the picture, the time interval of the collision in seconds is $(1.0, 2.0)$, the vertices in the collision are $A1$ and $B1$;

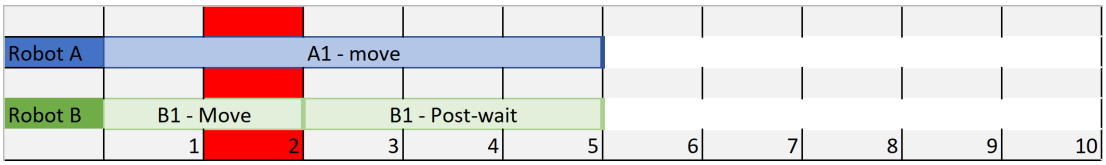

Figure 2.2. An illustration of a collision between two vertices.

for both vertices, the collision happens in their movement phase. We want to avoid such collisions, i.e. execution of the colliding operation phases at the same time. Let $(s_{a1}, s_{a1}+d_{a1})$ be the time interval of the movement phase of vertex $A1$ and $(s_{b1}, s_{b1}+d_{b1})$ be the time interval of the movement phase of vertex $B1$, where s denotes start time of respective operation phase, and d denotes duration of respective operation phase. To avoid this collision, we need to make one operation phase start after the other or vice versa. The collision avoidance can be expressed by the following inequalities.

$$s_{a1} + d_{a1} \leq s_{b1} + xM, \tag{2.1}$$
$$s_{b1} + d_{b1} \leq s_{a1} + (1 - x)M. \tag{2.2}$$

These constraints use the well-known trick of using binary variable x and sufficiently high number M to keep only one constraint active. The other will be virtually unbounded. The meaning of the variable x is the following:

$$x = \begin{cases} 1 & \text{if the second constraint in the collision constraint is active} \\ 0 & \text{if the first constraint in the collision constraint is active.} \end{cases}$$

Figure 2.3 shows two possible resolutions of the collision. In the upper diagram, $x = 1$, and therefore, the movement phase of vertex $A1$ comes after the movement phase of vertex $B1$. In the bottom diagram, $x = 0$, and the order of the movement phases is therefore different.

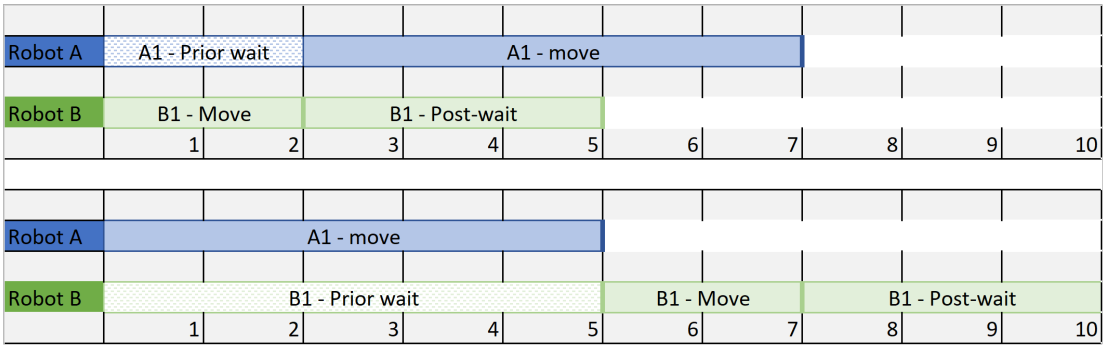

Figure 2.3. An illustration of a collision resolution.

Every optimisation problem needs an optimisation criterion, and this case is no different. As mentioned in Chapter 1, the criteria of interest for robotic cell optimisation are cycle time minimisation and energy consumption minimisation. Formally, we can write the first criterion as

$$\min \omega. \tag{2.3}$$

The cycle time ω is the time in which all vertex operations complete. In the case of Figure 2.6, it would be the time when a robot reaches the point called final. In general, we would be interested in cases when one robot can immediately restart its cycle (an example of this problem can be found in the work of Šůcha et al. [19]). However, this cannot be easily implemented in Process Simulate. For that reason, we define a cycle time as the time between the start of the first vertex operation and end of the last vertex operation.

Before we can formally define the energy consumption criterion, we need to have a look at the Energy Profile [8].

2.1.1 Energy Profile

Energy Profile is a function of time, and its value is consumed energy. The time represents how much time the robot needed to complete the movement of a vertex. Since each vertex has its own Energy Profile, we will denote Energy Profile of vertex v_i as E_i^M, where M denotes the fact that function represents energy consumed during the movement phase of vertex v_i.

$$E_i^M(t) = \sum_{k=a}^{b} q_k t^k, q_k \in \Re, \forall v_i \in V \tag{2.4}$$

The function is a Laurent polynomial in a field of real numbers which differs from regular polynomials by allowing negative degrees. For simplicity, a Laurent polynomial will be called just a polynomial in this thesis. The function is a sum of products of coefficients q_k and indeterminants t raised to the k-th power. The number a denotes the lowest degree of the polynomial, whereas the number b denotes the highest degree.

In some cases, the function can be a constant or a linear function, but usually, it is a polynomial where the lowest degree is negative. The function can be constant if the limit of the joint speed has a zero effect on vertex duration, i.e. when the robot accelerates only to small speed. The function could be linear, e.g. for some lower limits of the joint speed. Using this function, we can describe the total energy consumed during robot movements as follows.

$$E_{total}^M(\vec{dm}) = \sum_{v_i \in V} E_i^M(dm_i) \tag{2.5}$$

In this equation, vector $\vec{dm}$ represents durations of all vertices; the function value $E_i^M(dm_i)$ is a value of Energy Profile of vertex v_i when the argument of the function dm_i is the vertex duration.

It is important to realise that robots consume energy even when they are not moving. Moreover, wait consumption depends on the robot position. Therefore we must differ between wait consumption during prior waiting and post-waiting or complementary waiting. Also, we assume that functions describing wait consumption are linear. These consumptions of vertex v_i can be described as

$$E_i^N(dn_i) = P_{N_i} dn_i, \forall v_i \in V \tag{2.6}$$

$$E_i^W(dw_i + dc_i) = P_{W_i}(dw_i + dc_i), \forall v_i \in V. \tag{2.7}$$

The constants P_{W_i} and P_{N_i} are power coefficients. The first equation covers the consumption during the prior wait. The other one shows consumption during wait

after a vertex completed its movement. To get the total wait consumption of one vertex, we need to sum up these two consumptions. Let $\overline{W}$ denote all waits associated with a vertex.

$$E_i^{\overline{W}} = E_i^N(dn_i) + E_i^W(dw_i + dc_i), \forall v_i \in V \tag{2.8}$$

For obtaining total wait consumption, it is necessary to sum up wait consumptions of all vertices.

$$E_{total}^{\overline{W}} = \sum_{v_i \in V} E_i^{\overline{W}} \tag{2.9}$$

Finally, we can devise the optimisation criterion as a minimisation of a sum of (2.5) and (2.9).

$$\min_{\vec{dn},\vec{dm},\vec{dw},\vec{dc}} E_{total}^M(\vec{dm}) + E_{total}^{\overline{W}} \tag{2.10}$$

In Figure 2.4, we can see the Energy Profile of 'via0,via1,via2,via3,final" vertex from Figure 2.5. The x-axis shows vertex duration in seconds; the y-axis shows energy consumption in Joules. The yellow dots show measured vertex duration and energy consumption. It was measured using joint speed (or more precisely the limit of thereof) set to 100 %, 77.5 %, 55 %, 32.5 % and 10 %. Finally, in the green, there is the interpolated polynomial curve.

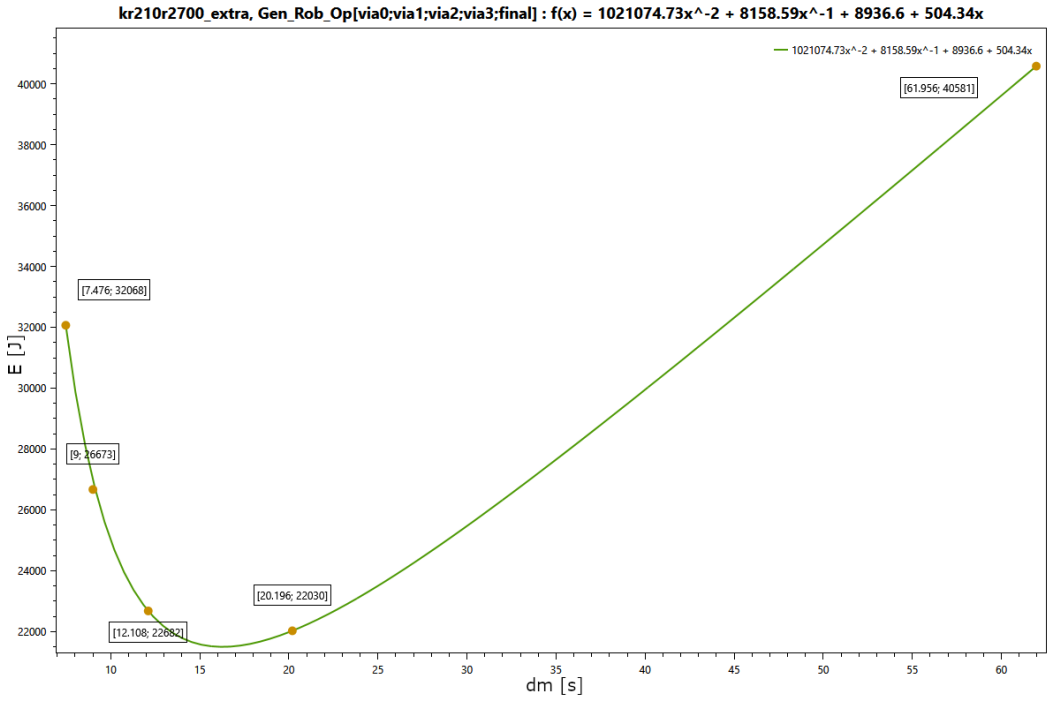

Figure 2.4. Energy profile of the "via0,via1,via2,via3,final" vertex from the Figure 2.5.

One of the goals of this thesis is to find an estimator of this Energy Profile; more about this can be found in Chapter 4.

2.2 Examples

In Figure 2.5, we can see the Path Editor tool in Process Simulate. This tool serves for creating a path a robot should undergo. The corresponding trajectory, i.e. when and how the robot approaches the points, is generated by Process Simulate. Our point of interest in this tool is mainly the zone column, which specifies how the robot should attain the space point given by the X, Y and Z columns. And then the joint speed column which limits the speed the robot can achieve. The units for the speed are percents of the maximum speed.

Paths & Locations	Attachment	X	Y	Z	RX	RY	RZ	Motion Type	Joint Speed	Zone	Duration
Seq1_diplomka											7.73
Gen_Rob_Op									0.00		7.73
HOME		-0.00	-1765.00	3084.00	90.00	90.00	0.00	Joint	100.00	fine	0.25
via0		157.19	822.17	3808.76	0.00	0.00	0.00	Joint	100.00	coarse	0.01
via1		-561.58	598.35	4268.44	0.00	0.00	0.00	Joint	100.00	coarse	4.34
via2		-447.05	-1246.06	3026.54	0.00	0.00	0.00	Joint	100.00	coarse	1.16
via3		1221.07	165.46	2982.24	0.00	0.00	0.00	Joint	100.00	coarse	1.13
final		2061.85	119.68	2986.66	0.00	0.00	0.00	Joint	100.00	fine	0.83

Figure 2.5. A screenshot of Process Simulate Path Editor tool.

In Figure 2.6, there is a graph representation of the path shown in Figure 2.5. We can see the initial dummy node, which leads to the home node. The following node groups the rest of operations since only the last one has a fine zone. The red arrow denotes the start of a new cycle.

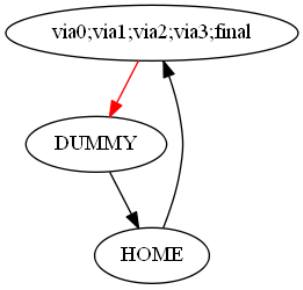

Figure 2.6. The graph representation of the compund operation defined in Figure 2.5.

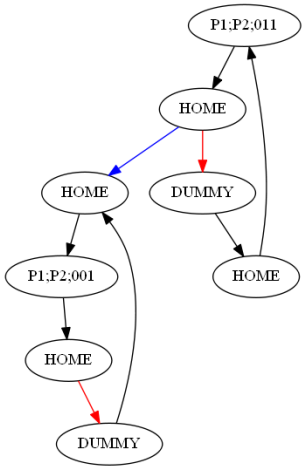

Figure 2.7. The graph representation of the compund operation defined in Figure 2.8.

There is also the Sequence Editor tool in Process Simulate. This tool allows setting time lags between compound operations of different robots as shown in Figure 2.8 (see the black arrow between the two compound operations). Figure 2.7 shows the graph representation of the compound operations defined in Figure 2.8 (the time lag constraint is denoted by the blue edge). Both editors can be used together, e.g. one can first define a path for all robots in the path editor and then specify time-lags between the robots.

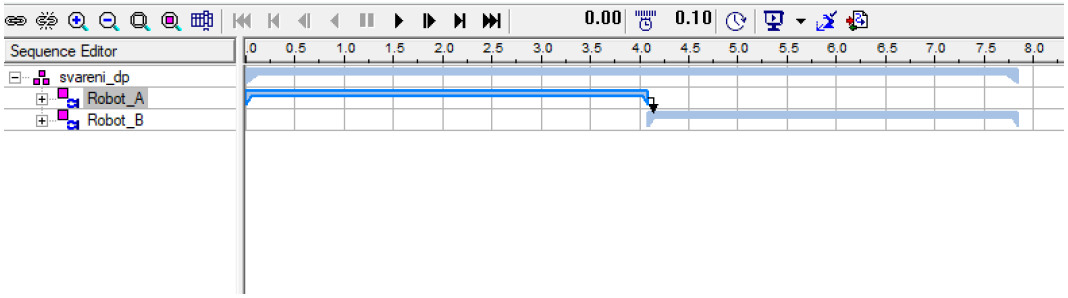

Figure 2.8. A screenshot of Process Simulate Sequence Editor tool.

