TECHNISCHE UNIVERSITÄT WIEN

DISSERTATION

Process Optimization and Control of a Patching Plant for Shuttering Panels

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der technischen Wissenschaften (Dr.techn.)

unter der Leitung von

Univ.–Prof. Dipl.–Ing. Dr.techn. Andreas Kugi
E376
Institut für Automatisierungs– und Regelungstechnik

eingereicht an der
Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von

Dipl.-Ing. Matthias Hofmair
Matrikelnummer: 0456800
Gernotgasse 2/3, 1150 Wien, Österreich

Wien, im August 2016

Dekan: Univ.–Prof. Dipl.–Ing. Dr.techn. Markus Rupp

Tag des Kolloquiums: 16.08.2016

Prüfungsvorsitzender: Em.O.Univ.–Prof. Dipl.-Ing. Dr.techn. Erich Gornik

Erster Gutachter: Univ.–Prof. Dipl.-Ing. Dr.techn. Andreas Kugi

Zweiter Gutachter: Prof. Dr. Olle Hagman

Vorwort

Diese Arbeit entstand im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Automatisierungs- und Regelungstechnik (ACIN) der Technischen Universität Wien. Das Forschungsprojekt *Holonic Integration of Cognition, Communication and Control for a Wood Patching Robot* wurde mit Partnern aus Forschung, namentlich Luleå Tekniska Universitet und Technische Universität München, sowie Industrie, namentlich Lip opažne plošce Bohinj, d.o.o., Microtec GmbH Srl, Springer Maschinenfabrik AG und TTTech Computertechnik AG, durchgeführt. Finanziert wurde es zum großen Teil aus Mitteln der EU-Forschungsförderung im 7. Rahmenprogramm (FP7/2007-2013), Fördervertrag Nr. 284573.

Ich möchte mich sehr herzlich bei Prof. Dr. Andreas Kugi für die hervorragende Betreuung meiner Arbeit bedanken. Er ist nicht nur fachlich ein Vorbild, sondern in ebenso hohem Maße persönlich. Außerdem bedanke ich mich bei Prof. Dr. Olle Hagman für die Erstellung des Zweitgutachtens und bei Prof. Dr. Erich Gornik für die Übernahme des Prüfungsvorsitzes.

Meinen Kollegen und Freunden vom ACIN danke ich - nicht nur, aber ganz besonders - für die kurzweiligen Mittagspausen, die wir gemeinsam verbracht haben. Sie gestalten den Arbeitsalltag bunt und lebendig. Danke für die schöne Zeit. Ebenso bin ich auch für die fachlich inspirierenden Diskussionen dankbar. Eure Unterstützung hat mir sehr geholfen. Besonders hervorheben will ich hier Martin Böck, Stefan Eberharter, Florian Schausberger, Dr. Andreas Steinböck und Martin Melik-Merkumians.

Unseren Projektpartnern gebührt ebenso mein Dank. Sie alle haben maßgeblich zum Erfolg des Projekts beigetragen. Dem Team von Microtec, insbesondere Dr. Federico Giudiceandrea, Dr. Konrad Tschurtschenthaler und Thomas Prenn, danke ich für Elan, Innovationsgeist und unkomplizierte, produktive Zusammenarbeit besonders herzlich. Auch Reinhard Rieger von der Springer AG danke ich für die gute Zusammenarbeit. Natürlich dürfen auch Franci Sodja und sein Team von Lip Bohinj nicht unerwähnt bleiben. Vielen Dank für die großartige Unterstützung bei der Inbetriebnahme der Anlage.

Schlussendlich sind es aber nicht nur die fachlichen Aspekte, die zum Erfolg meiner Arbeit beigetragen haben. Die Unterstützung, die ich im privaten Umfeld von meiner Familie und meiner Freundin Carolin Gasser erfahren habe, sind mir Rückhalt und Motivation, heute und in Zukunft.

Wien, im August 2016 Matthias Wolfgang Hofmair

Abstract

Growing competition and cost pressure in the timber market increase the need for production automation. The goal of this thesis is the automation and optimization of the patching process of wood defects, such as loose dead knots or resin galls, for shuttering panels[1].

In this patching process, the raw panels go through an optical defect scanner that determines position and shape of the wood defects. Then, the panels are processed at a patching robot. It consists of two xy-machine-tables that position the panel underneath a patching tool. This tool mends the detected defects by drilling the respective area and inserting a unisize, circular wood patch with high pressure to seal the hole. To maximize the throughput of the plant, several research tasks in the field of process optimization and control have to be performed. The results of these research tasks, which combine ideas and knowledge from algorithmic geometry, combinatorial optimization, control theory and computer science, are documented in this thesis.

The process optimization begins with the following research question: How should the patch arrangement look like, such that each defect is covered by the minimum number of unisize, circular patches. The proposed patch placement algorithm is based on the concept of hexagonally closest packing. This optimization step saves production time and enhances material utilization. Similar problems arise in telecommunications and sensor coverage.

Applying the patch placement algorithm to every defect of one panel yields a list of patch locations. These patch locations have to be approached in the time optimal sequence, called robot path. This problem is similar to the well-known traveling salesman problem. Two solution strategies are proposed, a classical Ant Colony Algorithm and a Local Search Receding Horizon Algorithm. The latter combines the Receding Horizon Concept known from the discipline of model predictive control with a simple, heuristic, local search routine for combinatorial optimization problems. Similar problems arise in a variety of industrial and

[1]The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under Grant Agreement No. 284573.

logistical applications.

To conclude the optimization process, the robot motion between each consecutive pair of patch locations needs to be executed in a time optimal way. To this end, a real-time capable Bang-Bang Trajectory Generator accounting for arbitrary initial velocity and acceleration of the patching robot is proposed. The algorithms for patch placement, path planning and trajectory generation are tested using challenging artificial test cases and real production scenarios of the prototype patching plant.

Next to process optimization, a major research task is real-time control of the patching robot. A cascaded control structure is used to cope with the challenging, unpredictable friction conditions in the drivetrain of the xy-machine-tables. A master-slave control concept is used to synchronize the two xy-machine-tables during certain process stages, where both xy-machine-tables move one panel cooperatively. The main scientific challange for the real-time control is the development of a control strategy that is able to position the panel in a fast and precise manner despite undefined relative motion between the panel and the patching robot. Therefore, absolute position measurement using visual tracking of the panel is required. This is only accomplished at a very low sampling rate and with a time delay. To overcome this issue a strategy for sensor data fusion called Trajectory Updating is proposed. It works robustly and, therefore, is applicable to a variety of similar industrial positioning tasks. Other strategies for trajectory tracking in the presence of slip are devised mainly in the field of autonomous, mobile robots.

Building on the real-time control, a process logic control for the prototype patching plant is developed. It basically consists of two interacting state-machines, one for each xy-machine-table of the patching robot. It is tested by means of a simulation model of the prototype patching plant.

Finally, in order to review the research results in an industrial environment, the optimization algorithms as well as the control strategies are implemented and extensively tested at the prototype patching plant, set up at a sawmill in Slovenia. The acquired measurement data is analyzed with special attention to the above mentioned challenges of friction, synchronization and sensor data fusion.

Kurzzusammenfassung

Wachsende Konkurrenz und steigender Kostendruck in der Holzindustrie verlangen nach einem höheren Automatisierungsgrad. Ziel dieser Arbeit ist die Automatisierung und Optimierung des Flickprozesses von Holzdefekten, wie z.B. Astlöchern und Harzblasen, für Schalungsplatten[2].

Im Zuge dieses Flickprozesses fahren die unbearbeiteten Schalungsplatten durch einen optischen Scanner zur Detektion von Position und Form der Holzdefekte. Dann werden die Schalungsplatten mit einem sogenannten Patchroboter bearbeitet. Dieser besteht aus zwei XY-Maschinentischen, welche die Platte unter dem Flickwerkzeug positionieren. Das Werkzeug bohrt die Holzdefekte aus und presst kreisförmige Vollholzstoppel einheitlicher Größe, genannt Patches, ein. Um den Durchsatz der beschriebenen Anlage zu maximieren, stellen sich eine Reihe von Forschungsaufgaben auf den Gebieten der Prozessautomatisierung und -optimierung. Die Resultate dieser Forschungsarbeit, die Ideen und Wissen aus den Disziplinen der algorithmischen Geometrie, der kombinatorischen Optimierung, der Regelungstechnik und der Informatik kombiniert, sind in dieser Arbeit dokumentiert.

Die Prozessoptimierung beginnt mit folgender Fragestellung: Wie muss die Anordnung der Patches aussehen, damit jeder Holzdefekt mit der minimalen Anzahl an Patches abgedeckt wird. Der vorgestellte Algorithmus zur Patchplatzierung basiert auf dem Konzept der hexagonal dichtesten Packung. Dieser Optimierungsschritt verringert die Produktionszeit bei gleichzeitiger Erhöhung der Ressourceneffizienz. Ähnliche Problem stellen sich in der Telekommunikation und bei der Sensorabdeckung.

Der Algorithmus zur Patchplatzierung wird auf jeden Holzdefekt einer Platte angewandt. Dadurch erhält man eine Liste von Patchpositionen für diese Platte, welche in der zeitoptimalen Reihenfolge, genannt Roboterpfad, angefahren werden sollen. Dieses Pfadplanungsproblem ist ähnlich dem Problem des Handelsreisenden, englisch Traveling Salesman Problem. Zur Lösung des Problems

[2]Die Forschung zur Erzielung dieser Ergebnisse wurde im 7. Rahmenprogramm der EU unter der Fördervertragsnummer 284573 finanziell unterstützt.

werden zwei Algorithmen herangezogen, ein klassischer Ameisenalgorithmus, englisch Ant Colony Algorithm, sowie ein lokaler Suchalgorithmus mit bewegtem Horizont, englisch Local Search Receding Horizon Algorithm. Letzterer kombiniert das Konzept der Optimierung mit bewegtem Horizont, bekannt aus der modellprädiktiven Regelung, mit einer einfachen, heuristischen, lokalen Suchroutine zur Lösung kombinatorischer Optimierungsprobleme. Ähnliche Probleme sind in einer Vielzahl industrieller und logistischer Anwendungen zu finden.

Im letzten Schritt der Prozessoptimierung muss die zeitoptimale Roboterbewegung zwischen den einzelnen Patchpositionen berechnet werden. Dies geschieht mithilfe eines sogenannten Bang-Bang Trajektoriengenerators. Dieser ist echtzeitfähig und berücksichtigt beliebige Anfangsgeschwindigkeiten und -beschleunigungen des Patchroboters. Die Algorithmen zur Patchplatzierung, Pfadplanung und Trajektoriengenerierung werden ausführlich anhand anschaulicher Testszenarien als auch realer Produktionszyklen der Prototypanlage getestet.

Neben der Prozessoptimierung ist die Roboterregelung eine wichtige Aufgabe. Ein kaskadierter Regelkreis ermöglicht schnelles und präzises Positionieren trotz erheblicher Haftreibungseffekte im Roboter. Eine sogenannte Master-Slave-Anordnung wird zur Synchronisation der beiden XY-Maschinentische herangezogen. Dies ist erforderlich, wenn beide XY-Maschinentische eine Platte gemeinsam bewegen. Die wissenschaftliche Herausforderung besteht in Entwurf und Implementierung einer Regelungsstrategie, welche die Platte trotz undefinierter Relativbewegung zwischen Roboter und Platte, schnell und präzise positioniert. Daher ist eine Absolutpositionsmessung der Platte erforderlich. Diese erfolgt visuell und folglich mit einer geringen Abtastrate sowie einem Zeitverzug. Zur Lösung dieses Problems wird ein System zur Sensordatenfusion, genannt Trajektorienaktualisierung, englisch Trajectory Updating, vorgestellt. Diese Regelungsstrategie funktioniert verlässlich in rauen Produktionsumgebungen. Somit ist sie für eine Reihe von industriellen Positionierprozessen einsetzbar. Andere Strategien zur Positionierung bei Schlupf wurden in der Literatur hauptsächlich zur Trajektorienfolge für mobile autonome Roboter entwickelt.

Aufbauend auf der Roboterregelung wird die Prozessablaufsteuerung der Prototypenanlage entwickelt. Diese besteht im Wesentlichen aus zwei interagierenden Zustandsautomaten, jeweils einer pro XY-Maschinentisch. Diese Steuerung wird mithilfe eines Simulationsmodells der Anlage getestet.

Abschließend werden die Regelungs- und Optimierungsalgorithmen auf der Prototypenanlage in einem Sägewerk in Slowenien implementiert und ausführlichen Tests in industriellem Umfeld unterzogen. Die aufgenommenen Messdaten werden im Detail und mit Fokus auf die besonderen Herausforderungen für die Regelung, nämlich Haftreibung, Synchronisation und Trajektorienaktualisierung, analysiert.

Contents

Symbols and Parameters

In the following, a list of symbols and plant parameters is provided. Values are only given for fixed, physical plant parameters.