Chapter 3
MILP

In this chapter, we shall construct two mixed integer linear programs. The first one is the time optimisation MILP; then we devise the energy optimisation MILP.

3.1 Time Optimisation

In this section, we derive the MILP for time optimisation. As the name suggests, we try to minimise the cycle time ω of the given robotic cell. However, if we tried to create a MILP with cycle time minimisation criterion, we would end up with non-linear constraints. For this reason, the model will not be constructed in the time domain but directly in the transformed domain. Therefore the optimisation criterion will be as follows:

$$\max \tau = \max \frac{1}{\omega} (\equiv \min \omega) \tag{3.1}$$

The variable τ shall be called inverse time cycle, i.e. production rate.

3.1.1 Constants

There are five constants in the model. The constants $\underline{dm}_i$, $\overline{dm}_i$, have values obtained from the initial simulation and represent the minimal and maximal duration of vertex v_i. These durations are measured using the maximal and the minimal upper bound on the speed of vertices, that is on the Process Simulate operations encapsulated by these vertices. The constant $\underline{dw}_i$ is a lower bound on post-wait. Its value is based on OLP wait commands. The constant ϵ ensures a minimum time gap between two colliding vertices. All these constants except for $\underline{dw}_i$ are positive real numbers whereas $\underline{dw}_i$ is a non-negative real number. The binary constant h_{ij} denotes, whether the edge (i, j) starts a new cycle or not.

$$h_{ij} = \begin{cases} 1 & \text{if the edge } (v_i, v_j) \text{ starts a new cycle} \\ 0 & \text{otherwise} \end{cases}$$

3.1.2 Variables

There is one binary variable: auxiliary variable x_i. It has the same meaning as variable x in Chapter 2.

The other variables $Dn_i, Dw_i, Dc_i, S_i, Dm_i, \tau$ are non-negative real numbers. As already mentioned, τ is the inverse cycle time. The following variables are associated with a vertex v_i. Three variables Dn_i, Dw_i, Dc_i describe the transformed prior wait, transformed post-wait and transformed complementary wait. The remaining two describe transformed start time S_i and transformed vertex duration D_i. In the rest of this section, the descriptor "transformed", which refer to the fact the model is not in the time domain, will be omitted for simplicity.

3.1.3 Constraints

Each robot has associated vertices, and we must ensure their precedence. In other words, for all edges e_{ij} containing consecutive vertices v_i, v_j it must be true that the vertex v_j starts after the vertex v_i finished. For different vertices finishing means a slightly different thing. All vertices but the dummy ones finish when their movement, post-wait and complementary wait finish. The dummy vertices do not have any movement time, but they can have a post-wait.

$$S_i + Dm_i + Dw_i + Dc_i = S_j - Dn_j + h_{ij}, \forall (v_i, v_j) \in E \tag{3.2}$$
$$S_i = S_j - Dn_j, \forall (v_i, v_j) \in E, v_i \in V_{dummy} \tag{3.3}$$
$$S_i + Dw_i = S_j - Dn_j, \forall (v_i, v_j) \in E, v_i \in V_{dummy} \tag{3.4}$$

Then, all vertices have lower and upper bounds on their duration, and also lower bounds on wait time. In the transformed domain, it looks as follows.

$$\tau \underline{dm}_i \leq Dm_i \leq \tau \overline{dm}_i, \forall v_i \in V \tag{3.5}$$
$$\tau \underline{dw}_i \leq Dw_i, \forall v_i \in V \tag{3.6}$$

The vertices which cannot have the prior wait, $V \setminus V_{priorwait}$, (conversely, the set $V_{priorwait}$ contains vertices which do have the prior wait) have the variable associated with it set to zero. For more details about the prior wait, refer to Chapter 2.

$$Dn_i = 0, \forall v_i \in V \setminus V_{priorwait} \tag{3.7}$$

All dummy vertices, V_{dummy}, start at zero. Since we are in the transformed domain, the zero has no units.

$$S_i = 0, \forall v_i \in V_{dummy} \tag{3.8}$$

The precedence between vertices of different robots or entities must be ensured. This precedence is defined by link $l_{ij} = (v_i, v_j)$, where v_i and v_j are vertices of the different robots or entities. An example of the entity would be a working table, and an example of a so-called non-robotic vertex associated with this entity would be a turn of the table. The difference between non-robotic and robotic vertices is that non-robotic vertices cannot have post-wait after they finished their movements. The vertex v_j can start only after the preceding vertex finished. In the case of non-robotic vertices, it means that the vertex finished its movement, in case of robotic vertex, there can be post-wait that needs to be finished. In addition to this, there can be an offset set on the link which shows the time which the following vertex has to wait before it can start.

$$S_i + Dm_i + Dw_i \leq S_j - Dn_j - \tau o_{ij}, \forall (v_i, v_j) \in L \tag{3.9}$$
$$S_i + Dm_i \leq S_j - Dn_j - \tau o_{ij}, \forall (v_i, v_j) \in L \tag{3.10}$$

In case a collision of two vertices v_i and v_j appeared, it needs to be resolved. The collision resolution was described in Chapter 2 in detail. Here will be just a brief reminder. The collision occurred during operation phase op_1 of vertex v_1 and operation phase op_2 of vertex v_2. To avoid this collision, we want to make sure that these operation phases are not executing at the same time; in other words, one phase must end before

the other one starts. The collision avoidance is done with the help of two inequalities shown below (they are the same as the inequalities (2.1) and (2.2) from the previous chapter).

$$t_2^i \leq t_1^j + x_k M, \tag{3.11}$$
$$t_2^j \leq t_1^i + (1 - x_k)M. \tag{3.12}$$

In the case of our model, the constant M will be set to the cycle time ω [19]. Before we proceed to look at these equations in the transformed domain, let us have a look at one of the constraint pairs in the time domain and explain the need for linearisation of the model.

$$\begin{aligned} s_i + dm_i &\leq s_j + x_k\omega - \epsilon \\ s_j + dm_j &\leq s_i + (1 - x_k)\omega - \epsilon, \quad \forall (v_i, v_j) \in CR^{M\times M} \end{aligned} \tag{3.13}$$

Here the problem lies in the multiplication of two variables $x_k\omega$. The problem can be solved by multiplying all constraints with a new variable $\tau = \frac{1}{\omega}$. This approach will, however, bring new multiplications of constraints so we must do a substitution for all such pairs. An example of the substitution would be the $S_i = s_i\tau, S_j = s_j\tau, Dm_i = dm_i\tau, Dm_j = dm_j\tau$. Now the model becomes linear.

Below there are nine pairs of constraints corresponding to each combination of operation phases $(N, M, W)^2$ in which the collision can occur. Since the set of collision resolutions can be empty, these constraints need not appear in every model.

$$\begin{aligned} S_i + Dm_i &\leq S_j + x_k - \epsilon\tau \\ S_j + Dm_j &\leq S_i + (1 - x_k) - \epsilon\tau, \\ &\forall (v_i, v_j) \in CR^{M\times M} \end{aligned} \tag{3.14}$$

$$\begin{aligned} S_i + Dm_i &\leq S_j + Dm_j + x_k - \epsilon\tau \\ S_j + Dm_j + Dw_j + Dc_j &\leq S_i + (1 - x_k) - \epsilon\tau, \\ &\forall (v_i, v_j) \in CR^{M\times W} \end{aligned} \tag{3.15}$$

$$\begin{aligned} S_j + Dm_j &\leq S_i + Dm_i + x_k - \epsilon\tau \\ S_i + Dm_i + Dw_i + Dc_i &\leq S_j + (1 - x_k) - \epsilon\tau, \\ &\forall (v_i, v_j) \in CR^{W\times M} \end{aligned} \tag{3.16}$$

$$\begin{aligned} S_i + Dm_i + Dw_i + Dc_i &\leq S_j + Dm_j + x_k - \epsilon\tau \\ S_j + Dm_j + Dw_j + Dc_j &\leq S_i + Dm_i + (1 - x_k) - \epsilon\tau, \\ &\forall (v_i, v_j) \in CR^{W\times W} \end{aligned} \tag{3.17}$$

$$\begin{aligned} S_i &\leq S_j - Dn_j + x_k - \epsilon\tau \\ S_j &\leq S_i - Dn_i + (1 - x_k) - \epsilon\tau, \\ &\forall (v_i, v_j) \in CR^{N\times N} \end{aligned} \tag{3.18}$$

$$\begin{aligned} S_i &\leq S_j + x_k - \epsilon\tau \\ S_j + D_j &\leq S_i - Dn_i + (1 - x_k) - \epsilon\tau, \\ &\forall (v_i, v_j) \in CR^{N\times M} \end{aligned} \tag{3.19}$$

$$S_i \leq S_j + Dm_j + x_k - \epsilon\tau$$

$$S_j + Dm_j + Dw_j + Dc_j \leq S_i - Dn_i + (1 - x_k) - \epsilon\tau, \quad \forall (v_i, v_j) \in CR^{N \times W} \tag{3.20}$$

$$S_j \leq S_i + x_k - \epsilon\tau$$
$$S_i + Dm_i \leq S_j - Dn_j + (1 - x_k) - \epsilon\tau, \quad \forall (v_i, v_j) \in CR^{M \times N} \tag{3.21}$$

$$S_j \leq S_i + D_i + x_k - \epsilon\tau$$
$$S_i + Dm_i + Dw_i + Dc_i \leq S_j - Dn_j + (1 - x_k) - \epsilon\tau, \quad \forall (v_i, v_j) \in CR^{W \times N} \tag{3.22}$$

The set $CR^{op_i \times op_j} \subseteq CR$ is a subset of the set of the collision resolutions which contains only collisions occurring in operation phase op_i of vertex v_i and operation phase op_j of vertex v_j.

3.1.4 Model

This subsection shows the complete model for time optimisation as devised above.

$$\max \tau \tag{3.1}$$

s.t.

$$S_i + Dm_i + Dw_i + Dc_i = S_j - Dn_j + h_{ij}, \forall (v_i, v_j) \in E \tag{3.2}$$
$$S_i = S_j - Dn_j, \forall (v_i, v_j) \in E, v_i \in V_{dummy} \tag{3.3}$$
$$S_i + Dw_i = S_j - Dn_j, \forall (v_i, v_j) \in E, v_i \in V_{dummy} \tag{3.4}$$

$$\tau \underline{dm}_i \leq Dm_i \leq \tau \overline{dm}_i, \forall v_i \in V \tag{3.5}$$
$$\tau \underline{dw}_i \leq Dw_i, \forall v_i \in V \tag{3.6}$$
$$Dn_i = 0, \forall v_i \in V \setminus V_{priorwait} \tag{3.7}$$
$$S_i = 0, \forall v_i \in V_{dummy} \tag{3.8}$$

$$S_i + Dm_i + Dw_i \leq S_j - Dn_j - \tau o_{ij}, \forall (v_i, v_j) \in L \tag{3.9}$$
$$S_i + Dm_i \leq S_j - Dn_j - \tau o_{ij}, \forall (v_i, v_j) \in L \tag{3.10}$$

$$S_i + Dm_i \leq S_j + x_k - \epsilon\tau$$
$$S_j + Dm_j \leq S_i + (1 - x_k) - \epsilon\tau, \quad \forall (v_i, v_j) \in CR^{M \times M} \tag{3.14}$$

$$S_i + Dm_i \leq S_j + Dm_j + x_k - \epsilon\tau$$
$$S_j + Dm_j + Dw_j + Dc_j \leq S_i + (1 - x_k) - \epsilon\tau, \quad \forall (v_i, v_j) \in CR^{M \times W} \tag{3.15}$$

$$S_j + Dm_j \leq S_i + Dm_i + x_k - \epsilon\tau$$
$$S_i + Dm_i + Dw_i + Dc_i \leq S_j + (1 - x_k) - \epsilon\tau, \quad \forall (v_i, v_j) \in CR^{W \times M} \tag{3.16}$$

$$S_i + Dm_i + Dw_i + Dc_i \leq S_j + Dm_j + x_k - \epsilon\tau$$
$$S_j + Dm_j + Dw_j + Dc_j \leq S_i + Dm_i + (1 - x_k) - \epsilon\tau,$$

$$\forall (v_i, v_j) \in CR^{W \times W} \quad (3.17)$$

$$S_i \leq S_j - Dn_j + x_k - \epsilon\tau$$
$$S_j \leq S_i - Dn_i + (1 - x_k) - \epsilon\tau,$$
$$\forall (v_i, v_j) \in CR^{N \times N} \quad (3.18)$$

$$S_i \leq S_j + x_k - \epsilon\tau$$
$$S_j + D_j \leq S_i - Dn_i + (1 - x_k) - \epsilon\tau,$$
$$\forall (v_i, v_j) \in CR^{N \times M} \quad (3.19)$$

$$S_i \leq S_j + Dm_j + x_k - \epsilon\tau$$
$$S_j + Dm_j + Dw_j + Dc_j \leq S_i - Dn_i + (1 - x_k) - \epsilon\tau,$$
$$\forall (v_i, v_j) \in CR^{N \times W} \quad (3.20)$$

$$S_j \leq S_i + x_k - \epsilon\tau$$
$$S_i + Dm_i \leq S_j - Dn_j + (1 - x_k) - \epsilon\tau,$$
$$\forall (v_i, v_j) \in CR^{M \times N} \quad (3.21)$$

$$S_j \leq S_i + D_i + x_k - \epsilon\tau$$
$$S_i + Dm_i + Dw_i + Dc_i \leq S_j - Dn_j + (1 - x_k) - \epsilon\tau,$$
$$\forall (v_i, v_j) \in CR^{W \times N} \quad (3.22)$$

$$Dn_i, Dw_i, Dc_i, S_i, Dm_i, \tau \in \Re_0^+, \forall i \in V \quad (3.23)$$

$$x_i \in \{0, 1\} \quad (3.24)$$

$$\underline{dm_i}, \overline{dm_i}, \underline{dw_i}, \epsilon, h_{ij} \in const. \quad (3.25)$$

3.2 Energy Optimisation

This section deals with the derivation of the MILP for energy optimisation. It is derived similarly as in the previous section with a few small differences. The biggest one is a different optimisation criterion. Another difference is that we do not need to linearise the original model, because, in the model, there is no variable multiplication. In the time optimisation, there was a problem with a multiplication of $x_k\omega$, but here ω will be constant.