Variable	SI-Unit	Value	Description
t	s	–	time
x	m	–	longitudinal coordinate
y	m	–	lateral coordinate
$(\cdot)_0$	–	–	initial
$(\cdot)_d$	–	–	desired
$(\cdot)_t$	–	–	trajectory
$(\cdot)_r$	–	–	patching robot
$(\cdot)_p$	–	–	panel
$(\cdot)^*$	–	–	optimal
$\dot{(\cdot)} = \mathrm{d}/\mathrm{d}t$	–	–	total time derivative
Raw material			
L	m	[1.015, 3.015]	panel length
W	m	[0.510, 0.520]	panel width
R	m	0.015	patch radius
m_p	kg	[4, 20]	panel mass, nominal 10kg
Patch placement			
$\mathbf{\Delta} = \{\mathbf{D}_i\}$	–	–	defect list for one panel side
$\mathbf{D}_i = \{\mathbf{d}_{i,j}\}$	–	–	defect polygon
N	–	–	number of elements in $\mathbf{\Delta}$
$\mathbf{d}_{i,j} = \begin{bmatrix} x_{i,j} & y_{i,j} \end{bmatrix}^{\mathrm{T}}$	$\begin{bmatrix} \mathrm{m} & \mathrm{m} \end{bmatrix}^{\mathrm{T}}$	–	node j of defect polygon $\mathbf{D}_i$
$n_{\mathbf{D}_i}$	–	–	number of nodes of $\mathbf{D}_i$

$\mathbf{P}_i = \{\mathbf{p}_{i,k}\}$	–	–	patch list for $\mathbf{D}_i$
$\mathbf{p}_{i,k} = \begin{bmatrix} x_{i,k} & y_{i,k} \end{bmatrix}^{\mathrm{T}}$	$\begin{bmatrix} \mathrm{m} & \mathrm{m} \end{bmatrix}^{\mathrm{T}}$	–	patch position
$n_{\mathbf{P}_i}$	–	–	number of elements in $\mathbf{P}_i$
$\boldsymbol{\delta} = \begin{bmatrix} \delta_x & \delta_y & \delta_\varphi \end{bmatrix}^{\mathrm{T}}$	$\begin{bmatrix} \mathrm{m} & \mathrm{m} & 1 \end{bmatrix}^{\mathrm{T}}$	–	displacement of defect polygon
Path planning			
$\boldsymbol{\Pi} = \{\mathbf{P}_i\}$	–	–	set of patch lists for all defects of one panel side, length N
$\mathbf{X} = \{\mathbf{x}_i\}$	–	–	set of nodes
$\mathbf{x}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^{\mathrm{T}}$	$\begin{bmatrix} \mathrm{m} & \mathrm{m} \end{bmatrix}^{\mathrm{T}}$	–	one node
$n = \sum_{k=1}^{N} n_{\mathbf{P}_i}$	–	–	number of elements in $\mathbf{X}$
$\boldsymbol{\Xi} = \{\xi_{i,j}\}$	–	–	set of arcs connecting $\mathbf{X}$
$\mathbf{C} = [c_{i,j}]$	–	–	cost matrix, size $n \times n$
$\boldsymbol{\psi} = [\psi_i]$	–	–	path vector, length n
J	–	–	path cost
Ant Colony Algorithm (ACOA)			
θ	–	–	iteration of ACOA
Θ	–	–	maximum number of iterations
M	–	–	number of ants
$p_{i,j}^{m}(\theta)$	1	[0, 1]	probability that ant m transitions from node i into j
$\mathbf{T}(\theta) = [\tau_{i,j}(\theta)]$	–	–	pheromone matrix, size $n \times n$
$\mathbf{H} = [\eta_{i,j}]$	–	–	heuristic matrix, size $n \times n$
α	–	–	weight of pheromone values
β	–	–	weight of heuristic values
ρ	1	[0, 1]	evaporation rate of pheromone
$\mathcal{M}_h^m$	–	–	local memory of ant m in iteration h of path building process

$\mathcal{N}_h^m$	–	–	feasible neighborhood of ant m in iteration h of path building process
ε	–	–	weight of pheromone deposition of elitist ant
Local Search Receding Horizon Algorithm			
$\boldsymbol{\Xi}_-$	–	–	set of arcs taken out of current path
$\boldsymbol{\Xi}_+$	–	–	set of arcs added to current path
I	–	–	number of iterations
Trajectory generation			
v_M	m/s	–	maximum velocity
a_M	m/s^2	–	maximum acceleration
j_M	m/s^3	–	maximum jerk
$j_i,\ i = \{1, \ldots, 7\}$	m/s^3	–	jerk values in phases P1 to P7
$T_i,\ i = \{1, \ldots, 7\}$	s	–	duration of phases P1 to P7
$\bar{T}_x$	s	–	trajectory time in x-direction
$\bar{T}_y$	s	–	trajectory time in y-direction
$\pi_a = \begin{bmatrix} j_1^* & T_1^* \end{bmatrix}$	–	–	parameter vector of $\mathcal{C}^0$-trajectory for acceleration
$A_t(a_0, a_d)$	–	–	function computing π_a
$\pi_v = \begin{bmatrix} \mathbf{j}_3^* & \mathbf{T}_3^* \end{bmatrix}$	–	–	parameter vector of $\mathcal{C}^1$-trajectory for velocity
$V_t(v_0, a_0, v_d)$	–	–	function computing π_v
$\pi_x = \begin{bmatrix} \mathbf{j}_7^* & \mathbf{T}_7^* \end{bmatrix}$	–	–	parameter vector of $\mathcal{C}^2$-trajectory for position
$X_t(x_0, v_0, a_0, x_d)$	–	–	function computing π_x

$a_t(t;\mathbf{j}_7,\mathbf{T}_7,a_0)$	–	–	acceleration trajectory at t
$v_t(t;\mathbf{j}_7,\mathbf{T}_7,v_0,a_0)$	–	–	velocity trajectory at t
$x_t(t;\mathbf{j}_7,\mathbf{T}_7,x_0,v_0,a_0)$	–	–	position trajectory at t
$T_j/2$	s	–	point-to-point trajectory, duration of phase of maximum jerk
T_a	s	–	point-to-point trajectory, duration of phase of maximum acceleration
T_v	s	–	point-to-point trajectory, duration of phase of maximum velocity
Control structure and trajectory updating			
e_x	m	–	correction of the set point x_d of the trajectory generator
e_v	m/s	–	correction of the set point v_t of the velocity controller
I_m	A	–	motor current
Process logic control			
$\mathbf{x}_{fi} = \begin{bmatrix}0 & y_{cnv1}\end{bmatrix}^{\mathrm{T}}$	$\begin{bmatrix}\mathrm{m} & \mathrm{m}\end{bmatrix}^{\mathrm{T}}$	–	Feed-In Position
$\mathbf{x}_{mo} = \begin{bmatrix}L + x_{lb21} & y_{cnv2}\end{bmatrix}^{\mathrm{T}}$	$\begin{bmatrix}\mathrm{m} & \mathrm{m}\end{bmatrix}^{\mathrm{T}}$	–	Move-Out Position
$\mathbf{x}_{ho1} = \begin{bmatrix}x_{ho} & y_{i-1}\end{bmatrix}^{\mathrm{T}}$	$\begin{bmatrix}\mathrm{m} & \mathrm{m}\end{bmatrix}^{\mathrm{T}}$	–	Handover Position 1
$\mathbf{x}_{ho2} = \begin{bmatrix}L - x_{ho} & y_{i-1}\end{bmatrix}^{\mathrm{T}}$	$\begin{bmatrix}\mathrm{m} & \mathrm{m}\end{bmatrix}^{\mathrm{T}}$	–	Handover Position 2
Sampling times			
T_s	s	0.001	velocity- and position controller
T_{si}	s	$1.25\mathrm{e}-4$	current controller
T_{sv}	s	0.200	visual position tracking
T_{dv}	s	0.100	time delay visual position tracking

Geometry and mass of the patching robot			
$r_{r,i}$	m	0.036	effective radius of rubber belts of XY-Table $i = \{1, 2\}$
$J_{r,i}$	$\mathrm{kg\,m^2}$	0.015	total inertia of drivetrain in longitudinal direction
$m_{r,i}$	kg	125.375	total moving mass in lateral direction
i_x	1	3	transmission ratio of drivetrain in longitudinal direction
$\varphi_{r,i} = i_x x_{r,i}/r_{r,i}$	1	–	angular position of sprocket driving the rubber belt
x_{lb11}	m	-1.575	longitudinal position of Light Barrier 11
x_{lb12}	m	-0.875	x-position of LB 12
x_{lb13}	m	-0.175	x-position of LB 13
x_{lb21}	m	0.175	x-position of LB 21
x_{lb22}	m	0.875	x-position of LB 22
x_{lb23}	m	1.575	x-position of LB 23
Sensor properties of the patching robot			
ρ_{xr}	$\mathrm{m^{-1}}$	5.4e4	resolution of rubber belt position x_r
ρ_{yr}	$\mathrm{m^{-1}}$	1.0e5	resolution of slide position y_r
ρ_{xp}	$\mathrm{m^{-1}}$	1.0e4	resolution of interferometer, i.e. of panel position x_p
Actuator properties of the patching robot			
$M_{r,i}$	N m	–	torque applied by servo motor
M_n	N m	8.7	nominal torque of servo motor
T_φ	s	$20\mathrm{e}-3$	time constant of servo motor, estimate
$F_{r,i}$	N	–	force applied by linear motor
F_n	N	560	nominal force of linear motor

T_y	s	$20\text{e}-3$	time constant of linear motor, estimate
T_p	s	2	duration of patching action
T_c	s	1	duration to engage/release clamping mechanism
T_z	s	1	duration to elevate/lower xy-tables
Parameters of the trajectory generator			
$\dot{\varphi}_M = i_x \dot{x}_M / r_r$	s^{-1}	27.778	x-direction, maximum angular velocity
$\ddot{\varphi}_M = i_x \ddot{x}_M / r_r$	s^{-2}	219.380	x-direction, maximum angular acceleration
$\dddot{\varphi}_M = i_x \dddot{x}_M / r_r$	s^{-3}	10969.000	x-direction, maximum angular jerk
$\dot{y}_M$	m/s	0.500	y-direction, maximum velocity
$\ddot{y}_M$	m/s^2	2.896	y-direction, maximum acceleration
$\dddot{y}_M$	m/s^3	144.800	y-direction, maximum jerk
Simulation model			
$\mathbf{z}_{r,i}$	–	–	state of XY-Table i
$\mathbf{z}_{p,j}$	–	–	state of Panel j
v_γ	m/s	5	maximum velocity of conveyors
T_γ	s	2	rise time up to v_γ
$d_{r\varphi,i}$	N m s	0.1	angular viscous friction parameter in x-direction
$d_{ry,i}$	N s/m	112	viscous friction parameter in y-direction
$\epsilon_{x,i,j}$	m	0.003	disturbance modeling slip in x-direction
$\epsilon_{y,i,j}$	m	0	disturbance modeling slip in y-direction

CHAPTER 1

Introduction

The majority of companies in timber industry are SMEs with limited funds for production automation and research. It, therefore, lags behind other industrial branches in these matters, although this situation has been slowly changing over the past decades. In the following, a brief overview of the main processes in timber industry and their degree of and potential for automation shall be given.

In wood construction and interior design, automated solutions for most process stages are now available, see [11]. The reason for the high degree of automation of wood construction and interior design is that its processes are rather similar to other industrial branches with highly automated production, such as metal processing. The similarity is due to the fact that in these stages of wood processing, the raw materials, i.e. for example solid wood panels, plywood, glue laminated wood or beams, exhibit homogeneous properties. In particular, they are of defined shape, surface quality and strength. Thus, existing handling and manufacturing processes, starting at production planning using CAM software and going down to the individual processing tasks using CNC workstations and robots, only require minor modifications for wood processing.

By contrast, the major aggravating factor for the automated production of these wood panels and beams etc. is the inhomogeneity of the unprocessed logs and lumber. The ideally cylindrical shape of logs, in practice, is of varying diameter, shows taper, bulges and dents. Furthermore, wood, as a natural product, inherently exhibits defects, such as cracks, knots, resin galls and rot. As a consequence, production of these panels and beams either require a lot of manual labor or a lot of valuable material is lost or both. To remedy this situation requires flexible automated material analysis and processing.

Every piece of unmachined wood is unique and, therefore, depending on the application, it needs to be analyzed regarding its properties, e.g., density, moisture content, strength, modulus of elasticity, shape, defect locations. This is a large field of research. A variety of different measurement methods are inves-

tigated and several are already successfully implemented in industry. Focusing on shape and defects, the most important methods are based on image analysis, laser scanning, micro- and ultrasonic waves as well as x-ray and computed tomography, see [4, 32, 55, 58, 59, 66, 82]. For overall wood grading purposes, multi-sensor approaches are necessary, in particular to accurately estimate the mechanical properties, see [14, 40].

The measured information is used to maximize the yield of each log. First, the log needs to be bucked to pieces of predefined length. Then, the cutting pattern for each part needs to be optimized. Finally, depending on the intended use, wood defects are patched. Patching instead of cutting out the defects during the bucking process increases material utilization and thus profitability. These tasks constitute complex optimization problems that not only need to consider information about the piece of wood, but also about the subsequent processing steps and the market demands. A lot of effort from both sides, the scientific community and the industry, is put into these problems and yet almost fifty percent of the sawn timber volume is lost during these processes, see [5]. Bucking optimization already starts at harvesting wood, see [70, 71]. Cutting pattern optimization based on computed tomography scans is state of the art, see [26, 71]. Patching of wood defects is done manually in the majority of cases, so that optimization of this process has not been investigated scientifically yet. If, next to volume optimization, also market demands are taken into account, the achieved revenue can be increased further, see [70, 86].

The scientific effort as well as the investment in expensive technology for material inspection indicate the potential of yield optimization techniques. However, the exploitation of the optimization results is oftentimes mediocre. This is because bucking and cutting the logs according to the computed patterns is not achieved with sufficient accuracy. According to [71], the harsh conditions of bucking will make accurate process measurements impossible in the near future. The problem of cutting the logs accurately, i.e. aligning them rotationally to the saw is called log rotation problem, see [6, 88, 89]. Currently, rotation accuracy is in the range of ten to twenty degrees. From a mechatronic point of view, log rotation is a very challenging task. Relative motion between the mechanics and the log cannot be avoided due to the irregular log shapes and the high processing speed. Therefore, some sort of absolute position tracking of the log and feedback control is imperative. This again is difficult because of the fast and shaky motion of the logs. Thus, a rigorous scientific approach combining mechanical and control engineering as well as state of the art sensor technology would be required to come to better solutions.

As already mentioned, patching of wood defects is largely a manual task. This disruption of the otherwise automated production process is inefficient, not only because of the labor costs. They make up the second largest part of expenses in timber manufacturing, only surpassed by costs for raw material, see [5]. The major problem is that human perception is subjective and precision

of manual work is prone to large variations. Thus, the achieved material quality in terms of homogeneity varies greatly. The wood patching process has not been scientifically investigated yet. Also, industry is only slowly taking up the matter of its automation. Nevertheless, this process holds a lot of potential for increased homogeneity and utilization of the wood material as well as increased productivity.

Objective of this work

In this thesis, patching of shuttering panels is investigated as a case study for a large variety of plain as well as laminated wooden products. The current industrial standard for this process are semi-automatic patching tools. An operator manually positions the raw panel underneath the patching tool and triggers the actual patching process. A drill is used to eliminate the defect. Then, a unisize, circular patch is inserted with high pressure to seal the hole. In this process, the operator carries out two major tasks. First, the defects need to be detected and classified, in principle a simple task for a human. However, to make this decision with a hundred percent consistency over an entire working day, not to mention in a team of several people, cannot be expected from humans. This negatively affects material quality and homogeneity. Second, the operators need to position the panels fast and with an accuracy of 1mm. This requires permanently elevated concentration which inevitably fades towards the end of a stressful working shift. Therefore, the goal of this thesis is the transformation of a traditional wood patching line for shuttering panels into a fully automated patching plant. To this end, several research tasks have to be performed.

First, a machine carrying out the positioning and patching operation of the shuttering panels shall be designed. Major design criteria for such a patching robot are robustness, reliability, fast material handling and cost efficiency. The patching robot receives the panel dimensions and defect data from a wood defect scanner that analyzes every unique wood panel before processing. These two machines make up the core of the prototype patching plant. This task is carried out in collaboration with Springer Maschinenfabrik AG and Microtec GmbH Srl.

Second, based on the patching robot design, the throughput-optimal processing sequence for each unique wood panel shall be computed. The input for these computations is the data collected by the defect scanner. It comprises panel shape and location as well as the shape of its wood defects.

Third, real-time control of the patching robot is a central task of this thesis. Thereby, positioning speed and accuracy are the key requirements. They need to be accomplished despite the unavoidable variations in shape, material quality and weight of the raw panels and the harsh production environment in sawmills. To this end, algorithms for visual identification and position tracking of the panels are contributed by Luleå University of Technology.

Finally, the prototype patching plant shall be set up at the sawmill of Lip Bohinj, d.o.o.. The task is to implement the process logic control of the plant in a safe and robust fashion. Then, extensive tests in a near industrial environment shall be carried out to demonstrate the plant's capability.

Outline of this thesis

The thesis is structured based on these research tasks. In Chapter 2, the raw material, i.e. the unprocessed shuttering panel, is analyzed. Based on that, a suitable design of the patching robot is presented together with an overview of the envisaged patching technique and process. The patching robot consists of the given patching tool used in the semi-automatic patching process and two xy-machine-tables to the left and the right of the tool, respectively. The xy-tables position the panel underneath the tool which carries out the patching action as described above. After the robot design, the patching plant is described as a whole, including the information flow within the plant.

In Chapter 3, processing optimization for each unique wood panel is described. These optimization algorithms need to consider the envisaged patching technique and process. In Section 3.1, a patch placement algorithm is developed based on the panel's dimensions and a list of its defects described as polygons. It computes the minimum number of patches and their arrangement to cover each defect polygon of the given list. This yields a list of patch locations that the patching robot must process. In Section 3.2, a path planning problem is formulated to approach these patch locations in the time optimal processing sequence. Two solution strategies, an Ant Colony Algorithm and a Local Search Receding Horizon Algorithm, are presented. The patch list is sorted according to the computed sequence. In Section 3.3, a time optimal trajectory generator is designed to compute the robot motion in between two patch locations in real-time. This trajectory generator is able to start at an arbitrary initial velocity and acceleration, which is required for the real-time control strategies presented in the next chapter. The algorithms for patch placement, path planning and trajectory generation are tested using challenging artificial test cases and real production scenarios of the prototype patching plant.

In Chapter 4, a position control strategy for the patching robot is proposed. A cascaded control structure suited to deal with the high friction forces within the xy-machine-tables of the patching robot is presented. A master-slave concept is used for the synchronization of the two xy-tables, which is necessary when both xy-tables move one panel cooperatively. The main challenge for the real-time control is the undefined relative motion between the xy-tables and the panel. In order to position the panel correctly despite this slip, a strategy for sensor data fusion is developed.

Chapter 5 is concerned with the process logic control for the patching robot.

It consists of two interacting state-machines, one for each xy-table. The process logic control is tested by means of a simulation model.

In Chapter 6, measurement results of the prototype patching plant prove the feasibility of the proposed concepts for real-time and process logic control. Four experiments are conducted. First, friction within the xy-tables is documented, then performance during coupled positioning of the two xy-tables is analyzed. The main result is a measurement of the entire patching process of a panel. The results are analyzed in detail, in particular regarding positioning speed and actuator utilization. Finally, the feasibility of the proposed sensor fusion concept to achieve the required positioning accuracy is demonstrated.

In Chapter 7, the work is summarized and possible future research tasks are identified.

CHAPTER 2

Patching plant

In Chapter 1, a brief overview of the semi-automatic patching process is given. Based on this semi-automatic process, which is state of the art, the patching plant is developed. To this end, all the necessary facts about the raw material for the process are established in Section 2.1. The patching robot design and the automated patching process are presented in Section 2.2. In Section 2.3, the prototype patching plant and the data flow within the plant are described. The definition of the coordinate systems of the patching robot and the shuttering panel are given in Section 2.4.

2.1 Shuttering panel

The design of a manufacturing machine begins with an analysis of the raw material, i.e. the unprocessed shuttering panels[1]. Figure 2.1a depicts an unprocessed panel exhibiting several wood defects. The panels' length ranges between 1.015m to 3.015m in half meter steps, the 2m-panel being the most common. They are between 0.510m to 0.520m wide and 0.021m to 0.026m thick. Depending on the dimensions, the type of wood and the humidity of the panel, their weight ranges between 4kg to 20kg. Ideally, the panels are rectangular and flat. However, in reality, the front and rear end of a panel might exhibit steps, as these panels consist of several slim laths that are agglutinated lengthwise. Moreover, the top few panels in a stack are frequently bent because they are stored outdoors and exposed to weather.

The considered defects are mainly loose dead knots, as shown in Figure 2.1b, and lack of material, but also resin galls, blue stain, embedded bark and rot. The number of defects per square meter, per panel side is approximately seven, but frequently goes up to twenty.

[1]For brevity, they simply shall be referred to as panels throughout the rest of the document.

(a) Shuttering panel. (b) Loose dead knot.

Figure 2.1: Typical unprocessed shuttering panel exhibiting several defects, mainly loose dead knots.

2.2 Patching robot

Based on the brief overview of the manual patching process in Chapter 1 and the description of the raw panels in Section 2.1, this section is concerned with the design of the patching robot. Only in recent years, industry has taken up the matter of automated patching of wood defects. Given the contour and location of the defects, basically three approaches exist in industry:

First, the exact area of the defect is milled by numerically controlled machines and either an individually shaped dowel or putty is used to seal the hole. This process is rather slow and, therefore, several patching tools operate on one work piece in parallel, making the process expensive. In turn, this complex technique offers the possibility to correct the wood without impairing its appearance. If the wood shall be used for furniture, doors or windows, this certainly is an important feature, but for shuttering panels it is unnecessary.

Second, small defects and cracks can be covered using putty only. After the putty is applied by an extremely fast tripod manipulator, the panels are exposed to UV-light which makes the putty cure. Naturally, this process can only be applied to high quality wood as it cannot repair bigger defects.

Third, there is the semi-automatic patching tool already mentioned in Chapter 1. After manually positioning the panel underneath the tool, a drill of radius $R = 0.015\text{m}$ is used to eliminate the defect. Then, an equally sized patch is inserted with high pressure to seal the hole, no glue or putty is used. In case of big defects, several of these unisize patches have to placed next to each other to cover the entire defect. Approximately 1500 of these patching tools are in use in Europe. They are the current industrial standard for patching shuttering panels. They are inexpensive and robust and shall, therefore, be integrated in the new patching robot design.

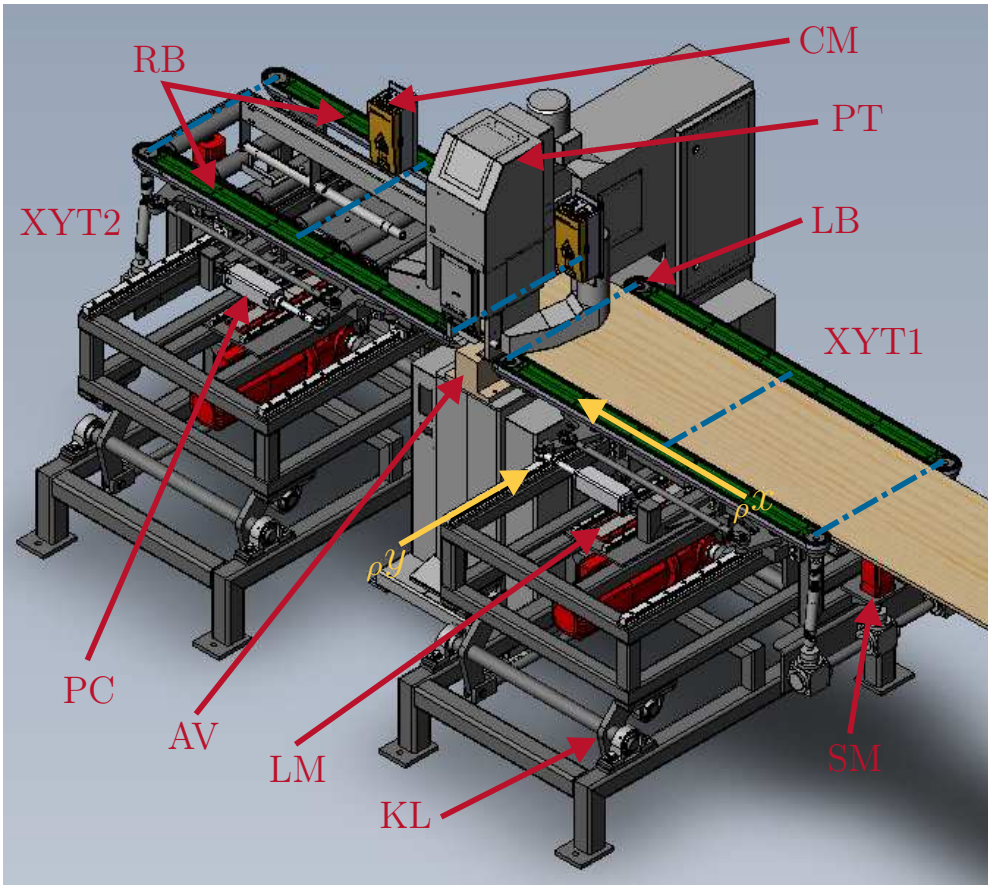

Figure 2.2: CAD drawing of the patching robot. The dash-dotted blue lines indicate light barriers that cannot be seen in the sketch. In positive x-direction, the light barriers are numbered 11, 12, 13 and 21, 22, 23 for XY-Table 1 and XY-Table 2, respectively. The planar xy-coordinate system of the patching robot $({}_\rho O\ {}_\rho x\ {}_\rho y)$ originates in the center of the drill of the patching tool.

2.2.1 Prototype

The patching tool (PT) already automates the patching process itself, the remaining task is positioning. To do that, two xy-machine-tables (XYT1, XYT2)[2] are placed to the left and to the right of the tool, see Figure 2.2. This construction serves two purposes: First, the anvil (AV) supporting the panel, where the patches are driven in by a pneumatic hammer, remains in place and thus absorbs the impact force. Second, the patching robot can handle two panels simultaneously, thus minimizing the waiting periods for the tool.

In *longitudinal direction*, the panel is moved by a rubber belt conveyor (RB). It consists of two rubber belts that are mechanically synchronized by a distribution gear which is driven by one synchronous servo motor (SM). The

[2]For brevity, in the rest of the document they shall be denoted as xy-tables.

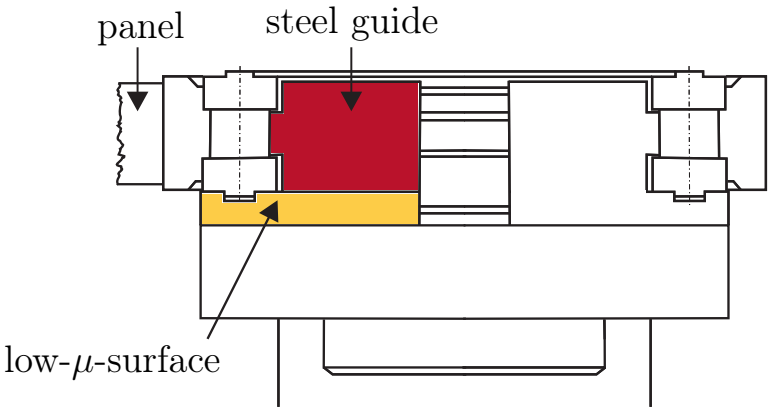

Figure 2.3: Sectional drawing of the right rubber belt.

rubber belts clamp the panel on its sides by means of a pneumatic cylinder (PC). High clamping pressure is necessary to transmit the acceleration forces to the panel. However, at the same time, this leads to high friction forces within the drive train of the rubber belts. To alleviate this unavoidable effect, two measures are taken, see Figure 2.3. The rubber belt basically is a rubber-coated roller chain. In order to absorb the high clamping forces, the inner guides of the chain are made of steel. Plastic guides would deform under the pressure and thus impair smooth motion of the chain. Furthermore, underneath the chain, there is a layer of synthetic material with a very low friction coefficient. On the outer side, these measures are not required.

The acceleration forces are transmitted from the rubber belts to the panel via friction. Consequently, relative motion, i.e. slip, between the belts and the panel may occur. In order to reduce this effect, the panel needs to be uniformly clamped. Therefore, exact parallel alignment of the chain guides relative to each other and relative to the effective rolling radius of the sprockets driving the belt is of importance.

The belt position is measured using the motor encoders. To ensure the required accuracy, the drive train has to be sufficiently stiff and without free play in the gearboxes and the cardan shafts. Additionally, due to the panel slip, cameras (CM) for visual tracking of the absolute position of the panel are installed.

In *lateral direction*, the entire slide on which the belt conveyor is mounted moves on two rails. It is actuated by a synchronous linear motor (LM). Since the panels are frequently longer than the xy-tables, they need to be elevated. This is accomplished by a knee lever mechanism (KL). Thus, the panel does not contact the conveyor when moving laterally. Otherwise, unnecessary wear of the conveyor belt would be the result. Lateral position measurement is accomplished using the motor encoder.

Each xy-table has three light barriers (LB) detecting the panels, one at each

end and one in the middle as indicated by the blue lines. Important parameters of the patching robot are to be found in Chapter *Symbols and Parameters*.