The optimisation criterion as described in Chapter 2 has the following form:

$$\min_{\vec{dn}, \vec{dm}, \vec{dw}, \vec{dc}} \sum_{v_i \in V} P_{N_i} dn_i + P_{W_i}(dw_i + dc_i) + E_i^M(dm_i). \quad (3.26)$$

Since we cannot have a non-linear element in the optimisation criterion of a MILP, we have to replace the non-linear part $E_i^M(dm_i)$ with a new variable E_i. The criterion will then take the following form:

$$\min_{\vec{dn}, \vec{dm}, \vec{dw}, \vec{dc}} \sum_{v_i \in V} P_{N_i} dn_i + P_{W_i}(dw_i + dc_i) + E_i. \quad (3.27)$$

It is the minimisation of total energy consumption by all robots. The energy consumption is computed as a sum of total energy consumed during movements of all vertices and total energy consumed during all idle periods.

3.2.1 Constants

We can differentiate three types of constants based on the way they are obtained. Firstly, ω, the cycle time which we try to achieve is input to the algorithm. It can be either defined by the user or set to the result of the time optimisation. Then there are constants obtained during initial simulations. The constants $\underline{dm}_i$ and $\overline{dm}_i$ are the minimal and the maximal duration of the operations associated with vertex v_i; these two values are measured with the speed upper bound of the vertex v_i set to the minimum and maximal value respectively. The minimal and maximal value of the speed upper bound is input to the algorithm. Similarly, $\underline{dw}_i$ is the minimal post-wait associated with vertex v_i; it is obtained by reading OLP wait commands.

The constants P_{N_i} and P_{W_i} are obtained during simulation in the following way. For the former one, we take energy consumed during the first time interval of the corresponding vertex v_i and divide it with the length of the interval. The latter constant is obtained similarly, but not from the first time interval but the last one. This time interval can be set in Process Simulate, and it can be seen as a sampling interval at which we measure the energy consumed. The value of these constants could be measured during extra simulation with artificial waits added. However, the simulation is usually computationally expensive. For that reason, we prefer the first option. One may ask why it is a valid approach. The reason for this is that for small speeds (e.g. during initial acceleration or final deceleration), the curve of consumption is similar to a linear function as can be seen in Figure 2.4. Another way to look at is that during small speeds, most power is used to keep the robot in position and less power is used to move the robot.

Then there is a constant denoting whether the edge (v_i, v_j) starts a new cycle.

$$h_{ij} = \begin{cases} 1 & \text{if the edge } (v_i,\, v_j) \text{ starts a new cycle} \\ 0 & \text{otherwise} \end{cases}$$

The constant ϵ ensures a minimum time gap between two colliding vertices.

Finally, there are also constants associated with the energy profile, namely, slope $slope_{it}$ and offset $offset_{it}$ (not to be confused with link offset o_{ij}). These two constants are parameters of the lower bound line approximating the energy profile, and they are obtained during initial computation.

3.2.2 Variables

There is a binary variable x_i with the same meaning as in the time optimisation. Then there are non-negative real variables $E_i, dn_i, dw_i, dc_i, s_i, dm_i$. In all these variables the index i refers to the vertex v_i. The variable E_i denotes approximated energy consumption. The waiting variables dn_i, dw_i, dc_i denote prior wait, post-wait and complementary wait respectively. The start time of the vertex is s_i, and its duration is dm_i

3.2.3 Constraints

Firstly, we need to describe the constraints that ensure the order of the vertices of one robot. For all consecutive vertices of one robot represented by the edge $e_{ij} = (v_i, v_j)$ except for those where v_i is a dummy vertex, it must hold that vertex v_j starts only after the vertex v_i finished, i.e. after it completed its movement and did its mandatory post-wait dw_i. The role of the variables dc and dn was explained in Chapter 2. If the edge e_{ij} starts a new cycle, then $h_{ij} = 1$ and it adds the cycle time ω to the start time of vertex v_j. This condition can be written as follows:

$$s_i + dm_i + dw_i + dc_i = s_j - dn_j + h_{ij}\omega, \forall (v_i, v_j) \in E \tag{3.28}$$

If v_i is a dummy vertex, the condition is quite similar. The difference is that a dummy vertex has zero duration and never has a complementary wait. However, it can have a post-wait dw_i. Therefore there can be two different constraints.

$$s_i = s_j - dn_j, \forall (v_i, v_j) \in E, v_i \in V_{dummy} \tag{3.29}$$
$$s_i + dw_i = s_j - dn_j, \forall (v_i, v_j) \in E, v_i \in V_{dummy} \tag{3.30}$$

All vertices have lower and upper bound on their durations.

$$\underline{dm_i} \leq dm_i \leq \overline{dm_i}, \forall v_i \in V \tag{3.31}$$

Post-wait for all vertices has also a lower bound.

$$\underline{dw_i} \leq dw_i, \forall v_i \in V \tag{3.32}$$

The vertices, $V \setminus V_{priorwait}$, that cannot have a prior wait have this variable set to zero. (For more details refer to Chapter 2.)

$$dn_i = 0, \forall v_i \in V \setminus V_{priorwait} \tag{3.33}$$

All dummy vertices, V_{dummy}, have to start at 0 s.

$$s_i = 0, \forall v_i \in V_{dummy} \tag{3.34}$$

The precedence between vertices of different robots or entities must be ensured. This precedence is derived analogously to the previous section.

$$s_i + dm_i + dw_i \leq s_j - dn_j - o_{ij}, \forall (v_i, v_j) \in L \tag{3.35}$$
$$s_i + dm_i \leq s_j - dn_j - o_{ij}, \forall (v_i, v_j) \in L \tag{3.36}$$

Since we replaced the energy profile $E_i^M(dm_i)$ in the criterion with a variable E_i, we need to introduce a new constraint that will tie these two things together. Once again, we cannot use the energy profile directly because we are creating a MILP. We need to make an approximation of the energy profile.

The energy profile is approximated by linear functions which create its lower bound as shown in Figure 3.1. We will assume that the energy profile is always convex. Each linear function is defined by its offset $offset_{it}$ and slope $slope_{it}$, and it is found as a tangent to the polynomial curve in certain points. A tangent t from the set of tangents T can be described as $t : y = slope_{it}x + offset_{it}$.

$$offset_{it} \leq E_i + slope_{it} dm_i, \forall v_i \in V, \forall t \in T \tag{3.37}$$

The collision of two vertices was already explained in Chapter 2 and Section 3.1.3. Here again, there will be nine pairs of constraints corresponding to each possible pair of operation phases $(N, M, W)^2$ in which the collision can occur. The constant M will be again set to the value of the desired cycle time ω.

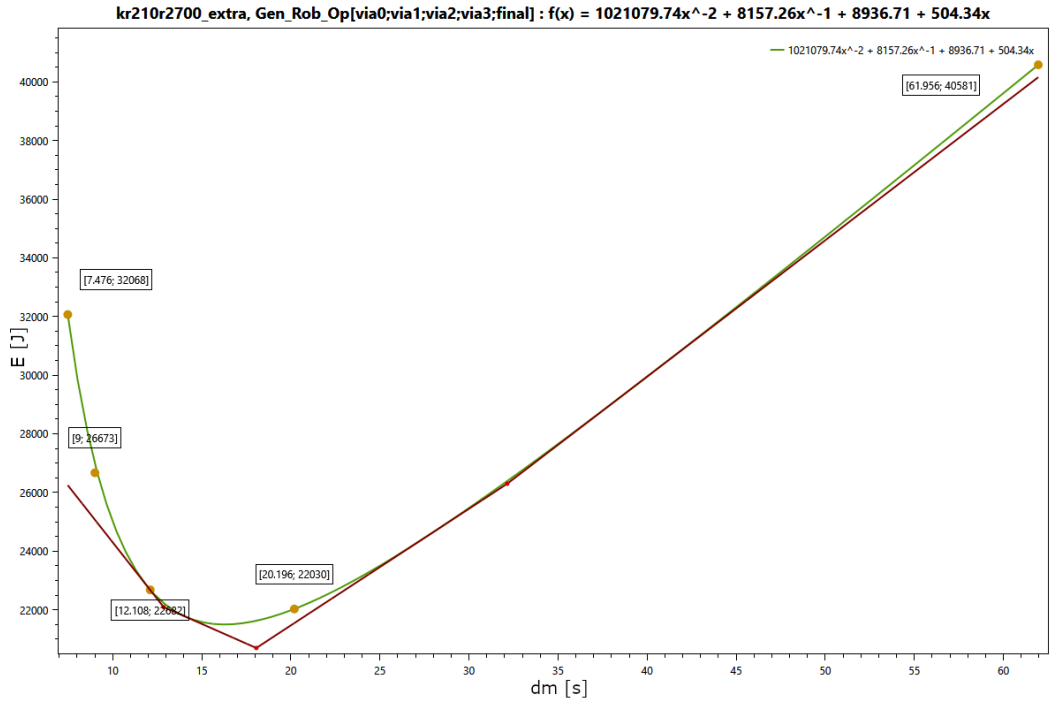

Figure 3.1. Energy profile of the "via0,via1,via2,via3,final" vertex from Figure 2.5 approximated with linear functions.

$$s_i + dm_i \leq s_j + x_k\omega - \epsilon$$
$$s_j + dm_j \leq s_i + (1 - x_k)\omega - \epsilon, \forall (v_i, v_j) \in CR^{M\times M} \quad (3.38)$$
$$s_i + dm_i \leq s_j + dm_j + x_k\omega - \epsilon$$
$$s_j + dm_j + dw_j + dc_j \leq s_i + (1 - x_k)\omega - \epsilon, \forall (v_i, v_j) \in CR^{M\times W} \quad (3.39)$$
$$s_j + dm_j \leq s_i + dm_i + x_k\omega - \epsilon$$
$$s_i + dm_i + dw_i + dc_i \leq s_j + (1 - x_k)\omega - \epsilon, \forall (v_i, v_j) \in CR^{W\times M} \quad (3.40)$$
$$s_i + dm_i + dw_i + dc_i \leq s_j + dm_j + x_k\omega - \epsilon$$
$$s_j + dm_j + dw_j + dc_j \leq s_i + dm_i + (1 - x_k)\omega - \epsilon, \forall (v_i, v_j) \in CR^{W\times W} \quad (3.41)$$
$$s_i \leq s_j - dn_j + x_k\omega - \epsilon$$
$$s_j \leq s_i - dn_i + (1 - x_k)\omega - \epsilon, \forall (v_i, v_j) \in CR^{N\times N} \quad (3.42)$$
$$s_i \leq s_j + x_k\omega - \epsilon$$
$$s_j + dj \leq s_i - dn_i + (1 - x_k)\omega - \epsilon, \forall (v_i, v_j) \in CR^{N\times M} \quad (3.43)$$
$$s_i \leq s_j + dm_j + x_k\omega - \epsilon$$
$$s_j + dm_j + dw_j + dc_j \leq s_i - dn_i + (1 - x_k)\omega - \epsilon, \forall (v_i, v_j) \in CR^{N\times W} \quad (3.44)$$
$$s_j \leq s_i + x_k\omega - \epsilon$$
$$s_i + dm_i \leq s_j - dn_j + (1 - x_k)\omega - \epsilon,$$

$$\forall (v_i, v_j) \in CR^{M \times N} \quad (3.45)$$

$$s_j \leq s_i + d_i + x_k\omega - \epsilon$$
$$s_i + dm_i + dw_i + dc_i \leq s_j - dn_j + (1 - x_k)\omega - \epsilon,$$
$$\forall (v_i, v_j) \in CR^{W \times N} \quad (3.46)$$

The set $CR^{op_i \times op_j} \subseteq CR$ is a subset of the set of the collision resolutions which contains only collisions occurring in operation phase op_i of vertex v_i and operation phase op_j of vertex v_j.

3.2.4 Model

This subsection shows the complete model for energy optimisation as devised above.

$$\min_{\vec{dn}, \vec{dm}, \vec{dw}, \vec{dc}} \sum_{v_i \in V} P_{N_i} dn_i + P_{W_i}(dw_i + dc_i) + E_i. \quad (3.27)$$

s.t.