2.2.2 Patching process

Given the design of the patching robot, a panel is positioned at a given, i.e. previously computed, patch location as follows:

1. XY-Table 1 is in Feed-In Position, i.e. the clamping mechanism of the rubber belts is released, XY-Table 1 is lowered and aligned with Conveyor 1. Panel 1 at Conveyor 1 approaches XY-Table 1 from the right.
2. As soon as the light barriers of XY-Table 1 indicate a panel occupies the table, the clamping mechanism engages.
3. Panel 1 is moved to the rear, i.e. left, end of XY-Table 1.
4. As soon as the patching tool is not occupied by the previous panel any longer, XY-Table 1 is elevated and free to move laterally.
5. The positioning process starts. Three possible cases can occur:
   (a) XY-Table 1 positions the panel.
   (b) As soon as the panel occupies the work space of XY-Table 2, it alignes with XY-Table 1, elevates itself and clamps the panel as well. Both xy-tables move the panel together. They are said to be coupled then.
   (c) As soon as the panel leaves the work space of XY-Table 1, it releases the panel and lowers itself. XY-Table 2 positions the panel.
6. After XY-Table 1 is in Feed-In Position again, it picks up Panel 2. Continue at Step 1.
7. After each positioning action, the patching tool is activated.
8. As soon as all patch positions of Panel 1 are processed, it is handed over to XY-Table 2, if it is not already at XY-Table 2.
9. The rear end of Panel 1 is moved to the front, i.e. right, end of XY-Table 2. XY-Table 2 alignes with Conveyor 2. This position is called Move-Out Position.
10. XY-Table 2 is lowered.
11. Panel 1 is moved out to Conveyor 2.
12. The rubber belts of XY-Table 2 are released as soon as XY-Table 2 is not occupied anymore.

After a panel is positioned correctly, the patching process is executed, which approximately takes $T_p = 2$s:

1. The panel is fixed by a blank holder pressing it against the anvil.
2. A circular hole, which is $2R = 0.030$m in diameter and approximately 0.009m deep, is drilled.
3. A hollow core drill together with a wooden lath moves directly over the hole. Since the offset between the drill and the hollow core drill is known and constant, it is implemented with a simple limit stop.
4. This hollow core drill manufactures the patch from the wooden lath. The circular patch sticks within the hollow core drill.
5. A pneumatically actuated hammer rams the cylindrical patch into the previously drilled hole.

2.3 Prototype patching plant

Figure 2.4 depicts a block diagram of the prototype patching plant. The top half of the figure shows the way of the panels through the production line, the bottom half the associated information flow.

The panels start at the inbound storage (IS), go through the defect scanner (DS) and on to the buffer storage (BS). The defect scanner is a timely synchronized camera network, which captures several images of top and bottom side of the panel[3]. At a dedicated computer for panel analysis (PA), these images are merged into one picture for the top and one for the bottom side. From these pictures, contour, length and width of the panel are extracted. Moreover, each of these pictures is analyzed by a defect detection algorithm, which describes the borders of each defect by a polygon. The panel pictures, contour and dimensions as well as the list of defect polygons, in short defect list, are sent to a database PC, called wood knowledge repository (KR). A new entry in the wood knowledge repository is created, which triggers the optimization algorithms for processing each panel side. A patch placement algorithm computes the minimum number of patches and their arrangement for each defect of one panel side. All the patch locations for one panel side are then handed over to a path planning algorithm, that computes the optimal processing sequence. The result is a list of patch locations for each panel side. Finally, a trajectory generator computes the optimal motion profiles for the patching robot between each pair of subsequent patch locations. The patch list and the corresponding robot trajectories are added to the respective panel entry in the wood knowledge repository.

[3]For details refer to our partner company MiCROTEC, see http://www.microtec.eu.

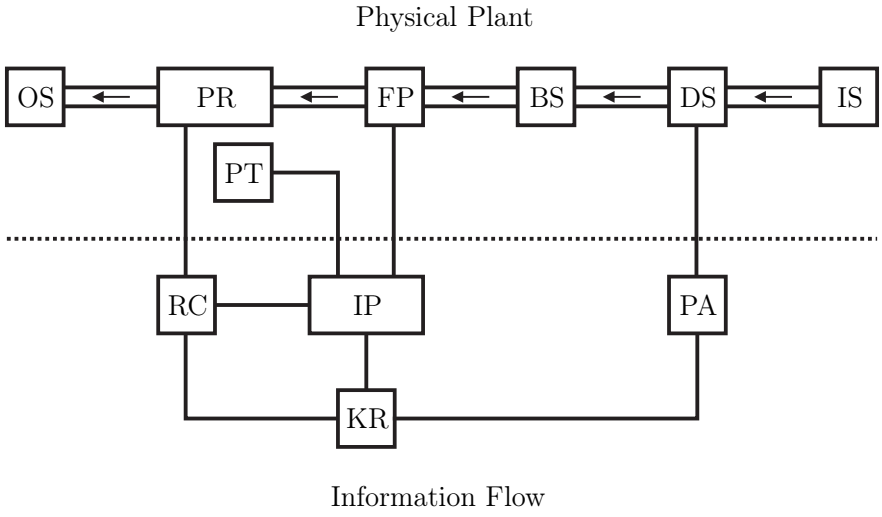

Figure 2.4: Block diagram of the essential plant components.

Since the scanner is much faster than the patching robot (PR), the buffer storage interrupts the panel flow. Thus, before a panel from the buffer storage moves on to the patching robot, the panel and its respective side to be processed need to be identified. To do that, a camera called panel finger printer (FP) takes an image of the panel. A computer for image processing (IP) compares this image to the panel images stored in the wood knowledge repository, looking for unique wood patterns, such as a unique wood grain or a unique arrangement of wood defects, see [74, 75]. The patch list corresponding to the identified panel side is sent from the wood knowledge repository to the real-time controller (RC) for controlling the patching robot and the conveyors. This controller is run on a separate PC and executes the entire plant control, in particular the patching process as described in Section 2.2. As already mentioned, slip between the rubber belts of the robot and the panel occurs. Therefore, two cameras for position tracking (PT) are installed at the robot. In the IP, the pictures of these cameras are compared to the respective picture from the wood knowledge repository, thus determining the absolute longitudinal position of the panel. After processing is finished, the panels are collected in the outbound storage (OS).

Finally, an outlook on the envisaged production plant shall be given. It consists of one defect scanner and three parallel patching lines. Each of these patching lines has two patching robots and a turner unit in between them. The first robot patches the top side, then the panel is turned and the second one patches the bottom side of the panel. At this final plant, the buffer storage provides the possibility for panel scheduling to maximize capacity utilization of the robots.

2.4 Definition of coordinate systems

Two coordinate systems are required, see Figure 2.5 and compare to Figure 2.2. The coordinate system of the panel (${}_{\pi}O\,{}_{\pi}x\,{}_{\pi}y$) originates in the bottom left corner of the smallest circumscribing rectangle of the panel. In case the panel is ideally rectangular, the origin is in the bottom left corner of the panel itself. In reality, however, the panel exhibits steps at its ends. Thus, the origin might not be at the panel itself. The coordinate system of the patching robot (${}_{\rho}O\,{}_{\rho}x\,{}_{\rho}y$) originates in the center of the drill of the patching tool.

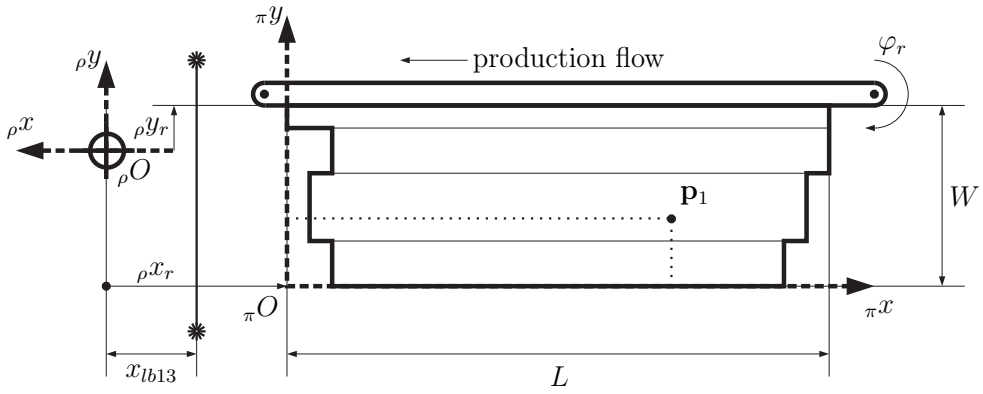

Figure 2.5: A panel consisting of four laths that are agglutinated lengthwise.

The longitudinal position ${}_{\rho}x_r$ of the rubber belt of the patching robot is defined in the following. It is measured indirectly via the angle of its driving sprocket $(\varphi_r - \varphi_{r0})/i_x$, i.e.

$$ {}_{\rho}x_r = \frac{r_r}{i_x}\,(\varphi_r - \varphi_{r0}). \tag{2.1} $$

In (2.1), φ_r is the angle of the motor shaft (as measured by the motor encoder), φ_{r0} is an offset, r_r is the effective radius of the sprockets driving the rubber belts and i_x is the transition ratio of the gearbox. However, the rotational motion of the motor does not have a point of reference. It is only obtained when a panel is clamped[4]: ${}_{\rho}x_r = 0$, when the front end of the panel, i.e. the ${}_{\pi}y$-axis, is underneath the center ${}_{\rho}O$ of the drill of the patching tool. To implement this definition, the longitudinal robot coordinate is reset to

$$ {}_{\rho}x_r(t_{rF}) = -x_{lb13} = \frac{r_r}{i_x}\,(\varphi_r(t_{rF}) - \varphi_{r0}), \tag{2.2} $$

when Light Barrier 13 of XY-Table 1 detects a rising flank in its interrupted-

[4]This implies the assumption of no slip between panel and rubber belt.

signal at time t_{rF}. Solving (2.2) for the offset value[5] yields

$$\varphi_{r0} = \varphi_r(t_{rF}) + \frac{i_x}{r_r} x_{lb13}. \tag{2.3}$$

The lateral position ${}_\rho y_r$ of the xy-table of the patching robot is defined as the position of the sidewall of the fixed rubber belt, i.e. the one that is not moved by the pneumatic cylinder, relative to ${}_\rho O$. Consequently, when the sidewall is aligned with the center of the drill of the patching tool, ${}_\rho y_r = 0$.

Therefore, a generic point ${}_\pi\mathbf{p}_1 = \begin{bmatrix} {}_\pi x_1 & {}_\pi y_1 \end{bmatrix}^{\mathrm{T}}$ given in $({}_\pi O\ {}_\pi x\ {}_\pi y)$ is transformed to $({}_\rho O\ {}_\rho x\ {}_\rho y)$ by

$${}_\rho\mathbf{p}_1 = \begin{bmatrix} {}_\rho x_1 \\ {}_\rho y_1 \end{bmatrix} = \begin{bmatrix} -{}_\pi x_1 + {}_\rho x_r \\ -(W - {}_\pi y_1) + {}_\rho y_r \end{bmatrix}. \tag{2.4}$$

[5] In the remainder of this document, the offset value φ_{r0} is neglected for brevity, meaning $\varphi_r \leftarrow \varphi_r - \varphi_{r0}$. It is solely required for the formal definition of the panel position.

CHAPTER 3

Optimization of panel processing

The algorithms for processing optimization are tailored to the specific design of the patching plant, in particular to the patching robot, as presented in Chapter 2. As soon as there is a new panel entry added to the wood knowledge repository, the time optimal processing program for each panel side is determined, see [48]. The inputs for these algorithms are the panel dimensions and a list of defect polygons. First, for each defect of one panel side, a *patch placement algorithm* calculates the minimum number of patches and their arrangement to cover the defect, see Section 3.1. The result is a list of patch locations. Second, a *path planning algorithm* determines the time optimal robot path, i.e. the processing sequence, connecting all the patch locations, see Section 3.2. The patch list is sorted according to this sequence. Third, a *trajectory generator* computes the time optimal trajectories between each pair of consecutive patch locations. This yields the desired position, velocity and acceleration profiles for each positioning action, see Section 3.3. Thus, time optimal processing of each panel side is achieved.

The panels are scanned for defects at a rate of approximately 0.2Hz. So, there is a time slot of 5s to execute all three algorithms twice, once for each panel side.

3.1 Patch placement

As already mentioned in Section 2.2, the patching tool only uses cylindrical patches with a radius of $R = 15$mm. They allow to patch approximately 85% of all wood defects using merely one patch. The other 15% are too big to be covered by one patch only, see Figure 3.1. Since no glue is used, it is obvious that the more patches are placed next to each other, the more fragile this arrangement becomes. Some defects are even too big for patching and thus the respective

Figure 3.1: Defect covered by three patches.

panel must be discarded from the patching process.

To avoid this whenever possible, a patch placement algorithm has to compute the minimum number of congruent patch circles and their arrangement to cover these big defect polygons. Striving for a minimum number of patches serves two important purposes. First, each patch requires approximately $T_p = 2$s processing time, the positioning time not included. Second, since the maximum number of patches per defect is limited due to quality reasons, minimizing the number of patches indirectly, but, nonetheless, significantly contributes to waste reduction.

3.1.1 Literature review

From a geometrical point of view, the problem of patch placement is covering an arbitrary polygon, i.e. the defect, with the minimum number of equal circles, i.e. the patches. Considerable research has already been conducted on the problem of optimal covering of circles, e.g., [87], equilateral triangles, e.g., [67, 72], and rectangles, e.g., [46, 73]. This problem is typically formulated in one of three ways. Given two of the following three variables - dimensions of the shape to cover, radius and number of circles, determine the third variable as well as the related arrangement of the circles. This problem is either solved by a mathematical proof, like in [46, 67, 87], or by optimization techniques, as in [72, 73]. The latter two papers propose an algorithm consisting of two levels. In the inner level, a fixed number of circles of a given radius is available to minimize the uncovered area of the polygon. Starting at a random circle configuration, the minimization is achieved using the BFGS Quasi-Newton Method, see [7]. In the outer level, the radius is adjusted according to the result of the inner level.

Regarding the covering of an arbitrary planar shape with congruent circles, an important theoretical result is presented in [61]. Given the arbitrary shape to

be covered and the minimum number of circles required to cover this shape, the following statement is proven: Let the radius of the circles approach zero, then the total area of the circles is 121% of the area of the shape to be covered. Clearly, if the radius approaches zero, the approximation of the given shape improves. Conversely this means, that the unavoidable overlap between the circles is 21% of the area to be covered.

Practical applications of polygon covering using congruent circles arise in telecommunications and sensor coverage. Typically, in homogeneous environments using only one type of communication beacon or sensor, their ranges are modeled as congruent circles. In [69], various definitions of covering and existing solution strategies are given. The task at hand, i.e. covering an entire region using congruent circles, is called *full coverage*[1]. There exist two versions of the full coverage problem. First, given a network that covers the area multiple times, determine the redundant nodes. If possible, compute multiple disjoint sets of nodes, so that each set covers the entire region, see [21, 91, 92, 97]. Then, switch on only one set at a time to save energy and prolong the lifetime of the sensors. Second, given the region to cover and either (a) the number of nodes or (b) their covering radius, compute the optimal deployment to minimize the respectively other (a) or (b), see [3, 22]. Commonly, these problems are tackled by dividing the region in the Voronoi diagram of the sensor/transmitter network, see [3, 21, 22, 91]. Every node has a contribution to the coverage of the region as big as its Voronoi polygon. If the Voronoi polygon of each node is within its covering circle, then the region is fully covered.

A different approach to the full coverage problem might be hexagonally closest packing, see [90]. Inscribing circles of radius $\sqrt{3}/2\,R$ in a tessellation of equilateral hexagons of side length R yields the arrangement of circles that leaves the least space between the non-overlapping circles uncovered. The dual problem to packing is covering. By circumscribing the hexagons of side length R by circles of radius R, one receives the covering of a plane with minimum overlap between the circles. For example, in [97], a hexagonal grid is used advantageously to improve the algorithm presented in [92], where a square grid is used.

3.1.2 Algorithm

The input for the patch placement algorithm is the defect list $\mathbf{\Delta} = \{\mathbf{D}_i\}$, $i = 1, \ldots, N$, of the respective panel side. Each element of the defect list defines one closed defect polygon by a list of vertices, $\mathbf{D}_i = \{\mathbf{d}_{i,j}\}$, $\mathbf{d}_{i,j} = \begin{bmatrix} x_{i,j} & y_{i,j} \end{bmatrix}^{\mathrm{T}}$, $j = 1, \ldots, n_{\mathbf{D}_i}$, where $x_{i,j}$ and $y_{i,j}$ denote the coordinates of vertex j of defect i with respect to the bottom left corner of the panel (${}_{\pi}O\ {}_{\pi}x\ {}_{\pi}y$), see Section 2.4. It

[1] Note that the full coverage problem in wireless sensor networks features one more constraint called connectivity. However, under the reasonable assumption that the communication range of the sensors is larger than twice the sensor range, full coverage implies connectivity. Thus, this constraint is obsolete.

describes the contour of the defect by connecting vertex $\mathbf{d}_{i,j}$ to vertex $\mathbf{d}_{i,j+1}$ until finally the last vertex $\mathbf{d}_{i,n_{\mathbf{D}_i}}$ is again connected to the first vertex $\mathbf{d}_{i,1}$.

The polygon $\mathbf{D}_i$ has to be entirely covered by a number $n_{\mathbf{P}_i}$ of circles of a given radius R. Each such circle is described by its center coordinates $\mathbf{p}_{i,k} = \begin{bmatrix} x_{i,k} & y_{i,k} \end{bmatrix}^{\mathrm{T}}$. All patches required to cover the defect $\mathbf{D}_i$ are merged into the set $\mathbf{P}_i = \{\mathbf{p}_{i,k}\}$, $k = 1, \ldots, n_{\mathbf{P}_i}$.

Problem formulation The task is to determine the minimum number $n_{\mathbf{P}_i}$ of patches and their arrangement to cover the entire defect. Furthermore, a set of constraints ensuring that the patches adhere to the panel (and do not fall out in later production stages) has to be met. In particular, each patch must have a minimum overlap with solid wood and the patches themselves must not overlap too much. This is formulated as an optimization problem

$$\min_{n_{\mathbf{P}_i}, \mathbf{P}_i} n_{\mathbf{P}_i} \tag{3.1a}$$

$$\tilde{\boldsymbol{\gamma}}(\mathbf{D}_i, \mathbf{P}_i, n_{\mathbf{P}_i}) \leq \mathbf{0}, \tag{3.1b}$$

where $\tilde{\boldsymbol{\gamma}}$ is the vector of constraints. In order to check the defect covering and the overlap of each patch with solid wood, the defect polygon is intersected with each patch circle. Also the patch circles themselves are intersected with each other to ensure sufficiently small overlap of the patches.

The problem with the formulation (3.1) is that the number of optimization variables $\tilde{n}_o = 1 + 2n_{\mathbf{P}_i}$ increases with the number of patches $n_{\mathbf{P}_i}$. Each patch has to be placed individually taking into account all the other $n_{\mathbf{P}_i} - 1$ patches. Furthermore, (3.1) constitutes a mixed-integer optimization problem, which is known to be quite challenging.

Solution based on hexagonally closest packing To overcome this issue, the patch placement algorithm is founded on the idea of hexagonally closest packing, see [90] and Figure 3.2. An equilateral hexagon is the one polygon that comes closest to the circular shape, while at the same time it can still be arranged in a pattern fully covering a plane, leaving no holes in between the hexagons. Thus, minimal overlap between the circles is ensured. This naturally implies maximum covered area (for a fixed number of patches) and consequently guarantees that the minimum number of patches is utilized.

The algorithm starts by creating a hexagon tessellation in a sufficiently large part of the xy-plane, where each hexagon of side length R represents a circle of equal radius. The defect polygon is put in the middle of the tessellated area, i.e. the centroid of the defect polygon coincides with the centroid of the tessellated area. The optimization is now achieved by moving the defect along the x- and y-axis, denoted by δ_x and δ_y, and by rotating it by an angle δ_φ until a global minimum in the sense of the previously described criterion is found. During this

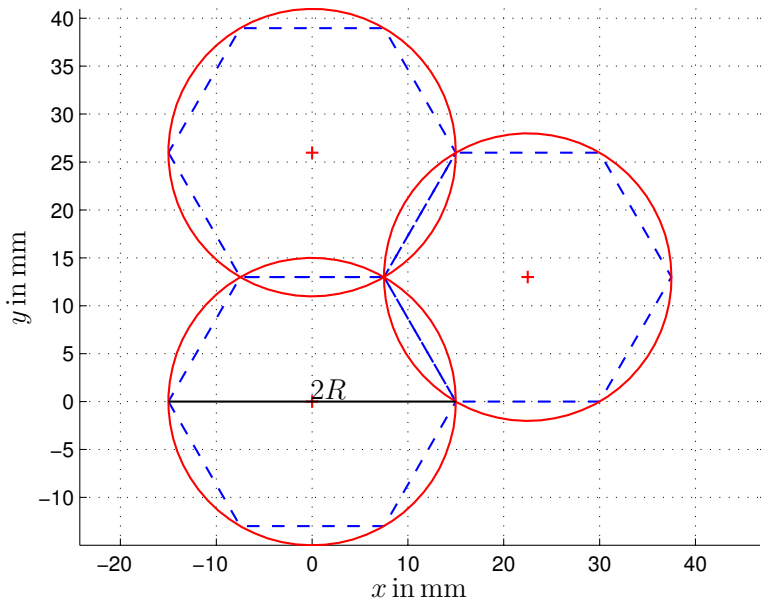

Figure 3.2: Hexagonally closest packing.

process hexagons that do not intersect with the defect polygon are taken out. Others are added where the defect polygon is not covered.

Compared to the original problem formulation, the patches are perfectly arranged to each other. Thus, the optimization problem can be implemented more efficiently in the form

$$\min_{\boldsymbol{\delta}} n_{\mathbf{P}_i} \tag{3.2a}$$

$$\boldsymbol{\gamma}(\mathbf{D}_i, \boldsymbol{\delta}) \leq \mathbf{0}, \tag{3.2b}$$

where $\boldsymbol{\delta} = \begin{bmatrix}\delta_x & \delta_y & \delta_\varphi\end{bmatrix}^{\mathrm{T}}$ is the displacement of the defect polygon relative to the tessellated area. This way, the number of optimization variables is reduced to $n_o = 3$. Moreover, the vector of constraints $\boldsymbol{\gamma}$ is significantly simplified, since the patch placement relative to each other is fixed and needs not be checked anymore. The majority of intersections, i.e. each patch with every other patch, becomes obsolete.

The hexagon tessellation has a periodic pattern. If the defect is moved over an x-distance of the width of a hexagon, $\bar{\delta}_x = 2R$, if it is moved over a y-distance of the height of a hexagon, $\bar{\delta}_y = \sqrt{3}\,R$, or if it is rotated around $\bar{\delta}_\varphi = \pi/3$, the pattern starts to repeat itself. So, the space of all possible solutions $\mathbb{D} = [0, \bar{\delta}_x] \times [0, \bar{\delta}_y] \times [0, \bar{\delta}_\varphi]$ is closed.

Nevertheless, the solution space still contains an infinite number of points. Only by applying a fixed step size $\boldsymbol{\delta}_d = \begin{bmatrix}\delta_{dx} & \delta_{dy} & \delta_{d\varphi}\end{bmatrix}^{\mathrm{T}}$ for the displacement

of the defect, one obtains a finite number of possible solutions $N_{\mathbb{D}} = \left(\bar{\delta}_x/\delta_{dx}\right)$ $\left(\bar{\delta}_y/\delta_{dy}\right)$ $\left(\bar{\delta}_\varphi/\delta_{d\varphi}\right)$. A natural choice for this step size is the positioning accuracy of the patching robot. Assuming a positioning accuracy of $\delta_{dx} = \delta_{dy} = 1\text{mm}$ and choosing $\delta_{d\varphi} = \pi/36$, yields $N_{\mathbb{D}} = 9360$. By going through all these solutions, it is guaranteed that the global minimum with respect to the positioning accuracy is obtained in a finite number of $N_{\mathbb{D}}$ steps.

Implementation Algorithm 1 gives an overview of the main functionality of the patch placement algorithm in pseudo code. As an input it receives the defect polygon `D`. Output is a patch list `PL` of length `nPopt` holding the xy-coordinates of the minimum number of patches to cover the defect. A defect polygon consists of one contour without holes. However, after, e.g., calculating the difference between a defect polygon and a hexagonal patch polygon, the remainder of the defect polygon might have a hole or fall apart into several individual contours. The data structure of the polygon has to be designed accordingly.

By moving the centroid `[xC0,yC0]` of the defect polygon `D` to the origin, `D` becomes `D0`. The hexagon tessellation around the origin of the xy-plane is stored in `hexList`, that holds `nHL=11*11` individual hexagon polygons. The list marking the hexagons needed to cover the defect is initially set to true, `coverList[1:nHL] = true`. The three dimensional solution space of (3.2) is divided in a mesh as described above and the nodes of that mesh are stored in a list `moveList` of length `nML=9360`. One of its entries consists of the longitudinal, lateral and rotational displacement `moveList[i] = [dX, dY, dPhi]` that is applied to `D0` in iteration step `i` of the main loop. This way the moved polygon `Dm` is generated. It is essential to make sure that between two consecutive elements of `moveList` only one of the three entries changes by one step. Thus, a 'smooth motion' of the defect polygon relative to the hexagon tessellation is guaranteed and the previous solution only has to be adopted slightly[2].

After the defect polygon was moved, a new covering is computed. First, all the hexagons that do not (at least partially) cover the defect polygon are deactivated[3], i.e. `coverList[j] = false`. Second, the defect polygon might not be covered entirely, since a part of the polygon might have moved to a region, where the hexagon tessellation has previously been deactivated. In this case, the hexagon covering this part is activated again. This is achieved by computing the centroid of each uncovered contour `[xCC,yCC]` of `Dm` and activating the hexagon whose center coordinates are closest to it. It is important to note that, given a smooth `moveList`, as described above, and under the assumption of a fine resolution of the solution space, it cannot happen that the area of one contour

[2]By contrast, if the defect polygon randomly 'jumped' from one location in the hexagon tessellation to a totally different one in one iteration step, a completely new covering would have to be computed.

[3]Thus, after the first iteration `i=1`, already big parts of the hexagon tessellation are deactivated, which speeds up computation considerably.

Algorithm 1 Patch placement algorithm

```
function PL = PatchPlacement(D)
//initialization
[xC0, yC0] = centroidArea(D);
move0 = [xC0, yC0, 0]; //D is rotated by 0°
D0 = move(D, move0); //move D to the origin
moveOpt = moveList[1]; //movement D0 vs hexTesselation
nPopt = inf; //number of required hexagons
coverListOpt = coverList; //index of required hexagons

for( i=1; i<=nML; i++ ) {
   covered = false;
   //move polygon by dX, dY, dPhi
   Dm = move(D0, moveList[i]);
   //deactivate hexagons that do not cover Dm
   for( j=1; j<=nHL; j++ ) {
      if( coverList[j] ) {
         //intersection polygon pIS between hexagon and Dm
         pIS = intersect(Dm, hexList[j]);
         if( area(pIS)==0 ) {
            coverList[j] = false;
         } else {
            //compute the polygon that is not covered yet
            Dm = difference(Dm, hexList[j]);
            if( area(Dm)<=0 ) {
               coverList[j+1:nHL] = false; //remaining hex. futile
               covered = true; break;
            }
         }
      }
   }
   //activate hexagons where Dm is not covered
   if( !covered ) {
      for( k=1; k<=numberContours(Dm); k++ ) {
         contour = getContour(Dm, k);
         [xCC, yCC] = centroidArea(contour);
         iCl = closestHexagon(hexList, xCC, yCC);
         coverList[iCl] = true;
      }
   }
   //update optimal solution, L,W...panel dimensions
   validSolution = checkConstraints(Dm, hexList, coverList, L, W);
   nP = countActiveHexagons(coverList);
   if( validSolution && nP<nPopt ) {
      nPopt = nP;
      moveOpt = moveList[i];
      coverListOpt = coverList;
   }
}

PL = generatePatchList(hexList, coverListOpt, -moveOpt, -move0);
```

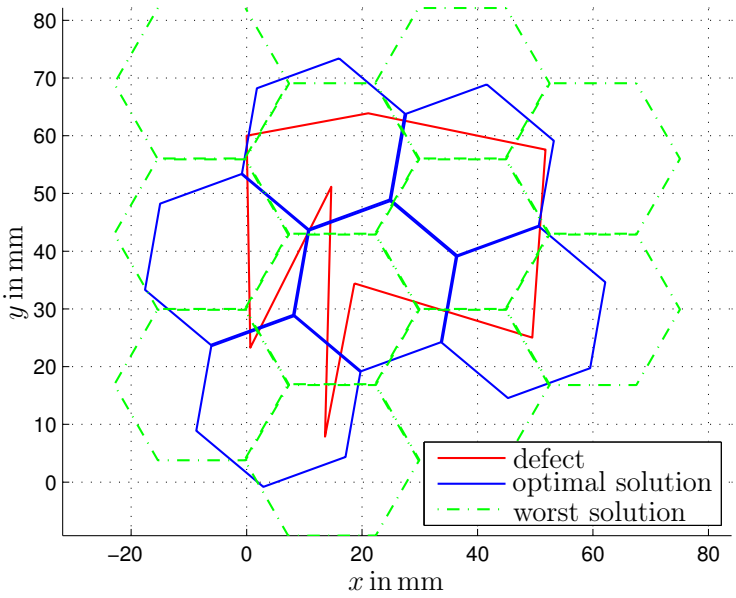

Figure 3.3: The patch placement algorithm reduces the number of required patches for the defect to a global minimum of six.

is so big that it requires more than one patch to be covered. Third, in case the solution obeys the constraints and requires less patches than the previous optimal solution, the optimal solution is updated.