$$s_i + dm_i + dw_i + dc_i = s_j - dn_j + h_{ij}\omega, \forall (v_i, v_j) \in E \quad (3.28)$$
$$s_i = s_j - dn_j, \forall (v_i, v_j) \in E, v_i \in V_{dummy} \quad (3.29)$$
$$s_i + dw_i = s_j - dn_j, \forall (v_i, v_j) \in E, v_i \in V_{dummy} \quad (3.30)$$

$$\underline{dm_i} \leq dm_i \leq \overline{dm_i}, \forall v_i \in V \quad (3.31)$$
$$\underline{dw_i} \leq dw_i, \forall v_i \in V \quad (3.32)$$

$$dn_i = 0, \forall v_i \in V \setminus V_{priorwait} \quad (3.33)$$
$$s_i = 0, \forall v_i \in V_{dummy} \quad (3.34)$$

$$s_i + dm_i + dw_i \leq s_j - dn_j - o_{ij}, \forall (v_i, v_j) \in L \quad (3.35)$$
$$s_i + dm_i \leq s_j - dn_j - o_{ij}, \forall (v_i, v_j) \in L \quad (3.36)$$

$$offset_{it} \leq E_i + slope_{it} dm_i, \forall v_i \in V, \forall t \in T \quad (3.37)$$

$$s_i + dm_i \leq s_j + x_k\omega - \epsilon$$
$$s_j + dm_j \leq s_i + (1 - x_k)\omega - \epsilon, \forall (v_i, v_j) \in CR^{M \times M} \quad (3.38)$$
$$s_i + dm_i \leq s_j + dm_j + x_k\omega - \epsilon$$
$$s_j + dm_j + dw_j + dc_j \leq s_i + (1 - x_k)\omega - \epsilon, \forall (v_i, v_j) \in CR^{M \times W} \quad (3.39)$$
$$s_j + dm_j \leq s_i + dm_i + x_k\omega - \epsilon$$
$$s_i + dm_i + dw_i + dc_i \leq s_j + (1 - x_k)\omega - \epsilon, \forall (v_i, v_j) \in CR^{W \times M} \quad (3.40)$$
$$s_i + dm_i + dw_i + dc_i \leq s_j + dm_j + x_k\omega - \epsilon$$
$$s_j + dm_j + dw_j + dc_j \leq s_i + dm_i + (1 - x_k)\omega - \epsilon,$$
$$\forall (v_i, v_j) \in CR^{W \times W} \quad (3.41)$$
$$s_i \leq s_j - dn_j + x_k\omega - \epsilon$$
$$s_j \leq s_i - dn_i + (1 - x_k)\omega - \epsilon,$$
$$\forall (v_i, v_j) \in CR^{N \times N} \quad (3.42)$$
$$s_i \leq s_j + x_k\omega - \epsilon$$

$$s_j + dj \leq s_i - dn_i + (1 - x_k)\omega - \epsilon, \quad \forall (v_i, v_j) \in CR^{N \times M} \tag{3.43}$$

$$s_i \leq s_j + dm_j + x_k\omega - \epsilon$$
$$s_j + dm_j + dw_j + dc_j \leq s_i - dn_i + (1 - x_k)\omega - \epsilon, \quad \forall (v_i, v_j) \in CR^{N \times W} \tag{3.44}$$

$$s_j \leq s_i + x_k\omega - \epsilon$$
$$s_i + dm_i \leq s_j - dn_j + (1 - x_k)\omega - \epsilon, \quad \forall (v_i, v_j) \in CR^{M \times N} \tag{3.45}$$

$$s_j \leq s_i + d_i + x_k\omega - \epsilon$$
$$s_i + dm_i + dw_i + dc_i \leq s_j - dn_j + (1 - x_k)\omega - \epsilon, \quad \forall (v_i, v_j) \in CR^{W \times N} \tag{3.46}$$

$$E_i, dn_i, dw_i, dc_i, s_i, dm_i \in \Re_0^+, \forall v_i \in V \tag{3.47}$$

$$x_i \in \{0, 1\} \tag{3.48}$$

$$E_i^{const}, \underline{dm_i}, \overline{dm_i}, \underline{dw_i}, \omega, \epsilon, slope_{it}, offset_{it}, h_{ij}, P_{N_i}, P_{W_i} \in const. \tag{3.49}$$

Chapter 4
Energy profile approximation

One of the goals of this thesis is to come up with an approximation of the energy profile (defined in Chapter 2).

In Chapter 2, we defined the energy profile as a function describing the dependence between a vertex duration and its energy consumption. The function was obtained by interpolating the measured data. In Chapter 3, we used this function or more precisely its piecewise linear approximation as one of the inputs to the MILP for energy consumption minimisation.

This chapter discusses possible data-driven approaches to energy profile approximation.

4.1 Initial measurements

Initial measurements were done on three types of manually created trajectories called "straight", "updown" and "downup". The straight trajectory was a movement defined by the movement of the first robotic joint j_1. The movement started with the joint j_1 set to $-90°$, then the robot moved to position where $j_1 = 0°$, and finally to the position $j_1 = 90°$. In other words, the robot moved horizontally due to the changes in the first joint position. Illustration of this movement can be seen in Figure 4.1. On the left is the start position, and on the right is the final position, whereas in the middle is the coarse point around which the robot moves. The discontinuous line shows the path that the end point of the robot undergoes.

The updown and downup movements are vertical movements (one from up to down, the other going the opposite way) without the midpoint, where the robot moves the second joint j_2. Since the position limits in the second joint differ in each used robot, the movement was going from the upper limit to bottom limit or vice versa of each robot. Both movements are illustrated in Figure 4.2, where the downup movement is robot movement from the left position to the right one, and the updown movement is the opposite one (from right position to the left one). These trajectories can be seen as vertices of graph G mentioned in the previous chapters.

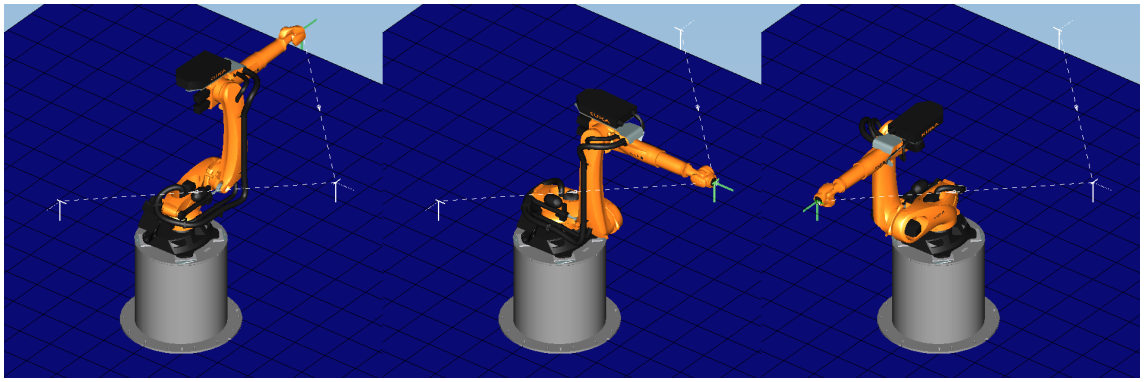

Figure 4.1. Illustration of the "straight" movement with Kuka KR 90 R3700 prime.

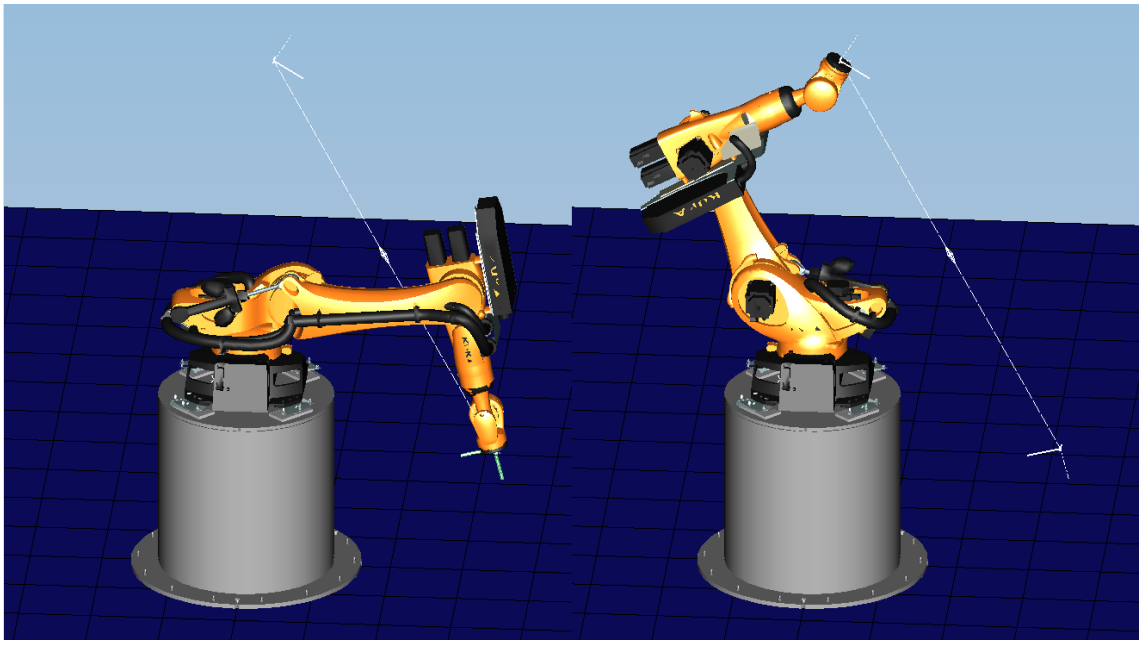

Figure 4.2. Illustration of the "downup" and "updown" movements with Kuka KR 300 R2500 ultra.

For these types of trajectories, the energy consumption was measured using Kuka-Krc controller on three Kuka robots: KR 90 R3700 prime, KR 210 R2700 extra, and KR 300 R2500 ultra. The first number in the robot name is its payload in kg; the second one is its reach in mm.

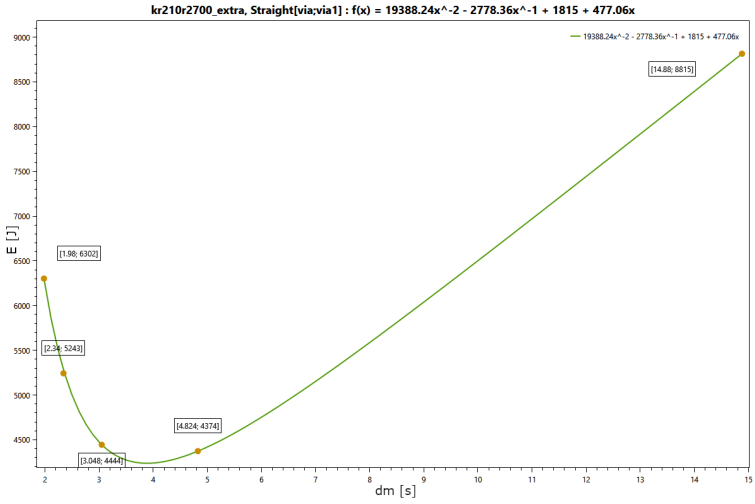

Figure 4.3. Energy profile of the "straight" vertex of Kuka KR 210 R2700 extra.

In Figures 4.3, 4.4 and 4.5, there are shown energy profiles for three different trajectories of two robots. The energy profiles were obtained by interpolating five samples representing duration and energy consumption. We can see that the minimum lies between the third and fourth sample points, which correspond to the limit of the joint speed 55 % and 32.5 %. Where exactly the minimum lies depends on the type of the function which we use to interpolate the measured data points. However, the trend is clear: going from the highest speed to the lowest the energy consumption decreases until some point and then it starts to increase again. A further illustration of this trend

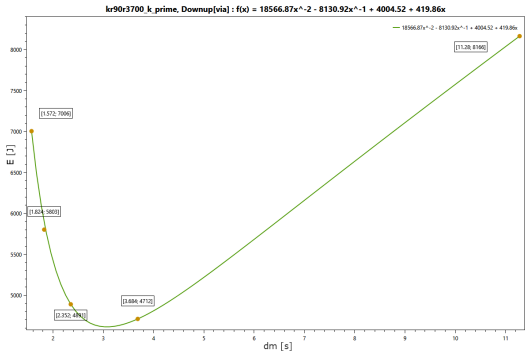

Figure 4.4. Energy profile of the "downup" vertex of Kuka KR 90 R3700 prime.

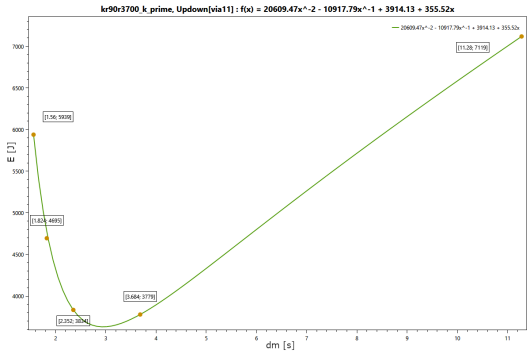

Figure 4.5. Energy profile of the "updown" vertex of Kuka KR 90 R3700 prime.

can be seen in Figure 4.9, where we explicitly plot the joint speed limits along with energy consumption. All this complies with findings of Meike et al. [15].

Since the three basic trajectories consist only of a movement in one joint, additional "complex" trajectory was made that involves a movement of all six joints. The movement is illustrated in Figure 4.7. The robot moved from the initial position shown on the left to the position on the right. Even then, the measured data shown in Figure 4.8 complies to what we would expect.

Finally, let us have a look at Figure 4.9. There are plotted measured data samples for the straight trajectory of Kuka KR 300 R2500 ultra. However, this time on the x-axis, there is the limit of the joint speed instead of duration. Please note that the imaginary curve that would interpolate the data points mirrors the curve that would interpolate the points in Figure 4.6 because the lowest speed limit corresponds to the highest duration. It is interesting to see that for some neighbourhood around the minimum increasing or decreasing the speed by the same amount produce a similar increase in energy consumption.

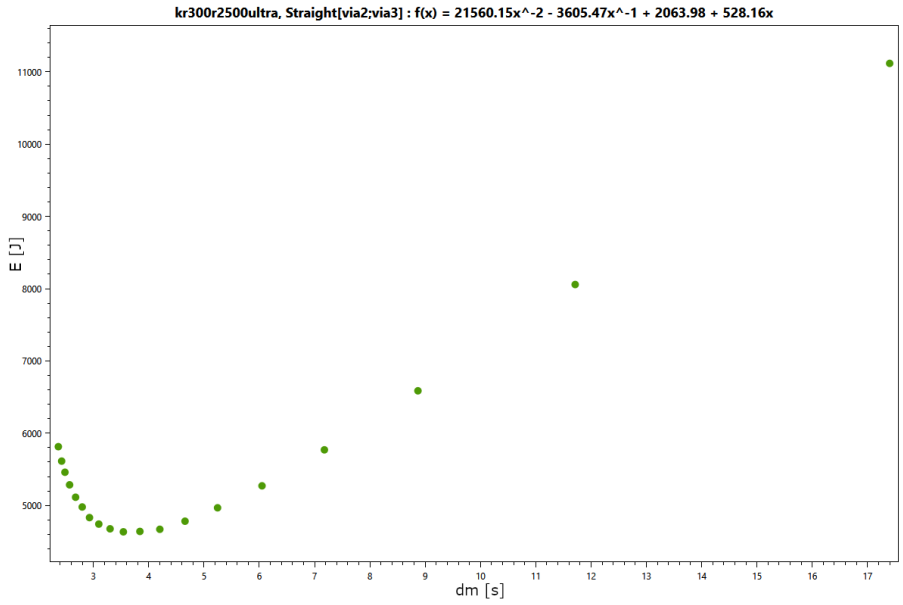

Figure 4.6. Dependence between duration and energy consumption on the straight trajectory of Kuka KR 300 R2500 ultra.