Finally, after going through the entire movement list `i=nML`, the globally optimal solution is derived. In function `generatePL`, the actual covering for `D` is computed. To this end, the inverse movement `-moveOpt` is applied to the hexagons activated in `coverListOpt`. Then, the patch hexagons are moved to the original location of `D`, i.e. the hexagons are moved by `-move0`. The output is a patch list `PL` for the given defect.

3.1.3 Results and Analysis

Figure 3.3 exemplarily shows the potential of the patch placement algorithm. The optimal solution requires six patches to cover the defect as compared to eleven patches of the worst solution. Major advantages of this approach are: First, the algorithm does not impose any restrictions on the shape of the defect polygon, in particular convexity is not required. Second, a global minimum with respect to the positioning accuracy is found. Third, any number of arbitrary constraints can be added. If one of these constraints is not met, the solution is not feasible and thus ignored. This is important with respect to the feasibility of the patch arrangement. Fourth, the algorithm can be terminated any time, which is essential for hard real-time constraints. Then, the best solution found

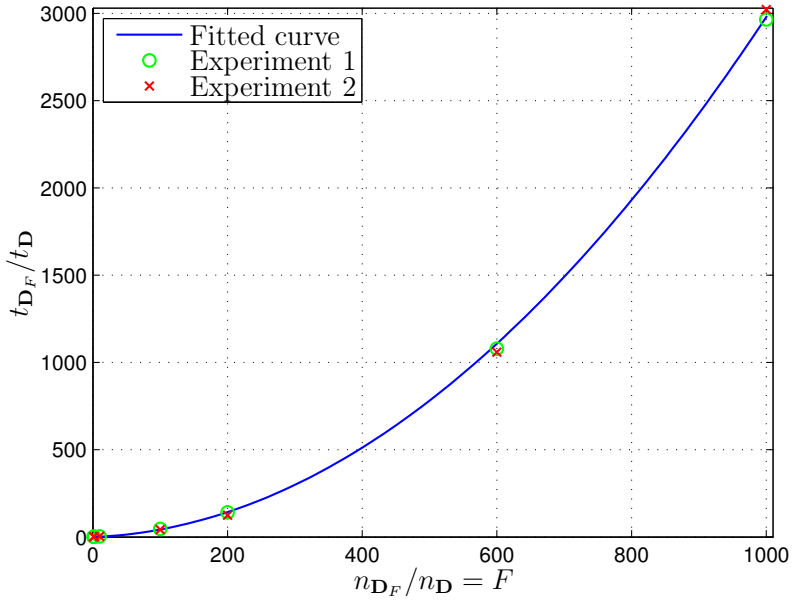

Figure 3.4: Normalized computation time $t_{\mathbf{D}_F}/t_{\mathbf{D}}$ versus normalized problem size $F = n_{\mathbf{D}_F}/n_{\mathbf{D}}$.

so far is going to be worked with. The major drawback of this algorithm is its rather high computational cost, which is characterized by its time complexity.

The time complexity of an algorithm depends on the problem size n, which, according to [83], is equal to the length of the input string. In case of the patch placement algorithm n is equal to the number of nodes $n_{\mathbf{D}}$ of the defect polygon $\mathbf{D}$. In order to analyze the time complexity of the patch placement algorithm the following experiment is carried out.

To begin with, polygons of the same shape and size, but with a different number of nodes are created. This is achieved by adding nodes exactly at the edges of the original polygon. Suppose an edge of the original polygon is given by the two nodes $\mathbf{d}_j$ and $\mathbf{d}_{j+1}$. Then, the number $n_{\mathbf{D}}$ of nodes of the original polygon is increased by a factor F by inserting $(F-1)$ nodes

$$\mathbf{d}_{j,f} = \mathbf{d}_j + f\,(\mathbf{d}_{j+1} - \mathbf{d}_j)/F,\ f = \{1, \ldots, F-1\} \tag{3.3}$$

into every edge of the polygon. This way, the polygons $\mathbf{D}_2$, $\mathbf{D}_{10}$, $\mathbf{D}_{100}$, $\mathbf{D}_{200}$, $\mathbf{D}_{600}$ and $\mathbf{D}_{1000}$ are created, for the factors $F \in \{2, 10, 100, 200, 600, 1000\}$.

Then, the patch placement algorithm is applied to these polygons. It goes through exactly the same computation steps for each of the defect polygons since they are of identical shape and size. Only the input size, i.e. the number of nodes $n_{\mathbf{D}}$, varies. The result is depicted in Figure 3.4. It shows the computation time of the patch placement algorithm $t_{\mathbf{D}_F}$ normalized to the computation time $t_{\mathbf{D}}$ of the original defect polygon over the normalized problem size $F = n_{\mathbf{D}_F}/n_{\mathbf{D}}$. Two

examples are chosen, Experiment 1 refers to a rather small defect requiring only three patches, whereas Experiment 2 is carried out with the big defect shown in Figure 3.3. The absolute computation time $t_{\mathbf{D}}$ of the original defects is 0.07s and 0.22s, respectively.

Clearly, the experiments reveal a quadratic dependence of the computation time on the problem size, in particular a least-squares curve fit of the experiments yields

$$\frac{t_{\mathbf{D}_F}}{t_{\mathbf{D}}} = c_2 \left(\frac{n_{\mathbf{D}_F}}{n_{\mathbf{D}}} \right)^2 + c_1 \frac{n_{\mathbf{D}_F}}{n_{\mathbf{D}}} + c_0 \tag{3.4}$$

with constants $c_2 = 0.0025$, $c_1 = 0.1479$ and $c_0 = 0.0028$.

Among all steps in the optimization process, the minimization of the number of patches has the largest contribution to savings of both production time and wood. Each patch requires approximately 3s of production time, including positioning and the patching action itself. Also, the maximum number of patches allowed per defect is limited due to quality reasons, which is why minimizing the number of patches needed per defect enables to process even lower quality raw material and as a consequence reduces the amount of rejects. If only one defect of a panel is too big to patch, the entire panel, i.e. up to 20kg of wood, needs to be rejected. In conclusion, it is worth computing the global minimum of patches required for each defect, even at higher computational costs.

3.1.4 Sanity check of the defect data

For panel analysis, several algorithms for identification of different kinds of defects are applied to one panel image. These algorithms are of non-deterministic nature and, therefore, they might yield overlapping defect polygons. This faulty data would corrupt the patch placement algorithm, which treats every defect polygon individually. This results in several overlapping patch arrangements that are not necessary. To overcome this problem, a *sanity check algorithm* is run on the defect data before it is forwarded to the patch placement algorithm. The goal is to ensure, there are no overlapping defect polygons and that each defect polygon only consists of one contour without holes. In order to have a robust algorithm, an indefinite number of non-convex defect polygons is assumed to be in one place. Based on this, a three-step algorithm was developed, see Algorithm 2. Its input is the defect list `DL` of length `nDL`. Output is a new defect list `DL` without overlapping defect polygons.

First, a pair of overlapping defect polygons is looked for by computing the intersection `pIS` of the polygons of the defect list `DL` with each other. Second, when such a pair `DL[i], DL[j]` is found, their union `pUN` is computed. Possible holes of `pUN` are neglected. Third, `DL[i]` and `DL[j]` are replaced by `pUN` in the defect list and the procedure is called recursively until no more intersections are found.

Algorithm 2 Sanity check algorithm

```
function DL = SanityCheck(DL)
nDL = numberOfElements(DL);
for( i=1; i<=nDL-1; i++ ) {
   for( j=i+1; j<=nDL; j++ ) {
      pIS = intersect(DL[i],DL[j]);
      if( area(pIS)>0 ) {
         pUN = unite(DL[i],DL[j]);
         pUN = neglectHoles(pUN);
         DL = deletePoly(DL,i,j);
         DL = addPoly(DL,pUN);
         //number of defects: nDL = nDL-1;
         call(DL = SanityCheck(DL));
      }
   }
}
```

3.2 Path planning

Applying the patch placement algorithm to each defect of one panel side yields a list of patch locations, in short patch list. The patching robot shall approach these patch locations in the optimal order. More precisely, the path planning algorithm aims at computing the minimum cost path between a given number of patch locations such that each one is visited exactly once. The costs can be chosen freely and are, for instance, path length, travel time or energy consumption. This problem is very similar to the well-known traveling salesman problem, where a salesman tries to find a minimum cost round trip through a given number of cities. However, the panel is not supposed to go on a round trip and come back to where it started, but to move forward in the production line. Nevertheless, the known algorithms for the traveling salesman problem can still be applied.

3.2.1 Literature review

The traveling salesman problem is an NP-hard combinatorial optimization problem, see [36], for which a variety of different solution strategies are proposed in the literature. Exact approaches formulate the traveling salesman problem as a linear integer problem, which is then solved using *Branch-and-Cut Algorithms*, see [2, 19]. The big advantage of these approaches is that they guarantee to find the global minimum cost round trip. On the downside, no polynomial time algorithms exist that find the exact solution of the traveling salesman problem. In comparison, heuristic methods, such as Evolutionary or Ant Colony Algorithms, find good, i.e. nearly optimal, solutions much faster. Therefore, they are more relevant to the application at hand, where the maximum available computation time per panel is limited.

Evolutionary Algorithms, i.e. Genetic Algorithms and its derivatives, are inspired by the process of evolution and natural selection, see [25, 29, 77]. In this context, a set of feasible solutions to a combinatorial optimization problem is given by a population of individuals. Each individual represents a feasible solution by its genome, which typically is a string over a finite alphabet. The genome of an individual determines its fitness on the basis of a problem specific quality criterion. The optimization is achieved by increasing the mean fitness level of the population in an iterative two-stage-process. First, a set of individuals, called parents, is chosen to generate offspring by recombination of their genomes. Parts of their genomes are merged by a recombination function to form one or more different feasible solutions. Then, these children undergo mutation, i.e. a random change in their genome, to create the population of the next generation. Second, after a new generation of the population is created, a selection procedure decides, which individuals die and which live, i.e. which solutions are deleted and which are kept. Individuals whose fitness level is higher have a greater chance of survival. Both, the choice which individual becomes a parent and the decision which individual survives the phase of natural selection are stochastic processes biased by the fitness level of each individual. Thus, also weak members of the population have a certain probability to generate offspring and survive. This encourages diversity of the population, which is essential for solution space exploration, and inhibits convergence towards a local optimum. Another important factor in solution space exploration versus solution convergence is the frequency and severity of mutations.

For theoretical analysis, Genetic Algorithms are described as a finite Markov Chain, where the transition probabilities between the states are gathered in a transition matrix. Analysis of this transition matrix allows to make statements on the statistical behavior, in particular the convergence properties of the Genetic Algorithm. In [76], it is proven that the Canonical Genetic Algorithm never converges to the global optimum, unless the fittest individual in a population is maintained. Schmitt provides a mathematically well-founded theory on the behavior of Genetic Algorithms , see [79, 80, 81]. By statistical analysis of the main operations, i.e. recombination, mutation and selection of the fittest, the asymptotic behavior of different versions and implementations of Genetic Algorithms is shown. Finally, a general-purpose Genetic Algorithm that asymptotically converges to one of possibly several global optima is presented.

Ant Colony Algorithms are another search metaphor inspired by nature. The foraging behavior of ants demonstrates their remarkable ability to solve shortest path problems between food and nest by a rather simple cooperative approach. Each ant leaves behind pheromone trails on the paths it traverses. Other ants can smell pheromone and tend to follow these trails, reinforcing them as they also deposit pheromone. In turn, pheromone evaporates over time. Within a fixed period of time, shorter paths can be traversed more often and thus they accumulate higher pheromone values. This way, over time, the shortest path

achieves a dominant status and the vast majority of ants follows it. Ant Colony Algorithms, see [23, 24, 29], mimic this behavior to solve combinatorial optimization problems, like the traveling salesman problem. These algorithms are based on a population of ants, i.e. simple software agents. At each time step, i.e. in each algorithm iteration, each ant independently builds a path by repeatedly making random decisions on which arc to traverse next. These decisions are biased by the cost associated with the arc and the pheromone value of the arc. On the one hand, the costlier an arc, the smaller the chance it is traversed; on the other hand, the higher the pheromone value, the higher the chance the respective arc is crossed. After each ant built its path, the pheromone value of all arcs are lowered through evaporation. Then, the pheromone value of each traversed arc is increased. Thus, after several time steps of the population building paths, one path emerges the dominant one.

Theoretical analysis of Ant Colony Algorithms lags far behind its practical success. In [37], feasible solutions are described by walks on the construction graph of the combinatorial optimization problem at hand. Based on this framework, two methods which guarantee the convergence to the global optimum are proposed. The probability that after a fixed number of time steps a globally optimal path is traversed by one ant can be made arbitrarily close to 100% either by increasing the number of ants or by reducing the evaporation rate of pheromone. In [24, 84], a two-step proof that the algorithm converges to one element of the set of globally optimal solutions is given. In a first step, convergence in value is proven, if there is a positive lower bound on the pheromone value. Thus, it is guaranteed that the proposed Ant Colony Algorithm finds an optimal solution with a probability that can be made arbitrarily close to 100% if given enough time. However, the lower bound on the pheromone value prohibits convergence to a particular solution. Therefore, in a second step, the lower bound on the pheromone value decreases over the time steps of the algorithm. If this decrease happens slowly enough, the property of convergence in value can be kept, while at the same time convergence in solution is obtained. Based on these results, two Ant Colony Algorithms with guaranteed convergence to the optimal solution are developed in [15, 38]. In [41], the finite-time behavior of an Ant Colony Algorithm with only one ant for a linear quality criterion with boolean input is described. Applying drift analysis, a mathematical tool for analysis of evolutionary algorithms, reveals a polynomial time complexity of this algorithm. In [54], the finite-time behavior of Ant Colony Algorithms for the traveling salesman problem is analyzed. Based on the relationship between pheromone rate and convergence time, the iteration time until the pheromone rate reaches its objective value is studied. Then, the convergence time is calculated using this objective value.

Evolutionary and Ant Colony Algorithms are able to find reasonably good paths very fast. A different heuristic approach to the traveling salesman problem are *Local Search Algorithms*, see [56, 57]. In contrast, these algorithms require a given feasible path, which they then improve incrementally. Therefore, Evo-

lutionary and Ant Colony Algorithms are often referred to as path construction algorithms and Local Search as path improvement algorithms. In the context of Local Search Algorithms, a feasible solution of a traveling salesman problem is a path on the fully connected construction graph of the respective problem instance. Local Search Algorithms improve a given path by iteratively replacing a set $\mathbf{\Xi}_-$ of arcs of the given path by another set $\mathbf{\Xi}_+$ of arcs that also yields a feasible solution, but with lower cost. The size of these sets is restricted to λ, which is why this technique is called λ-exchange. This principle was introduced in [65]. The core functionality of the presented algorithm randomly performs 3-exchanges, i.e. $\lambda = 3$, until no further improvement can be achieved. It is of complexity $\mathcal{O}(n^3)$. In [64], the Lin-Kernighan Heuristic is presented as an extension of the above basic algorithm. It is considered one of the most efficient methods to generate near-optimal solutions for the traveling salesman problem. The major improvement over the basic algorithm is a body of rules for the choice of the sets $\mathbf{\Xi}_-$ and $\mathbf{\Xi}_+$. These rules are further refined in [43]. Since the arcs going into the sets $\mathbf{\Xi}_-$ and $\mathbf{\Xi}_+$ are chosen such that a great number of possible, but futile exchanges are excluded, the presented Local Search Algorithm can afford to perform computationally more expensive 5-exchanges without exceeding the available computational resources. The algorithm is of complexity $\mathcal{O}(n^{2.2})$ and finds globally optimal solutions for all to optimality solved instances of the traveling salesman problem up to 13509 locations. Furthermore, the interested reader finds an extensive report on Local Search in [44].

Finally, it shall be mentioned that high-performance algorithms for the solution of combinatorial optimization problems, such as the traveling salesman problem, are frequently developed as *hybrid algorithms*. First, one makes use of the excellent global search ability of Evolutionary or Ant Colony Algorithms to generate good start solutions, which are refined using Local Search, see [24, 25].

When tackling NP-hard combinatorial optimization problems, such as the traveling salesman problem, it is of vital importance to keep the problem size n at a minimum. This is achieved utilizing a *Receding Horizon Concept*, originally proposed for optimal control problems, such as model predictive control, see [1]. For this concept to be applicable, the problem needs to exhibit a dedicated direction of propagation. In air traffic management, for example, optimal aircraft arrival sequencing and scheduling is key to maximum capacity utilization of the runways. Genetic and Ant Colony Algorithms are frequently used to solve this NP-hard combinatorial optimization problem. Instead of optimizing the landing sequence for the entire planning period, e.g., an entire day, at once, it is advantageous to partition the planning period into several so-called optimization horizons. Each of these smaller subproblems is then optimized. However, only the first few planes of the current optimization horizon are assigned to land, before the optimization horizon is shifted forward in the direction of propagation, i.e. time, and the next subproblem is solved. This problem is extensively studied in [51, 52, 53, 96]. Additional applications of the Receding Horizon Concept to

combinatorial optimization problems can be found in commercial planning, see [16]. In the following, the path planning problem is formulated and two different solution strategies are presented.

3.2.2 Problem formulation

The result of the patch placement algorithm is a patch list $\mathbf{\Pi} = \{\mathbf{P}_i\}$, $i = 1, \ldots, N$, i.e. a list of patch locations $\mathbf{p}_{i,k}$ the robot must approach. Details regarding this notation are described in Section 3.1.2. To keep the notation general, the patch locations are referred to as nodes $\mathbf{x}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^{\mathrm{T}}$ and merged into a node list $\mathbf{X} = \{\mathbf{x}_i\}$, $i = 1, \ldots, n$, $n = \sum_{k=1}^{N} n_{\mathbf{P}_i}$, where N is the number of defects of the panel side under consideration and $n_{\mathbf{P}_i}$ is the number of patches required to cover defect $\mathbf{D}_i$.

Patch clustering Since the traveling salesman problem is NP-hard, the computation time of the algorithm can be reduced greatly by keeping the problem size, that is the number of nodes n, at a minimum. For this specific application, the following measure can be taken. Suppose a panel exhibits eight small and two big defects, which makes a total of $N = 10$ defects. Each of these big defects requires three patches. So, there are in total $n = 14$ patches to be placed. Naturally, the patches covering the big defects are very close to each other. So, by clustering the patches according to their respective defect, the problem size decreases from $n = 14$ to $\bar{n} = N = 10$. Thus, the path planning algorithm is only applied to the reduced set $\mathbf{X} = \bar{\mathbf{\Pi}} = \{\mathbf{p}_{i,1}\}$, $i = 1, \ldots, N$ instead of the set $\mathbf{X} = \mathbf{\Pi} = \{\mathbf{P}_i\}$ with $\mathbf{P}_i = \{\mathbf{p}_{i,k}\}$, $k = 1, \ldots, n_{\mathbf{P}_i}$. This simple adjustment in the problem formulation considerably lowers the computation time, while at the same time the path costs remain virtually the same. In the following, unless explicitly mentioned otherwise, a general path planning problem $\mathbf{X}$ of size n is considered.

Cost matrix The path planning algorithm aims at computing the minimum cost path between a number n of nodes. In view of maximizing throughput, the goal is to determine the time optimal path. Therefore, the costs represent the time it takes the patching robot to move from node $\mathbf{x}_i$ to node $\mathbf{x}_j$, given by a function $c(\mathbf{x}_i, \mathbf{x}_j)$. As is frequently the case in xy-positioning, x- and y-movement of the patching robot are independent and simultaneous. Thus, the costs are defined as

$$c_{i,j} = c(\mathbf{x}_i, \mathbf{x}_j) = \max(\bar{T}_{x_{i,j}}, \bar{T}_{y_{i,j}}), \tag{3.5}$$

with $\bar{T}_{x_{i,j}}$ and $\bar{T}_{y_{i,j}}$ being the travel times[4] in longitudinal and lateral direction, respectively. They are computed between each pair of nodes yielding the cost

[4]The travel times depend on the specific robot trajectory, see (3.38).

matrix $\mathbf{C} = [c_{i,j}] \in \mathbb{R}^{n \times n}$. It holds the information about the traveling costs from any node to any other node. The diagonal elements of this matrix are zero, i.e. $c_{i,i} = 0$, $i = 1, \ldots, n$. Furthermore, the cost matrix is symmetric, $\mathbf{C} = \mathbf{C}^{\mathrm{T}}$, if the dynamics of the actuators of the patching robot in forward and backward direction are equal. In this case, the path planning problem is called symmetric.

Solution representation and quality criterion Let us consider a set of nodes $\mathbf{X}$ that is fully connected by a set of arcs $\boldsymbol{\Xi}$. Thus, the pair $G = (\mathbf{X}, \boldsymbol{\Xi})$ is a complete graph, i.e. every node $\mathbf{x}_i \in \mathbf{X}$ is directly connected to every other node $\mathbf{x}_j \in \mathbf{X}$. Each arc $\xi_{i,j} \in \boldsymbol{\Xi}$ is now associated with its respective cost $c_{i,j}$. The path planning problem is the problem of finding a minimum cost path on G from a previously specified start node, the one with the lowest x-coordinate, to an end node, the one with the highest x-coordinate, visiting all nodes exactly once. Therefore, the path vector can be defined as

$$\boldsymbol{\psi} = \begin{bmatrix} \psi_1 & \ldots & \psi_i & \ldots & \psi_n \end{bmatrix}, \; i = 2, \ldots, n-1 \tag{3.6a}$$

$$\psi_i \in \{1, \ldots, n\} \setminus \{\psi_1, \ldots, \psi_{i-1}, \psi_n\} \tag{3.6b}$$

with

$$\psi_1 = k, \text{ where } \min_k (x_k), \tag{3.6c}$$

$$\psi_n = k, \text{ where } \max_k (x_k). \tag{3.6d}$$

The path cost is the sum of the arc costs of the respective path $\boldsymbol{\psi}$ through the graph,

$$J = \sum_{i=1}^{n-1} (c_{k,l})|_{k=\psi_i, l=\psi_{i+1}} = \sum_{i=1}^{n-1} c(\mathbf{x}_{\psi_i}, \mathbf{x}_{\psi_{i+1}}). \tag{3.7}$$

A short remark shall clarify the notation introduced in (3.7): $\mathbf{x}_i$ is a generic node of the path planning problem. The notation $\mathbf{x}_{\psi_i}$ indicates that node $\mathbf{x}_{\psi_i}$ is assigned position i in the path $\boldsymbol{\psi}$. For example, $\mathbf{x}_{\psi_5}$ where $\psi_5 = 3$ means that $\mathbf{x}_3$ is fifth in $\boldsymbol{\psi}$.

Left-To-Right and Nearest-Neighbor Path Considering the panels elongated shape and the dedicated production flow, as described in Figure 2.4, an intuitive way to construct reasonable solutions is to sort the nodes $\mathbf{x}_i$ in ascending x-direction

$$\boldsymbol{\psi}^{l2r} = \begin{bmatrix} \psi_1 & \ldots & \psi_i & \ldots & \psi_n \end{bmatrix}, \; x_{\psi_{i-1}} \leq x_{\psi_i}, \; i = 2, \ldots, n. \tag{3.8}$$

It is called the from-left-to-right path. Another way of solution construction is the nearest-neighbor path, where the path proceeds from the current node to its

nearest neighbor that has not been visited yet until all nodes are visited,

$$\boldsymbol{\psi}^{nn} = \begin{bmatrix} \psi_1 & \dots & \psi_i & \dots & \psi_n \end{bmatrix}, \tag{3.9a}$$

$$\psi_{i+1} = j, \text{ where } \min_j c_{\psi_i,j}, \tag{3.9b}$$

$$j \in \{1, \dots, n\} \setminus \{\psi_1, \dots, \psi_i, \psi_n\}, \; i = 1, \dots, n-2. \tag{3.9c}$$

Note that the nearest-neighbor path starting from the left, in general, does not match the nearest-neighbor path starting from the right. These very simple solutions for the path planning problem at hand can be used to initialize more sophisticated algorithms.

In the following, two solutions to the path planning problem are proposed: First, an Ant Colony Algorithm which is recognized as an efficient approach to tackle large problem instances of the general traveling salesman problem is described. Second, a Local Search Algorithm combined with a Receding Horizon Concept presents a problem-tailored and simple solution to the specific traveling salesman problem at hand.

3.2.3 Ant Colony Algorithm

The objective function (3.7) can be minimized by means of an Ant Colony Algorithm. The foraging behavior of ants is determined by pheromone trails they deposit on their way between food and nest. Ants tend to follow these trails, while at the same time reinforcing them. Thus, the pheromone concentration serves as the ant colony's collective memory, which develops over time. The desirability of various paths is remembered and visible for other ants. For more detailed information, the reader is referred to, e.g., [23, 24]. In the following, only the core functionality of all Ant Colony Algorithms, i.e. the iterative, cooperative solution construction, shall be outlined.

In every iteration $\theta = 1, \dots, \Theta$, each ant $m \in \{1, \dots, M\}$ of a population of size M individually constructs feasible paths by making $h = 1, \dots, n-2$ random decisions[5] based on (3.10). The probability that an ant m decides to move from its current node i to node j is

$$p_{i,j}^m(\theta) = \begin{cases} \frac{\tau_{i,j}^{\alpha}(\theta)\eta_{i,j}^{\beta}}{\sum_{l \in \mathcal{N}_h^m} \tau_{i,l}^{\alpha}(\theta)\eta_{i,l}^{\beta}} & j \in \mathcal{N}_h^m \\ 0 & \text{otherwise,} \end{cases} \tag{3.10}$$

where $\mathcal{N}_h^m$ is the so-called feasible neighborhood[6] and $\tau_{i,j}(\theta)$ and

$$\eta_{i,j} = 1/c_{i,j} \tag{3.11}$$

[5] Start and end node are already fixed.

[6] Since G is a complete graph, the feasible neighborhood does not depend on the current node i, but only on the previously visited nodes, see (3.15).

denote the pheromone trail value and the heuristic information value on the arc $\xi_{i,j}$, respectively. The tuning parameters $\alpha \in \mathbb{N}$ and $\beta \in \mathbb{N}$ are used to scale the weight of the heuristic and the pheromone information relatively to each other.

The heuristic value $\eta_{i,j}$ represents a priori information on the problem, which is collected in a matrix $\mathbf{H} = [\eta_{i,j}] \in \mathbb{R}^{n \times n}$ similarly to $\mathbf{C}$. The costlier an arc, the less desirable it is for an ant to use. The pheromone value $\tau_{i,j}(\theta)$, on the contrary, represents learned desirability to use that arc. Thus, the pheromone matrix $\mathbf{T}(\theta) = [\tau_{i,j}(\theta)] \in \mathbb{R}^{n \times n}$, being the collective memory of all ants, changes over time as ants deposit pheromone on their paths. This is accomplished by the following pheromone update laws. First, a portion $\rho \in [0, 1]$ of pheromone evaporates

$$\tau_{i,j}(\theta + 1) = (1 - \rho)\, \tau_{i,j}(\theta) \tag{3.12}$$

and second, each ant m deposits pheromone along its path

$$\tau_{\psi_i^m, \psi_{i+1}^m}(\theta + 1) = \tau_{\psi_i^m, \psi_{i+1}^m}(\theta) + \frac{1}{J^m(\theta)},\ i = 1, \ldots, n - 1, \tag{3.13}$$

with $\boldsymbol{\psi}^m$ and J^m according to (3.6) and (3.7), respectively. Pheromone evaporation causes costly paths to be forgotten, while pheromone deposition enables ants to share their experience via the collective memory.

Finally, the feasibility of the ants' paths has to be discussed. To guarantee that ants visit each node exactly once, each ant m is given a local memory

$$\mathcal{M}_h^m = \{\psi_1^m, \ldots, \psi_h^m, \psi_n^m\} \tag{3.14}$$

with $\psi_1^m = \psi_1$ and $\psi_n^m = \psi_n$ according to (3.6). It contains all the nodes already visited up to iteration h of the construction process and thus updates the feasible neighborhood,

$$\mathcal{N}_h^m = \{1, \ldots, n\} \setminus \mathcal{M}_h^m. \tag{3.15}$$

The completeness of the graph G makes each node a neighbor of every other node, i.e. each node can be reached from any other node by crossing only a single arc. Thereby it is ensured that ants cannot get stuck at a node[7].