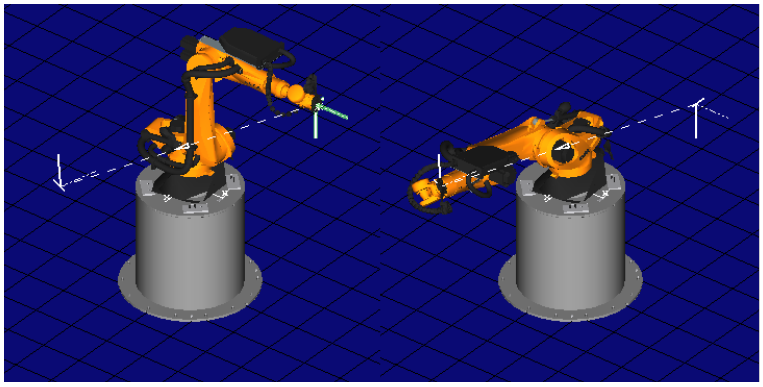

Figure 4.7. Illustration of the “complex” movement with Kuka KR 210 R2700 extra.

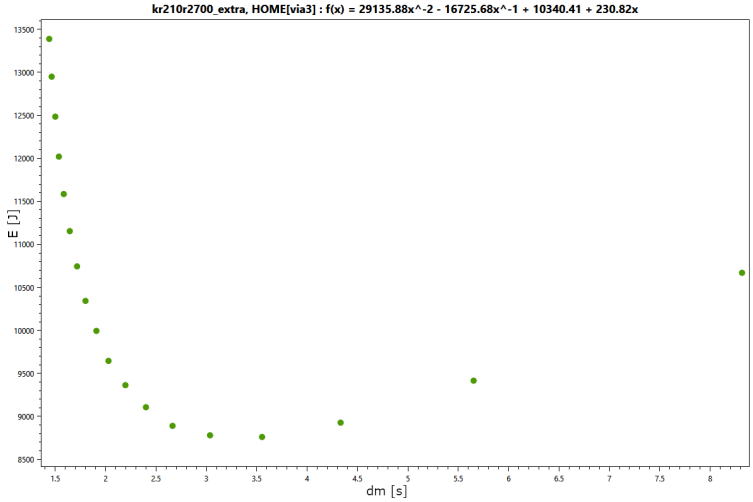

Figure 4.8. Energy consumption of the “complex” vertex of Kuka KR 210 R2700 extra.

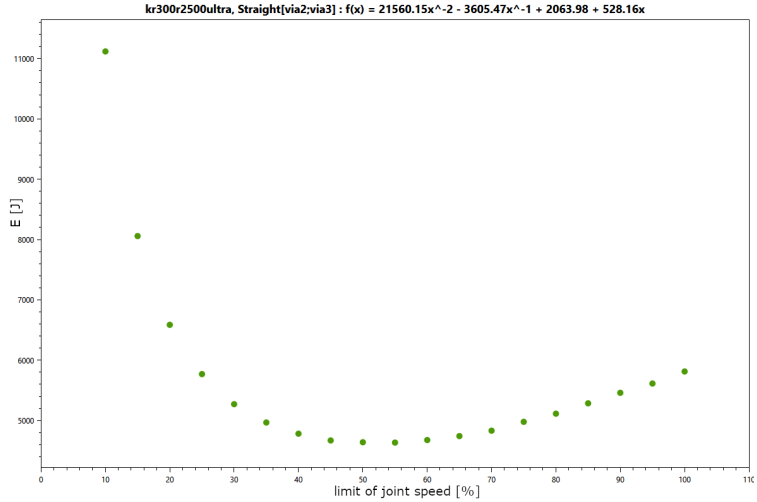

Figure 4.9. Dependence between the joint speed limit and energy consumption on the straight trajectory of Kuka KR 300 R2500 ultra.

4.2 Initial feature selection

To simplify things, what we try to achieve in this chapter is to find some input features (that could be measured via Process Simulate API with the standard controller), output features describing the estimated energy profile and the relationship between those two sets of features.

Firstly, we need to ask what interests us about the energy profile. In Chapter 3, we described how we use the approximation of the energy profile as a lower bound constraint for variables that go into the optimisation criterion. We can infer two things from that. One is that we need to have information about the height of the profile; hence, we could use the optimal energy E^* as one of the output parameters. The other thing is that we have to approximate any energy profile by linear functions since we cannot use non-linear constraints in the MILP. For that reason, we can directly estimate the piecewise linear approximation of the energy profile. For that, we will need some points which will serve as endpoints of the line segments. We could use the minimum and maximum duration (in this chapter it will be the duration of vertex with the joint speed limit set to 100 % and 10 % respectively) and their respective energy consumption as two points $(d_{min}, E_{d_{min}})$ and $(d_{max}, E_{d_{max}})$. The third point could be the duration needed to achieve the lowest consumption, (d^*, E^*). And also estimating the joint speed limit, v^* which is needed to get the duration of the least energy consuming trajectory. Let us call this triplet d^*, E^*, v^* *optimal duration*, *optimal energy* and *optimal speed* for simplicity. We can measure the minimum and maximum duration, so this leaves us with five features we will want to estimate, $(d^*, E^*, v^*, E_{d_{min}}, E_{d_{max}})$. Let the quintuple $(\hat{d}, \hat{E}, \hat{v}, \hat{E}_{d_{min}}, \hat{E}_{d_{max}})$ represent the estimated values of the output features.

Secondly, it is necessary to identify which parameters may influence the energy consumption, as well as which parameters can be obtained via Process Simulate API with the standard controller only. It is important to note that we only had one controller (that is a Process Simulate addon which allows measure energy consumption) available. This controller was Kuka-Krc; as the name suggests, it only allowed energy consumption measurements for Kuka robots, namely the robots from Quantec Series (robot series in the high payload category). The Kuka robots differed mainly in the maximum payload (varying from 90 to 300 kg) and its maximum reach (2500 to 3700 mm).

As we already saw in this chapter, the energy consumption of a vertex is influenced by its duration, so it makes sense to measure the duration with various joint speed limits set. The duration is influenced by the length of the path the robot undergo. The length could mean many things in this context; simple Euclidean distance between the points the robot should approach, angle distance made by the robot joints or even distance travelled by the Tool Centre Point Frame (TCPF). As the name suggests, the point could be at the point where a tool is attached to the robot, or at the tool itself. From the angle distance travelled by a joint, we can compute angle speed, acceleration and jerk. Jerk is the rate of change of acceleration. It is easy to imagine that fast acceleration to high speed would require more power than simple movement at constant low speed.

4.3 Data acquisition

Now that we have some notion about what features could be important, let us talk about how the features were obtained.

Because the simulations are very slow, we decided to save way more features than we planned to use as it is always easier to remove existing data than obtaining a new

one. The saved features were payload, reach and type of the robot, and speed and acceleration limits for all of its joints. We also saved the sum of Euclidean distance between given via points. These features could be obtained without the simulation. Then for five different joint speed limits (10 %, 32.5 %, 55 %, 77.5 %, 100 %) we measured the following data: duration, path length travelled by the end point of robot and for all joints we measured absolute displacement in radians, then maximum and average speed, acceleration, and jerk in $rads^{-1}, rads^{-2}$ and $rads^{-3}$ respectively. In total, there were more than two hundred features saved.

There are two main reasons why we chose the lower bound of speed limit set to 10 %. One is that the longer the duration is, the longer the already slow simulation lasts. One can see in the provided figures that lowering the speed limit to 10 % can increase the duration of vertex approximately tenfold. The other, more important reason is that for lower speed the curve resembles a linear function. So making other simulations at lower speeds would not give any useful information.

Data were obtained in two phases. In the first phase, the trajectories were generated, and in the other, the data were measured for those trajectories. The data generation was a rather complicated task since one cannot easily verify that the generated trajectory is viable. While it is possible to check that every single via-point is reachable by the robot, it is not possible to verify in advance if the robot can move along these points. For that reason, we chose to generate only the simplest paths. Those paths were just two fine points, where the first one was the robot default position, and the other was randomly generated by moving robotic joints into a feasible position.

The second phase was as if not more complicated as the first one. To obtain all the wanted data, we needed to make fourteen simulations at various speeds. Firstly, we did five basic simulations with the speed limits set to 10 %, 32.5 %, 55 %, 77.5 % and 100 % to obtain data samples with energy consumption. An example of these data samples can be seen in Figure 4.3. The yellow dots denote duration and energy consumption during the five simulations. Then we ran nine additional simulations to find the optimal energy consumption.

We assume that the curve approximating measured data changes from decreasing to increasing at some point. So we find the interval in which it happens and run the additional simulations to find the minimum energy consumption as well as the optimal duration and joint speed, E^*, d^*, v^*. The simulations were distributed evenly along the interval. Therefore, the optimal joint speed limit need not be the absolute best one. (The word "optimal" is a reference to the fact that this value is the one we will try to estimate rather than it would mean the absolute argument minimum of the energy profile.) Given the fact we do not have much data available, the fact the additional simulations are evenly distributed is useful.

The minimum was found in the following way. Assuming all energy profiles can be split into two continuous intervals the first one decreasing and the other increasing, and given five sample points we know there will be three consecutive data samples where the middle one will have lower energy consumption than the other two. Since the space between all samples has a size of 22.5 percentage points, the whole interval on which we will find the minimum will have a size of 45 p.p. Then we will make nine additional simulations with the speed limit evenly distributed within the interval. The difference in speed limit between consecutive simulations will be 4.5 p.p.

In theory, we could measure energy consumption for every possible speed limit (its precision in Process Simulate is two decimals) and thus get the absolute best. However, this is not a viable approach, as the simulation is the slowest part of the whole process.

Indeed, this phase was already very time-consuming as it was; getting the data for just one trajectory took minutes. So it was unfeasible to get a huge amount of data.

All of the things mentioned so far lead to the estimation of the energy consumed during the movement of the vertex. As we saw in the previous chapters, we also need the estimation of the energy consumed during the waits. We could try some similar approach to estimate the wait coefficient, but that would be unnecessarily complicated. Now let us imagine a situation in which a robot moves between two points extremely slowly, so slowly that within one second it makes no noticeable move. In that case, there would be effectively no difference between one second of waiting and one second of the movement. Therefore we can equal this movement energy consumption with wait energy consumption. We are going to exploit this observation later in this chapter when we talk about the implementation of the estimator.

This concept is illustrated in Figures 4.10 and 4.11. Both images show the robot viewer tool from Process Simulate, which allows displaying power consumption and total energy consumed. Both images show the power consumption for the Kuka KR 300 R 2500 ultra and the straight movement. The first picture shows the power consumption for the case when speed was set to 100 % and ten-second long wait was added at the end. The power consumption during the wait was around 0.56 kW. The second picture shows the case when speed was set to only 1 % with no wait added. The power consumption during the movement was around 0.55 kW.

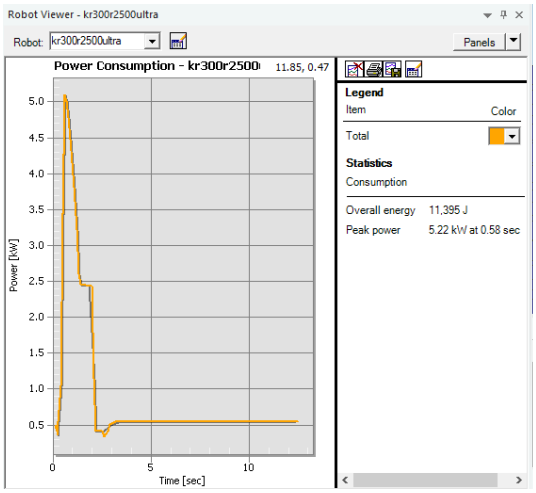

Figure 4.10. Power consumption of straight movement with speed limit set to 100 % and with 10 s wait at the end.

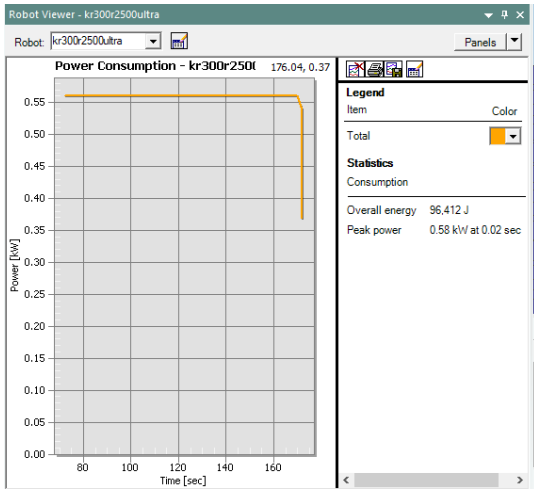

Figure 4.11. Power consumption of straight movement with speed limit set to 1 %.

4.4 Regression

Now that we have input and output features, we have to find out a relationship between them. For that, we will use regression analysis, i.e. a process of estimating the relationship between variables. In general, regression models involves three sets of elements. The first one is a set of observed inputs (predictors, input features) X; then there is a set of dependent variables (output features) Y and also the set of unknown parameters β which we will try to find. We can imagine that as some function of the input features and unknown parameters which produces the output features.

$$Y \approx f(X, \beta) \tag{4.1}$$

To be able to do the regression, we need to specify function f. In our case, we choose a linear function. Therefore the output Y will be a linear combination of input features. The linearity means that the function is linear in the unknown parameters. The function can be non-linear in the output features (e.g. it could contain elements such as x_1^2 or $x_1 x_2$). We, however, make the output features linear as well. Since we will try to predict more than one output feature given more than one input features, the regression will be called multiple multivariate linear regression. Below is an example of predicting an output feature y_j given one observation X.

$$y_j = \beta_{0,j} + \sum_{i=1}^{n} \beta_{i,j} x_i \tag{4.2}$$

The constant $\beta_{0,j}$ is also called an intercept. It is often defined as the mean of the output when all of the inputs are set to zero. If we add a dummy input feature $x_0 = 1$ for all observations, we can simplify the equation (4.2) in the following way.

$$y_j = \sum_{i=0}^{n} \beta_{i,j} x_i \tag{4.3}$$

The complete model could be written in the matrix form in the following way:

$$X\mathrm{B} = Y, X \in \Re^{o\times(n+1)}, \mathrm{B} \in \Re^{(n+1)\times j}, Y \in \Re^{o\times j}, \tag{4.4}$$

or in a more detailed way

$$\begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \dots & x_{1,n} \\ 1 & x_{2,1} & x_{2,2} & \dots & x_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{o,1} & x_{o,2} & \dots & x_{o,n} \end{pmatrix} \begin{pmatrix} \beta_{0,1} & \beta_{0,2} & \dots & \beta_{0,5} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_{n,1} & \beta_{n,2} & \dots & \beta_{n,5} \end{pmatrix} = \begin{pmatrix} y_{0,1} & y_{0,2} & \dots & y_{0,5} \\ \vdots & \vdots & \vdots & \vdots \\ y_{o,1} & y_{o,2} & \dots & y_{o,5} \end{pmatrix}. \tag{4.5}$$

The first matrix represents o observations in rows with an intercept and n input features in columns. The second matrix consists of coefficients of linear combinations of input features in the columns. In the rows, the coefficients correspond to j output features. Finally, in the third matrix, there are j output features in every row for each of o measured output features.

The regression model was written in Python with the help of machine learning library scikit-learn [20]. The input to the file was a .csv (comma-separated values) file with measured data. The data were then read and separated into inputs and outputs.

With the help of scikit-learn library, we decided to limit the number of features to thirty. All of them are shown in Table 4.1. In the upper bottom part of the table, the columns represent features associated with joints: maximal speed v_{max}, maximal acceleration a_{max}, maximal jerk j_{max}, average speed v_{avg}, average acceleration a_{avg} and average jerk j_{avg}. The rows represent the joints of the robot. If a cell is empty, the feature was not selected for the particular joint and any speed. If there is a number or numbers in the cell, it means a feature corresponding to the given joint, and the speed limit was selected. The following features were selected: Euclidean distance between the points the robot should approach; path length for speed limit set to 10 %; for the first joint: a_{max} for speed limits of 55 %, 77.5 % and 100 %, j_{max} for the same speed limits, and a_{avg} and j_{avg}, both for the speed limit of 77.5 %; for the second joint: a_{max} for the speed limit of 77.5 %, j_{max} for the speed limits of 32.5 %, 55 % and 77.5 %, and

a_{avg} for the speed limit of 55 %; for the third joint: a_{max} for the speed limit of 55 % and j_{avg} for the speed limit of 77.5%; for the fourth joint: a_{max} and j_{max} for the speed limit of 32.5 % and j_{avg} for the speed limit of 55 %; for the fifth joint: v_{max} and a_{max} for the speed limit of 100 % and j_{max} for the speed limit of 32.5 %; for the sixth joints v_{max} for the speed limits of 77.5 % and 100 %, a_{max}, j_{max}, v_{avg} and a_{avg} for the speed limit of 100 % and j_{avg} for the speed limit of 10 %.

	v_{max}	a_{max}	j_{max}	v_{avg}	a_{avg}	j_{avg}
joint 1		$\{55, 77.5, 100\}$	$\{55, 77.5, 100\}$		$\{77.5\}$	$\{77.5\}$
joint 2		$\{77.5\}$	$\{32.5, 55, 77.5\}$		$\{55\}$	
joint 3		$\{55\}$				$\{77.5\}$
joint 4		$\{32.5\}$	$\{32.5\}$			$\{55\}$
joint 5	$\{100\}$	$\{100\}$	$\{32.5\}$			
joint 6	$\{77.5, 100\}$	$\{100\}$	$\{100\}$	$\{100\}$	$\{100\}$	$\{10\}$
Other features:	euclid distance	path length ($v = 10$ %)				

Table 4.1. Selected input features.

For a comparison, another model was made where we selected only four features, that is Euclidean distance l_e, path length l_{10} and duration d_{10} for speed of 10 % and duration d_{100} for speed of 100 %. The reasoning behind a selection of these features is simple. The durations are chosen because we want to estimate the energy consumption for them. And the Euclidean distance and path length influence movement, speed and acceleration of the joints and therefore also influence energy consumption.

Linear Regression in scikit-learn uses ordinary least squares linear regression that "minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation."[21] Mathematically, it could be written in the following way:

$$\min_B ||XB - Y||_2^2. \tag{4.6}$$

For the second model we can rewrite $XB - Y$ in more detail in the following way:

$$\begin{pmatrix} 1 & l_e^1 & l_{10}^1 & d_{10}^1 & d_{100}^1 \\ 1 & l_e^2 & l_{10}^2 & d_{10}^2 & d_{100}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & l_e^o & l_{10}^o & d_{10}^o & d_{100}^o \end{pmatrix} \begin{pmatrix} \beta_{0,1} & \dots & \beta_{0,5} \\ \vdots & \vdots & \vdots \\ \beta_{4,1} & \dots & \beta_{4,5} \end{pmatrix} - \begin{pmatrix} E_{d_{min}}^1 & E_{d_{max}}^1 & E^{*1} & t^{*1} & v^{*1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ E_{d_{min}}^o & E_{d_{max}}^o & E^{*o} & t^{*o} & v^{*o} \end{pmatrix}. \tag{4.7}$$

4.5 Results and estimator implementation

Linear regression produced coefficients and intercepts for the given features which are used to get the estimated values of energy consumed during the minimal and maximal duration $\hat{E}_{d_{min}}, \hat{E}_{d_{max}}$, the minimal energy consumption $\hat{E}$, the optimal duration $\hat{d}$, and the optimal joint speed $\hat{v}$.

We decided to approximate the energy profile f_{EP} in the following way. The energy profile will be approximated by one constant segment representing the optimal estimated energy consumption, and two linear segments representing the growth of the

energy consumption to the duration extremes. The constant function will be $y_1 = \hat{E}$. To account for possible imprecisions caused by lower sample set, we chose to add a delta neighbourhood around the optimal duration with the size of $\Delta = \pm 2.5$ p.p., where the size of one percentage point was set as $\delta = \frac{(d_{max} - d_{min})}{100}$. Thus the line segment of this constant function will go from $d_1 = \hat{d} - 2.5\delta$ to $d_2 = \hat{d} + 2.5\delta$. These two points d_1 and d_2 and the value $\hat{E}$ will be then used as end points for the linear segments y_1 and y_2. The other end points will be $[d_{min}, \hat{E}_{d_{min}}]$ for y_1 and $[d_{max}, \hat{E}_{d_{max}}]$ for y_2. So we can write the estimated energy profile $\hat{f}_{EP}$ in the following way.

$$\hat{f}_{EP}(d) = \begin{cases} \frac{(\hat{E} - \hat{E}_{d_{min}})}{(d_1 - d_{min})}(d - d_{min}) + \hat{E}_{d_{min}} & d \in (d_{min},\, d_1) \\ \hat{E} & d \in (d_1, d_2) \\ \frac{(\hat{E} - \hat{E}_{d_{max}})}{(d_2 - d_{max})}(d - d_{max}) + \hat{E}_{d_{max}} & d \in (d_2, \infty) \end{cases} \tag{4.8}$$

The reason why the third segment goes to ∞ is that the user could choose to run the optimisation with smaller speeds than we measured. And while there is an upper bound on the duration, its value does not provide any valuable information so we need not try to find it using a simulation.

Please note that in the MILP we use whole functions for simplicity, but only the segments on a given interval will be of interest to us.

In Figure 4.12, we can see an example of the energy profile estimator on the previously mentioned updown trajectory. As for the wait time energy consumption estimation, we shall use the slope of the rightmost segment of the movement energy profile. We can view the segment as a change of energy consumed over time. The value of the slope is the derivation of the segment, i.e. the power consumed. To get the wait energy consumption, it is sufficient to multiply its length with the power consumed.

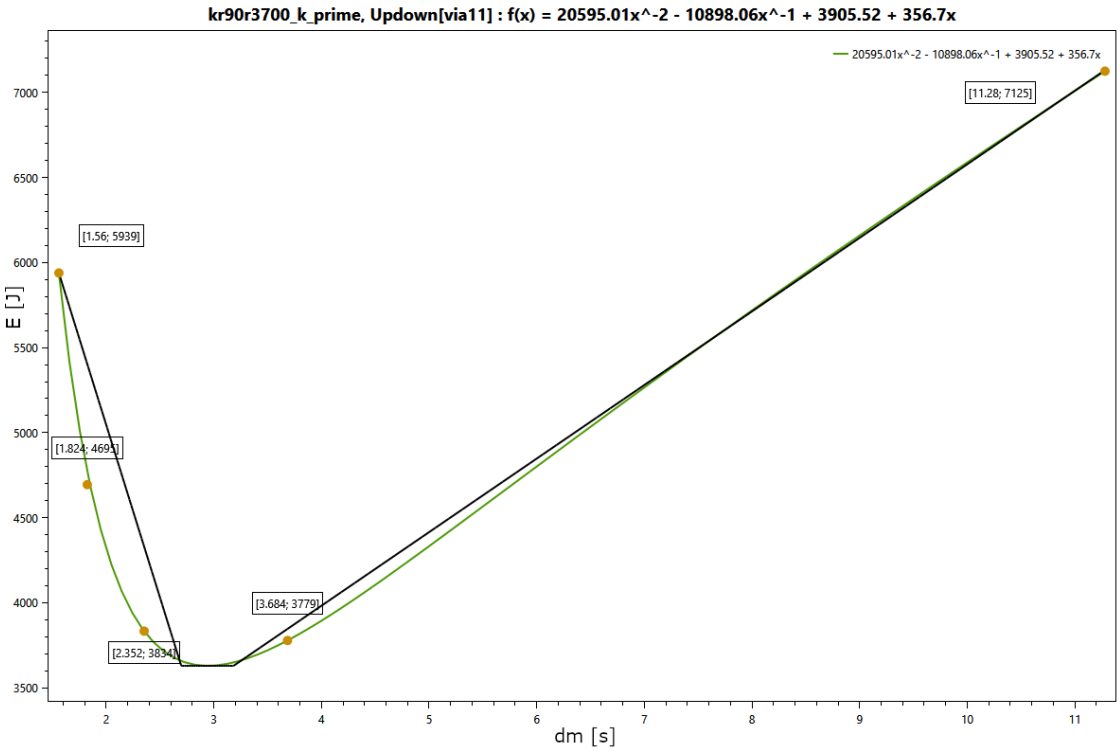

Figure 4.12. An example of the energy profile estimator.

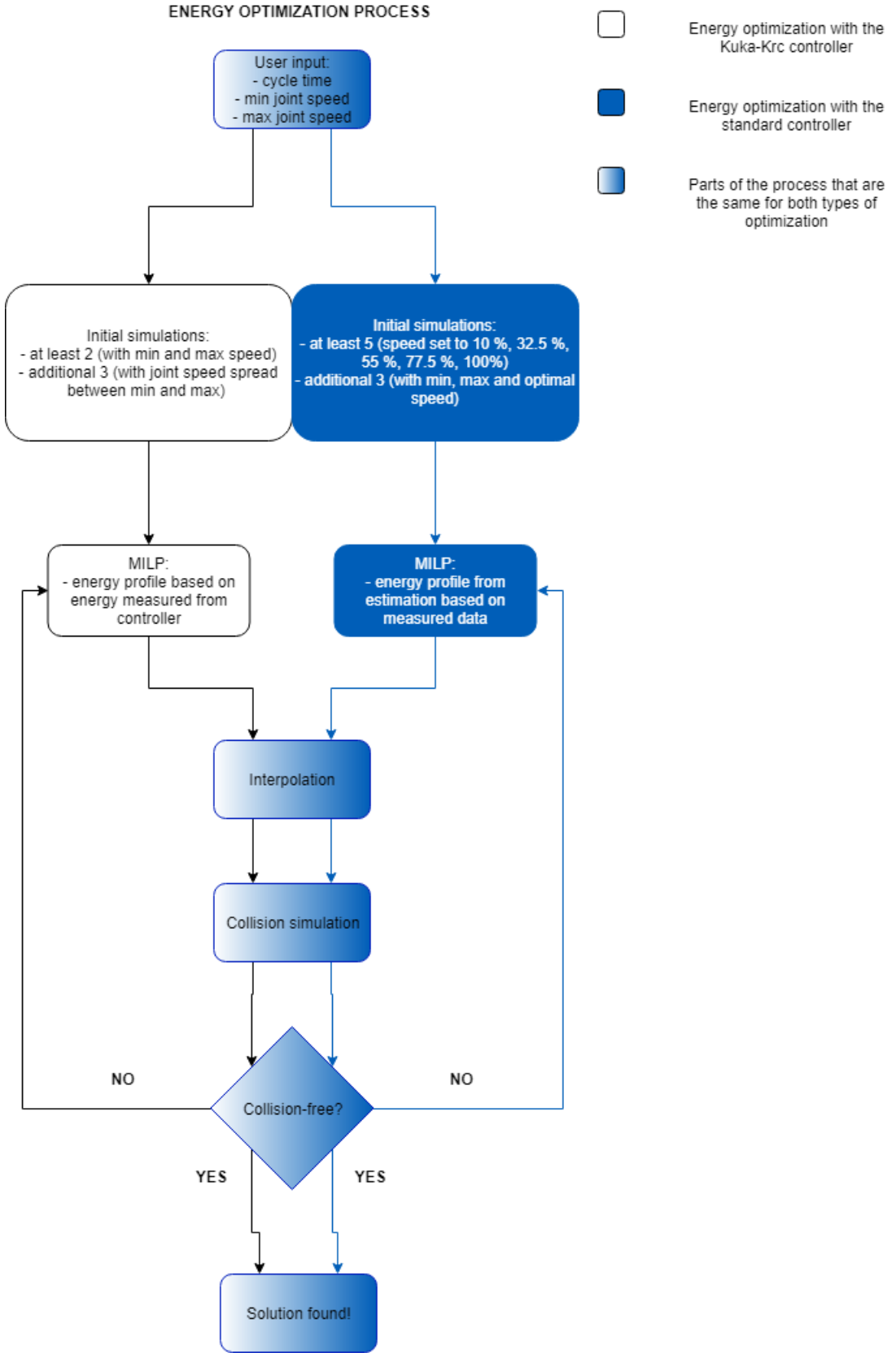

Figure 4.13. Diagram comparing energy optimisation processes.

In Figure 4.13, we can see a comparison of the two possible approaches to energy consumption optimisation on a state diagram. The first represents the case when we have a robotic controller which allows measuring energy consumption; the other represents the case where we have no controller available and have to use the estimator function.

In the state diagram, the cells in the blue-white gradient denote the parts of the algorithm, which is the same for both types of approaches. Let us described the optimisation with the controller available. Firstly, the user has to input the desired cycle time (or it can be chosen automatically as a result of time optimisation) and the minimum and maximum joint speed (please recall that this is limit of the joint speed). Then the initial simulations are done; the first two with minimum and maximum joint speed serves not only for energy consumption measurement but also for duration measurement (see the inequalities (3.31)). If the minimum and maximum joint speed are equal, no more simulations are done. If not, then additional three simulations with speeds proportionally spread between the minimum and maximum speed are done to receive more energy measurement samples.

The following phase is the MILP as defined in Section 3.2. Then we try to tie the optimisation results with simulation in the so-called interpolation phase. Finally, collision simulation is run to see if there are any collisions. If so, the collision resolution is added to the MILP, and the whole procedure repeats until a collision-free solution is found or if no feasible solution could be found.

As can be seen in the diagram, the optimisation where we have no controller available differs in two parts. First one is the section of initial simulations where we need to run five simulations with the same speeds which were used to obtain the estimator and possibly another two with the speeds set by the user to obtain duration limits for the vertices. The other part is the MILP; here, the difference lies in the way of getting the energy profile approximation. We also use the slope of the rightmost segment in the energy profile of a vertex as a coefficient of energy consumed during the waiting for a given vertex.

It was rather easy to implement the estimator within the existing framework. Firstly, functions measuring the features needed to be added. To allow possible future updates, all mentioned input features were measured, however those of them not among the thirty selected ones were given zero weight.

Then the MILP needed to be updated. For the energy consumed during waitings, it was sufficient to replace the measured values in the optimisation criterion (3.27) with the estimated ones. For the energy consumed during movements, it was necessary to only change the lower bound function (see Section 3.1.3 for details) with the estimated function. The following chapter will discuss the quality of this estimator function.

Chapter 5
Experimental results

This chapter describes the results of experiments where we tested the quality of the energy optimisation processes.

5.1 Experiment setup

All the energy consumption measurements were done in Process Simulate [22] with Kuka-Krc controller with the help of a Robot Viewer tool. It is important to note that Process Simulate measures the power consumption and consequently the energy consumption only when a robot is active, i.e. when it moves or when it waits due to some command, which is not realistic. In comparison, the MILP for both types of energy approximation considers the robots active for the whole cycle time. So to be able to produce comparable results, the artificial wait time was added to the last vertices of the inactive robots. In the following text when two values of consumption will be shown the first one will show the consumption when the artificial waits were added and the second one in the parenthesis will be the consumption of the original trajectory, i.e. provided by Process Simulate.

To get some base case, we firstly measured the energy consumption at 100 % ignoring all possible collision. Then we run time optimisation to get the lowest possible cycle time. Then we run both cycle time and energy optimisation to compare the improvement. Finally, we run only energy optimisation with some relaxed cycle time. All the tests were run with the interval of joint speed limits set from 10 % to 100 %.

All the energy optimisations were run once with consumed energy obtained from the controller and once with energy obtained from the estimator.

5.2 Results

It turned out that the regression model with thirty features, described in Section 4.4, does not provide useful results at all. The estimated energy profile was often way off from what we could reasonably expect. There might be many reasons for this behaviour. First, the features may not have been well selected in the first place. Or there were too many of them and too few input data.

On the other hand, the model with manually selected features provided much better results, as will be shown on the following pages. An example is shown in Figure 5.1. The blue curve shows the energy profile interpolating given data points (the data points can be seen along the same curve in Figure 4.3). The three segments, displayed in black as a piecewise linear function, are on the contrary obtained by the estimator function. We can see that the estimator overestimates the energy consumption for this particular example. If the estimator were to be found consistently overestimating the energy consumption, it would be good because then it would be sufficient to tune the values of the intercepts to get the correct results. However, that would require further research that would prove whether this proposition holds.

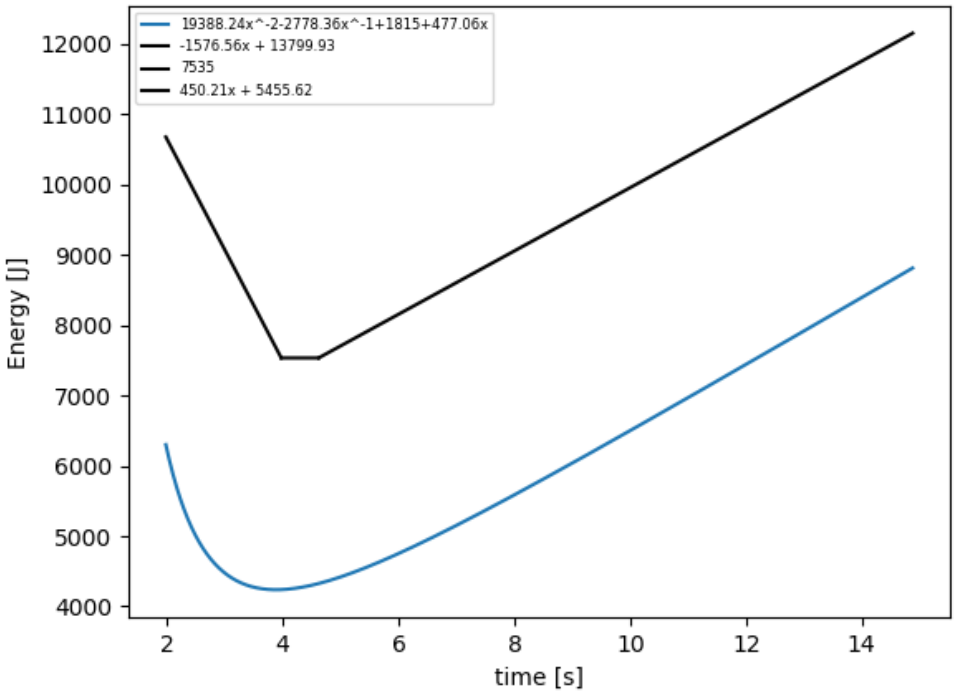

Figure 5.1. A comparison of two energy profiles for the straight vertex, defined in Section 4.1, of Kuka KR 210 R2700 extra.

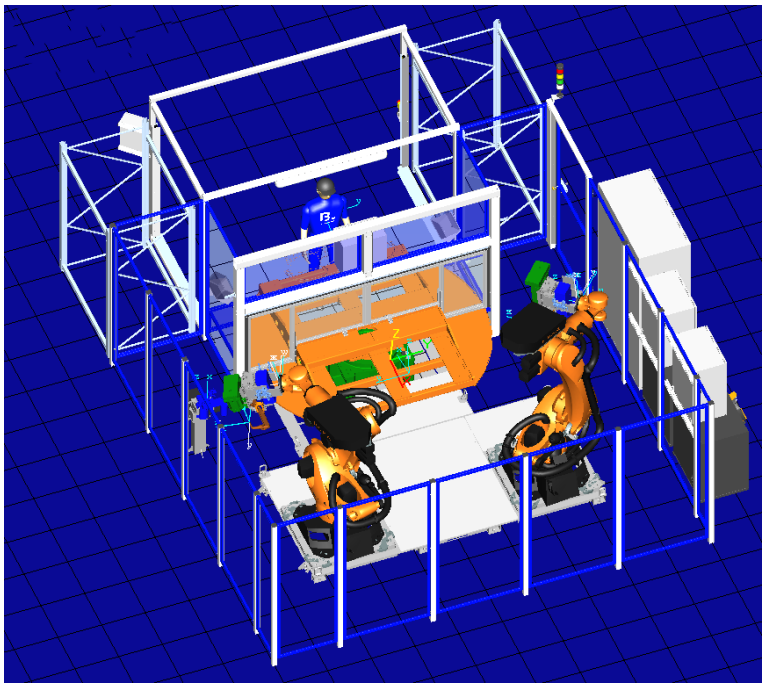

Figure 5.2. Real robotic cell provided by Blumenbecker Prag s.r.o.

5.2.1 Blumenbecker cell

One of the experiments was run on real robotic cell provided by Blumenbecker Prag s.r.o. (shown in Figure 5.2). The company is a partner of Czech Technical University in the eRobot project. The graph representing the simulation had 45 vertices. When we ran the simulation with the speed limit set to 100 %, the overall energy consumption was 200892 J (195231 J). Then we ran time optimisation. Since there were no collisions, the fastest solution was equal to the run with maximum speed. The cycle time was 60.75 s.

The energy consumption achieved by the MILP for energy optimisation was 193323.79 J. The real consumption, obtained in Process Simulate, was 184432 J. For the relaxed cycle time 70 s, the result was 182341.16 J. The real consumption was 169948 J.

The results with the estimator were the following: for the optimal cycle time, the optimised energy consumption was 211544.76 J. The real consumption was 190225 J. For the relaxed cycle time, the MILP returned 186669.1 J. The real consumption was 195747 J.

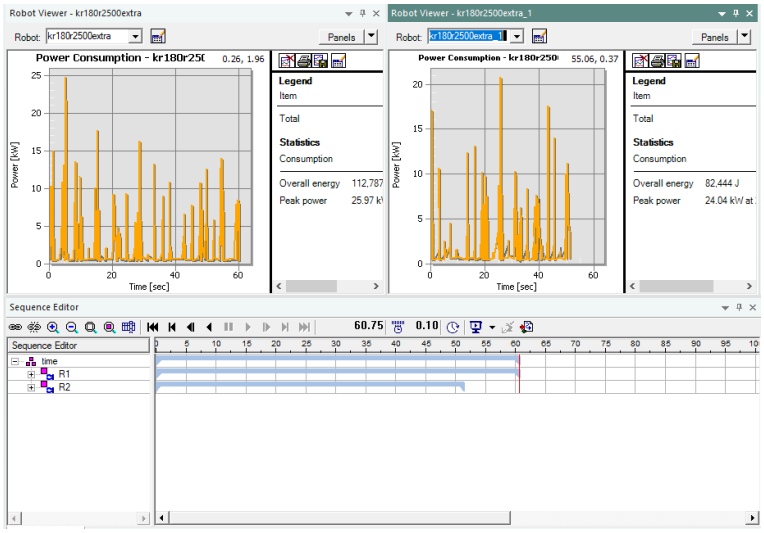

Figure 5.3. Power and energy consumption for time-optimal trajectory.

5.2.2 3 robots cell

This experiment was also run on another cell provided by Blumenbecker. In this cell, there are three robots Kuka KR 240 R 2900 ultra set up in such a way that many collisions need to be avoided (the cell is shown in Figure 5.4). The graph representing the simulation had 27 vertices.

Simulating at 100 % speed limit resulted in energy consumption of 109571 J, and cycle time 22.14 s. Collision-free, time-optimal solution resulted in cycle time of 30.58 s and consumption of 127067 J (113227 J). Since there were many collisions, another experiment was run. All vertices had a fixed speed limit of 100 % and a collision could be thus avoided only by adding wait time. The MILP for time optimisation provided a cycle time of 29.58 s and the consumption measured in Process Simulate was 132741 J (119565 J).

Then energy optimisation with energy profiles from the controller was run. The optimisation for the optimal cycle time provided a result with consumption 106358.88 J, while consumption in Process Simulate was 121652 J (107800 J). Then an optimisa-

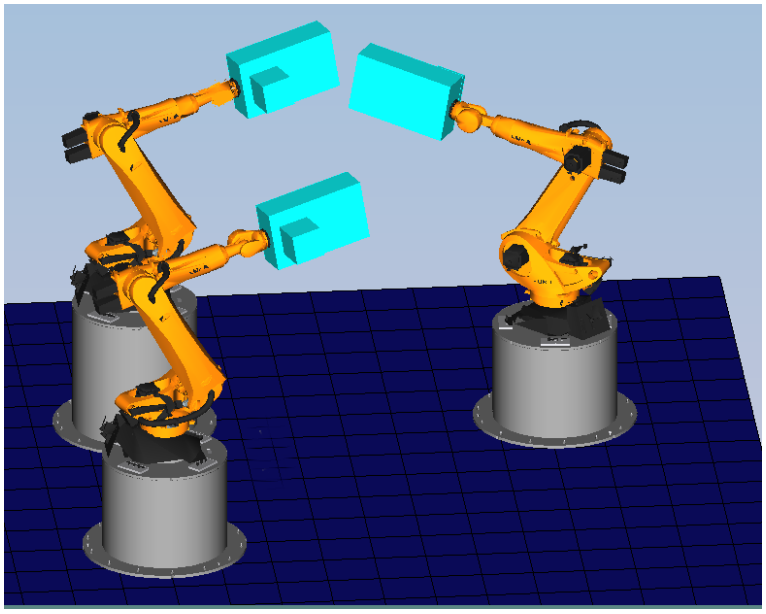

Figure 5.4. Robotic cell with three Kuka KR 240 R 2900 ultra robots.

tion was run with cycle time relaxed to 35 s. The MILP achieved consumption of 100800.77 J. The real consumption was 105955 J (90148 J).

The same experiments were also run with the estimator. The optimisation for the optimal cycle time provided consumption of 151556.35 J, and real consumption was 120809 J (107299 J). Then the optimisation was run with cycle time relaxed to 35 s. The MILP returned consumption of 145058.67 J. The real consumption was 121681 J (105741 J).

5.2.3 2 robots cell

The last experiment was done on a cell consisting of two Kuka KR 240 R 2900 ultra robots (shown in Figure 5.5), also provided by Blumenbecker. As in the previous case, the cell was set up in such a way that many collisions need to be avoided. The graph representing the simulation had 66 vertices. Initial consumption for the unlimited speed was 141153 J with a cycle time of 30.32 s. The time-optimal solution had a duration of 35.59 s and the energy consumption measured in Process Simulate was 138087 J. We made an additional experiment, the same as in the previous subsection. When the vertices had a fixed speed limit of 100 %, the MILP for time optimisation provided a cycle time 34.67 s, and the energy consumption was 151817 J (141103 J). That is an increase of energy consumption of almost 10 % in comparison with the base case, i.e. the case where vertices can have speed limit from interval 10 % to 100 %. It is another confirmation of the fact that the speed limit plays an important role in energy consumption. The reason the optimal cycle time for fixed speed limit is lower in this case and that of 3 robots cell than for the case when a vertex can have any speed limit from the interval from 10 % to 100 % is that the constraints in the initial MILP are different. The MILPs can then schedule all the vertices in a different way, which can result in different collisions and collision resolutions.

When the energy optimisation was run for the optimal cycle time, the MILP produced an output of 126355.96 J while the real consumption was 131607 J. The relaxed cycle time was 41 s and the MILP provided a solution consuming 122184.09 J. The real consumption in Process Simulate was 121378 J.

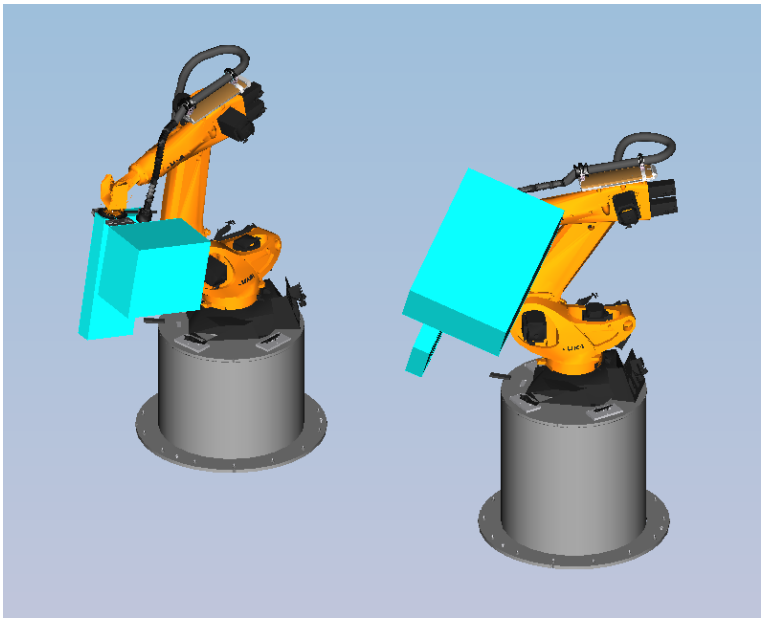

Figure 5.5. Robotic cell with two Kuka KR 240 R 2900 ultra robots.

Running the algorithm with the energy profile from the estimator, we got the following results. For the optimal cycle time, the MILP returned 320606.61 J. The real consumption was 141209 J (139196 J). With the cycle time relaxed to 41 s the result of optimisation was 291099.44 J. And the real consumption was 130125 J (127073 J).

5.3 Summary

This section provides a recap of the results obtained in the previous section. The results are shown in several tables. Table 5.1 shows a comparison between consumed energy for time optimisation T, for energy optimisation with optimal cycle time with energy measuring controller $T + E_1$ and for energy optimisation with optimal cycle time with estimator $T + E_2$. The last two columns show the decrease in energy consumption for the energy optimisation algorithm. Table 5.2 and Table 5.3 show the quality of energy optimisation, i.e. how far off is the optimal solution of the MILP from the real energy consumption measure in Process Simulate. The former table shows the optimisation with the energy profile generated using controller; the latter shows the optimisation with the energy profile generated from the estimator. Finally, Table 5.4 shows the results for relaxed cycle times. The first two columns contain the cell name and the type of optimisation, then there the increase of cycle time in per cent. The last two columns show the real energy consumption for the cycle time and energy decrease compared to time and energy optimal solution.

Cell	ω [s]	T [kJ]	$T + E_1$ [kJ]	$T + E_2$ [kJ]	Energy Consumption Decrease (E_1) [%]	Energy Consumption Decrease (E_2) [%]
Blumenbecker	60.75	200.9	184.4	190.4	**8.19 %**	5.24 %
3 robots	30.58	127.0	121.7	120.8	4.26 %	**4.92 %**
2 robots	35.59	138.0	131.6	141.2	**4.69 %**	-2.26 %

Table 5.1. Energy optimisation results for time-optimal trajectories.

In Table 5.1, we can see in the column $T + E_1$ and the penultimate column that we can save a significant amount of energy even for the optimal cycle time. For the cases when the energy profile was based on interpolation of measured energy, we consistently got an improved solution. Whereas when we based the energy profile on the estimator it provided a better result in one case and worse results in the others, and in one case it even increased energy consumption in comparison with the result of the time-optimal solution.

Cell	ω [s]	MILP output [kJ]	Real energy consumption [kJ]	Difference [%]
Blumenbecker	60.75	193.3	184.4	**8.44 %**
Blumenbecker	70	182.3	169.9	7.29 %
3 robots	30.58	106.4	121.7	**-12.57 %**
3 robots	35	100.8	106.0	**-4.86 %**
2 robots	35.59	126.4	131.6	**-3.99 %**
2 robots	41	122.2	121.4	**0.66 %**

Table 5.2. Quality of the energy optimisation with the controller.

Cell	ω [s]	MILP output [kJ]	Real energy consumption [kJ]	Difference [%]
Blumenbecker	60.75	211.5	190.2	11.2%
Blumenbecker	70	186.7	195.7	**-4.64 %**
3 robots	30.58	151.6	120.8	25.45 %
3 robots	35	145.1	121.7	21.67 %
2 robots	35.59	320.6	141.2	127 %
2 robots	41	291.1	130.1	124 %

Table 5.3. Quality of the energy optimisation with the estimator.

Table 5.2 and Table 5.3 show how much the MILP overestimated (the positive values) or underestimated (the negative values) the real solution. Even though this is not the most important criterion of the quality of the algorithm, it is good to see that results are rather reasonable. For the optimisation with data from the controller, we can see that the biggest error was about 12.5 %. The optimisation with the estimator provided less accurate results, especially for the 2 robots cell. That could be caused by specific properties of the cell. All the trajectories used to train the linear regression model were of the robot just following some points. However, this cell features many weld operations where the robot is required to undergo a long path in terms of Euclidean distance before doing the actual welding which causes an inaccuracy in energy profile estimation for vertices containing the weld operation.

In Table 5.4, we see how much energy we saved by relaxing cycle time in comparison with the optimal energy solution for the optimal cycle time. In Chapter 1, we mentioned the work of Meike et al. [13] which claimed that adding 50 % of extra time to an

Cell	Type	Time relaxation [%]	Energy Consumption [J]	Energy Consumption Decrease [%]
Blumenbecker	controller	15.2 %	169948	7.85 %
Blumenbecker	estimator	15.2 %	195774	-2.91 %
3 robots	controller	14.5 %	105955	12.90 %
3 robots	estimator	14.5 %	121681	-0.72 %
2 robots	controller	15.2 %	121378	7.77 %
2 robots	estimator	15.2 %	130125	7.85 %

Table 5.4. Energy optimisation results for relaxed cycle time.

operation can lead to energy savings of up to 20 %. Here, we added extra time to all operations, and we showed that even small cycle time addition could provide significant savings.

If we compare the results of energy optimisation with the result of time optimisation, then the decrease in energy consumption is up to 16.6 % for the 3 robots cell with energy profile based on controller data.

Chapter 6
Conclusion

This thesis focused on energy consumption optimisation. The main goal was to come up with a mathematical model that would describe this problem, then to come up with data-driven estimator of energy profiles, and finally to implement both in Process Simulate.

To be able to produce a model, it was necessary to get familiar with Process Simulate and its API. After exploring the options Process Simulate offers, it was possible to define the problem formally. Based on the problem statement, two mixed integer linear optimisation models were devised — one for time optimisation and another one for energy optimisation. The most important part of the MILP model was the so-called energy profile, a function approximating energy consumption for given vertex duration.

To get the energy profile, we need to have some information about energy consumption. Therefore we provided two ways of energy consumption information acquisition. One way was to use Kuka-Krc controller to obtain energy consumption samples and then interpolate them with a polynomial function. The other way covers the occasion when we have only the default controller available. For this case, we discussed a data-driven approach for approximation of the energy profile. We analysed features that could play an important role in energy estimation and provided a linear regression model. Based on this model, an estimator of energy profiles was implemented.

During the analysis, we also experimentally verified claims of Meike et al. [15] about the shape of the energy profile. Energy profile is indeed not a monotonous function.

An algorithm minimising energy consumption was described and implemented. Users can choose between two variants of the algorithm. In the first variant, the energy profile is created from controller data. In the other variant, the energy profile is generated from simulation data.

The implemented algorithm was tested on several robotic cells, including one real robotic cell provided by Blumenbecker. When we use the data from Kuka controller to produce energy profile, the tests showed that it is possible to decrease the energy consumption by up to 8 % for optimal cycle time and up to 16 % for relaxed cycle time. The MILP using energy profile estimated from simulation data was shown to overestimate the real energy consumption in most cases. Nevertheless, it was able to provide an improved solution in some cases.

6.1 Future work

There are many possible ways of improvement of this work. One could focus on better data acquisition either by finding a way how to generate better trajectories automatically or simply by measuring more of them. Using vertices from real robotic cells could also be a possibility.

Another possible approach could be the use of a different controller. In this work, we could not improve nor test the quality of the algorithm and the estimator with other robots than Kuka robots.

One could also focus on better analysis of the quality of the estimator and try to find whether it could be improved just by tweaking the intercepts.

References

[1] *Energy Use in Industry*.
https://www.eia.gov/energyexplained/index.php?page=us_energy_industry.

[2] European Commission, The. *EU energy in figures*. The European Commission, 2019.
https://publications.europa.eu/s/lKlH.

[3] *Pictures of the Future (Spring 2013)*.
http://web.archive.org/web/20170430133622/https://www.siemens.com/content/dam/internet/siemens-com/innovation/pictures-of-the-future/pof-archive/pof-spring-2013.pdf.

[4] *2020 climate & energy package*.
https://ec.europa.eu/clima/policies/strategies/2020/.

[5] *European Commission*.
https://cordis.europa.eu/project/rcn/108857/reporting/en.

[6] Meike, Davis, and Leonids Ribickis. Energy efficient use of robotics in the automobile industry. *2011 15th International Conference on Advanced Robotics (ICAR)*. 2011. Available from DOI 10.1109/icar.2011.6088567.

[7] Bukata, Libor. *Parallel Algorithms for Optimization of Production Systems*. 2018. Ph.D. Thesis.

[8] Bukata, Libor, Přemysl Šůcha, Zdeněk Hanzálek, and Pavel Burget. Energy optimization of robotic cells. *IEEE Transactions on Industrial Informatics*. IEEE, 2017, Vol. 13, No. 1, pp. 92–102.

[9] Dawande, Milind W. *Throughput Optimization in Robotic Cells*. Springer Science Business Media, LLC, 2007.

[10] Carabin, Giovanni, Erich Wehrle, and Renato Vidoni. A review on energy-saving optimization methods for robotic and automatic systems. *Robotics*. Multidisciplinary Digital Publishing Institute, 2017, Vol. 6, No. 4, pp. 39. Available from DOI 10.3390/robotics6040039.

[11] Riazi, Sarmad, Kristofer Bengtsson, Oskar Wigström, Emma Vidarsson, and Bengt Lennartson. Energy optimization of multi-robot systems. *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. 2015. Available from DOI 10.1109/coase.2015.7294285.

[12] Riazi, Sarmad, Oskar Wigström, Kristofer Bengtsson, and Bengt Lennartson. Energy and peak-power optimization of time-bounded robot trajectories. *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*. 2017. Available from DOI 10.1109/coase.2017.8256279.

[13] Meike, Davis, and Leonids Ribickis. Analysis of the energy efficient usage methods of medium and high payload industrial robots in the automobile industry. In: *10th International Symposium „Topical Problems in the Field of Electrical and Power Engineering“ Pärnu, Estonia*. 2011.

[14] Pellicciari, Marcello, Giovanni Berselli, Francesco Leali, and Alberto Vergnano. A minimal touch approach for optimizing energy efficiency in pick-and-place manipulators. *2011 15th International Conference on Advanced Robotics (ICAR)*. 2011. Available from DOI 10.1109/icar.2011.6088620.

[15] Meike, Davis, Marcello Pellicciari, Giovanni Berselli, Alberto Vergnano, and Leonids Ribickis. Increasing the energy efficiency of multi-robot production lines in the automotive industry. *2012 IEEE International Conference on Automation Science and Engineering (CASE)*. 2012. Available from DOI 10.1109/coase.2012.6386391.

[16] Gadaleta, Michele, Marcello Pellicciari, and Giovanni Berselli. Optimization of the energy consumption of industrial robots for automatic code generation. *Robotics and Computer-Integrated Manufacturing*. 2019, Vol. 57, pp. 452—464. Available from DOI 10.1016/j.rcim.2018.12.020.

[17] Wigström, Oskar, and Bengt Lennartson. Integrated OR/CP optimization for Discrete Event Systems with nonlinear cost. *52nd IEEE Conference on Decision and Control*. 2013. Available from DOI 10.1109/cdc.2013.6761100.

[18] Bukata, Libor, Přemysl Šůcha, and Zdeněk Hanzálek. Optimizing energy consumption of robotic cells by a Branch & Bound algorithm. *Computers & Operations Research*. 2019, Vol. 102, pp. 52—66. Available from DOI 10.1016/j.cor.2018.09.012.

[19] Sucha, P., Z. Pohl, and Z. Hanzalek. Scheduling of iterative algorithms on FPGA with pipelined arithmetic unit. *Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004.* Available from DOI 10.1109/rttas.2004.1317287.

[20] Buitinck, Lars, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013. pp. 108–122.

[21] *Linear Regression Example*.
`https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html`.

[22] *Process Simulate Documentation*.
`https://docs.plm.automation.siemens.com/tdoc/tecnomatix/13.0.1/PS_TC/`.

Appendix A
Abbreviations

API	Application Programming Interface
AREUS	Automation and Robotics for European Sustainable Manufacturing.
MILP	Mixed Integer Linear Programming.
OLP	Off-line programming.
TCPF	Tool Centre Point Frame.

Appendix B
Attachements

`src/data_csv.csv`	Measured data from trajectories in .csv format.
`src/regression.py`	A script loading the data from the .csv file and making the linear regression model.
`src/MovementGenerator.cs`	A class used to generate trajectories.
`src/Features.cs`	A class with a method for estimating features.
`src/FeatureConstants.cs`	A class storing the coefficients for the estimator.
`source_text/`	A folder with source files for this document.
`Models_for_Energy_Optimization_of_Robotic_Cells.pdf`	This document.