Initialization and choice of the parameter set The initialization of the pheromone matrix affects search space exploration and solution convergence, see [24]. On the one hand, if the initial pheromone values are too low, then the search is quickly biased by the first tours generated by the ants. Frequently, the consequence is convergence to a local optimum. On the other side, if the

[7] If G was not complete, it might happen that the feasible neighborhood $\mathcal{N}_h^m = \{\}$ before the construction process is finished.

α	β	ρ	M	Θ	ε
1	5	0.2	n	70	1

Table 3.1: Parameters of the implemented Ant Colony Algorithm.

initial pheromone values are too high, then many iterations are lost waiting until pheromone evaporation reduces enough pheromone values, so that pheromone added by ants actually starts to bias the search. A good way to initialize the pheromone matrix $\mathbf{T}(0) = [\tau_0]$ is to set its entries to a value slightly higher than the expected amount of pheromone deposited by the ants in one iteration, i.e.

$$\tau_{i,j}(0) = \tau_0 = M/J^0,\ J^0 = \min(J^{l2r}, J^{nn}), \tag{3.16}$$

where J^{l2r} and J^{nn} are the path costs (3.7) of the from-left-to-right path (3.8) and the nearest-neighbor path (3.9), respectively.

The parameter set for the implemented Ant Colony Algorithm is given in Table 3.1. It was chosen experimentally, which resulted in similar values as proposed in the literature, e.g., [24]. More importance is attached to rapid convergence compared to solution space exploration. To achieve this, the evaporation parameter ρ is set to a small value and the concept of an elitist ant is introduced, see [24].

Elitist Ant The elitist ant, denoted as $(\cdot)^\varepsilon$, holds the best-so-far path $\boldsymbol{\psi}^\varepsilon$, i.e. the best path found since the start of the algorithm. This provides the possibility to stop the algorithm any time, still yielding a useful result. Additionally, the elitist ant deposits pheromone in every iteration of the algorithm,

$$\tau_{\psi_i^\varepsilon,\psi_{i+1}^\varepsilon}(\theta+1) = \tau_{\psi_i^\varepsilon,\psi_{i+1}^\varepsilon}(\theta) + \frac{\varepsilon}{J^\varepsilon(\theta)},\ i = 1,\dots,n-1, \tag{3.17}$$

where $\varepsilon \in \mathbb{R}_+$. With growing ε, this provides increasingly strong reinforcement of the best-so-far path.

By the time the algorithm terminates, a number of $M\,\Theta$ paths were constructed, the best of which is $\boldsymbol{\psi}^* = \boldsymbol{\psi}^\varepsilon$. Applying the Ant Colony Algorithm to the clustered patch list $\bar{\mathbf{\Pi}}$ yields $\bar{\mathbf{\Pi}}^*$, which is now sorted according to the time optimal path $\boldsymbol{\psi}^*$. The clustered nodes that were not considered in the path planning algorithm have to be inserted accordingly to receive the optimal patch list $\mathbf{\Pi}^*$.

Start solution Taking into account the flow of the rectangular, elongated shape of the shuttering panels through the production line, it is beneficial to add a from-left-to-right path (3.8) and a nearest-neighbor path (3.9) into the set of start solutions. The risk with this approach is that in the early iterations of the

algorithm these good start solutions might build up very high pheromone values due the reinforcement by the elitist ant. This results in a monodirectional search, ignoring other promising solutions. However, this can be avoided by decreasing the parameter ε in (3.17).

Implementation In Algorithm 3, the main steps of the Ant Colony Algorithm are given in pseudo code. Input `xy` of the algorithm are the xy-coordinates of the clustered patch list $\bar{\mathbf{\Pi}}$. The output `xyOpt` contains the same nodes, but sorted according to their optimal processing sequence $\bar{\mathbf{\Pi}}^*$. Additionally, the optimal path costs `jOpt` are available. To initialize the algorithm, the cost matrix `C` and, based on it, the start solutions `pL2R` and `pNN` need to be computed according to (3.5), (3.8) and (3.9), respectively. The better one of the start solutions is chosen as the initial best-so-far path `pBSF`. Then, the heuristic Matrix `H` (3.11) is computed and the pheromone matrix `T` (3.16) is initialized.

Now, the iterative, cooperative solution construction begins. In every iteration θ, denoted `t` in the code, `M` ants[8] individually construct paths in the function `constructPath`. To do that, every ant needs to make $n-2$ random decisions (3.10). This formula needs to be evaluated $n-2$ times by M ants in every iteration `t` in the code. Two measures can be taken to speed that up. First, the numerator of (3.10) remains constant throughout an iteration `t` and can, therefore, be computed in advance and stored in a matrix `I`. Second, the *Roulette Wheel Selection Procedure*, see [24, 31], is a computationally efficient implementation of (3.10). After every ant finished its path construction, the best-so-far path `pBSF` is updated. Then, the pheromone values are updated by evaporation (3.12), deposition (3.13) and additional deposition of the elitist ant (3.17).

After the last iteration `tMax`, the best-so-far path is declared the optimal path and the patch list is sorted accordingly.

3.2.4 Local Search Receding Horizon Algorithm

The core of this algorithm is Local Search, which is based on the concept of λ-optimality. It was initially proposed in [65], see also [43, 45]. Then, the Local Search Algorithm is combined with a Receding Horizon Concept, which, for instance, is used in air traffic management, see [51, 52, 53, 96].

Local Search Algorithm based on 3-optimality The concept of λ-optimality is defined as follows: A path is said to be λ-optimal, if it is impossible to obtain a tour with smaller cost by replacing any λ of its arcs $\xi_{i,j}$, a set denoted $\mathbf{\Xi}_{-}$, by

[8] Using an object-oriented programming language, such as C++, the software agent ant would be implemented as a specific class containing all the required data structures and functions. This design translates the idea of an independent ant into software and thus makes the code more readable as well as robust, since access restrictions of classes prevent inadmissible operations.

Algorithm 3 Ant Colony Algorithm

```
function [xyOpt, jOpt] = AntColony(xy)
//initialization
C = computeCostMatrix(xy); //(3.5)
pL2R = constructLeft2RightPath(xy); //(3.8)
pNN  = constructNearestNeighborPath(C); //(3.9)
jL2R = pathCost(pL2R); jNN = pathCost(pNN); //(3.7)

if( jL2R<jNN ) { //best-so-far path, elitist ant
   pBSF = pL2R; jBSF = jL2R;
} else {
   pBSF = pNN; jBSF = jNN;
}

H = computeHeuristicMatrix(C); //(3.11)
T = initializePheromoneMatrix(jBSF,M); //(3.16)

//solution construction
for( t=1; t<=tMax; t++ ) {
   //numerator of (3.10), '.' inidicates elementwise operation
   I = T.^alpha .* H.^beta;

   //ants construct paths
   for( m=1; m<=M; m++ ) {
      ant_mem[m] = resetLocalMemory(); //(3.14)
      ant_path[m] = constructPath(I, ant_mem[m]);
      ant_cost[m] = pathCost(C, ant_path[m]);
   }

   //update best-so-far path, elitist ant
   [jMin, index_jMin] = min(ant_cost);
   if( jBSF>jMin ) {
      pBSF = ant_path[index_jMin];
      jBSF = ant_cost[index_jMin];
   }

   //pheromone update
   T = evaporation(T); //(3.12)
   T = deposition(T); //(3.13)
   T = depositionEA(T); //(3.17)
}

//optimal path, cost, sort nodes according to pOpt
pOpt = pBSF; jOpt = jBSF;
xyOpt = sort(xy, pOpt);
```

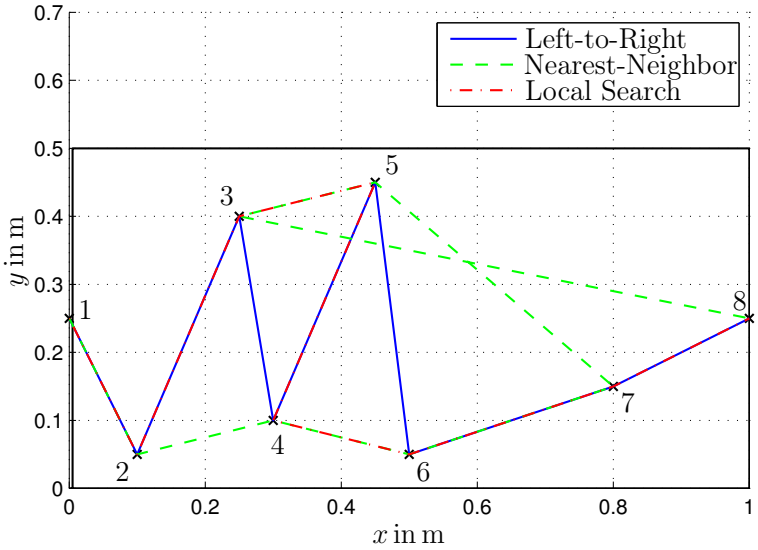

Figure 3.5: Two start solutions refined by local 3-exchanges.

any other set of λ arcs denoted $\boldsymbol{\Xi}_+$[9]. Consequently, an n-optimal tour, i.e. $\lambda = n$, is globally optimal. So by increasing λ, the problem gets more general, in the sense that the number of possible exchanges increases. This way, one receives increasingly strong necessary conditions for optimality, but it comes at the cost of increasing computational effort. Computer experiments showed that setting $\lambda = 3$ is the most efficient trade-off between solution quality and computational costs, see [65]. By means of 3-exchanges, it is possible to realize path inversions and insertions, see Figure 3.5. In this example, two initial solutions are given. Both, the from-left-to-right path (blue) and the nearest-neighbor path (green) are improved using Local Search, yielding the same optimal path (red). Optimization of the from-left-to-right path is achieved by replacing $\boldsymbol{\Xi}_- = \{\xi_{3,4}, \xi_{4,5}, \xi_{5,6}\}$ by $\boldsymbol{\Xi}_+ = \{\xi_{3,5}, \xi_{4,5}, \xi_{4,6}\}$. As can be seen, a 2-exchange realizes a path inversion. Replacing the arcs $\boldsymbol{\Xi}_- = \{\xi_{2,4}, \xi_{3,8}, \xi_{5,7}\}$ of the nearest-neighbor path by $\boldsymbol{\Xi}_+ = \{\xi_{2,3}, \xi_{4,5}, \xi_{7,8}\}$ realizes a path insertion by means of a 3-exchange. The nodes $\mathbf{x}_3$ and $\mathbf{x}_5$ are taken out and inserted between the nodes $\mathbf{x}_2$ and $\mathbf{x}_4$. This simple example already demonstrates the concept of the Local Search Algorithm. The algorithm begins by generating start solutions, see Section 3.2.2. Each of these solutions is then refined by randomly performing 3-exchanges until no further improvement can be achieved or until the maximum number of algorithm iterations $I = 4n$, where n is the problem size, is reached. The best one is assumed to be the optimal path $\boldsymbol{\psi}^*$. Details on how to efficiently implement the Local Search can be found in [65].

[9]Note that a subset $\tilde{\lambda} \leq \lambda$ of the arcs may be contained in both sets and thus remains the same.

Receding Horizon Concept Since the traveling salesman problem is NP-hard, the minimization of the problem size n reduces the computation time greatly. To achieve that, a specific solution strategy called Local Search Receding Horizon Algorithm is developed that exploits the features of the path planning problem at hand, see [47, 48]. The Receding Horizon Concept is based on the fact that a partial path is hardly ever influenced by nodes very far from it, in particular if there are several other nodes in between. Therefore, a piecewise local optimization process is applied in an iterative manner.

The entire problem is divided into several subproblems of size $h_o \ll n$. Each of these is then solved using the Local Search Algorithm. However, only the first $h_i < h_o$ nodes are actually appended to the final optimal path. The remaining nodes are close to the following nodes that were disregarded in the current subproblem. This is why, their order is most likely not optimal yet and, therefore, prone to change in the next subproblem. Still, the remaining $h_o - h_i$ nodes serve as a high quality start solution for the next subproblem and thus reduce the computational costs. In this context, h_o and h_i are referred to as optimization and implementation horizon, respectively.

To initialize the algorithm, the from-left-to-right path is computed $\boldsymbol{\psi}^{l2r} = \begin{bmatrix}\psi_1^{l2r} & \dots & \psi_n^{l2r}\end{bmatrix}$, see (3.8). In the following paragraph, the next iteration step $k+1$ of the Local Search Receding Horizon Algorithm is described based on the results of the current step k.

Given the optimal final path $\boldsymbol{\psi}^{(k),*} = \begin{bmatrix}\psi_1^* & \dots & \psi_{1+h_i k}^*\end{bmatrix}$ of iteration step k, in the next iteration step $k+1$, a partial path

$$\Delta\boldsymbol{\psi}^{(k+1)} = \begin{bmatrix}\psi_{2+h_i k}^{(k),*} & \dots & \psi_{1+h_i(k-1)+h_o}^{(k),*} & \psi_{2+h_i(k-1)+h_o}^{l2r} & \dots & \psi_{1+h_i k+h_o}^{l2r}\end{bmatrix} \tag{3.18}$$

of size h_o is optimized using the Local Search Algorithm, considering a fixed start node $\psi_{1+h_i k}^*$ and end node $\psi_{2+h_i k+h_o}^{l2r}$. After the optimization is performed, the optimal partial path is given by

$$\Delta\boldsymbol{\psi}^{(k+1),*} = \begin{bmatrix}\psi_{2+h_i k}^{(k+1),*} & \dots & \psi_{1+h_i k+h_o}^{(k+1),*}\end{bmatrix}. \tag{3.19}$$

Then, the first h_i nodes of (3.19) are appended to the optimal final path,

$$\boldsymbol{\psi}^{(k+1),*} = \begin{bmatrix}\psi_1^* & \dots & \psi_{1+h_i k}^* & \psi_{2+h_i k}^{(k+1),*} & \dots & \psi_{1+h_i(k+1)}^{(k+1),*}\end{bmatrix}. \tag{3.20}$$

Figure 3.6 illustrates this process. The hatched nodes represent the partial path $\Delta\boldsymbol{\psi}^{(k+1),*}$ under consideration in iteration step $k+1$. It is of length h_o and the superscript $(\cdot)^*$ indicates it has already been optimized. Now, the first h_i nodes of $\Delta\boldsymbol{\psi}^{(k+1),*}$ are appended to the final optimal path of the previous iteration step k to form the current optimal final path $\boldsymbol{\psi}^{(k+1),*}$, i.e. the white nodes. The black nodes are the remaining nodes which have not been optimized yet.

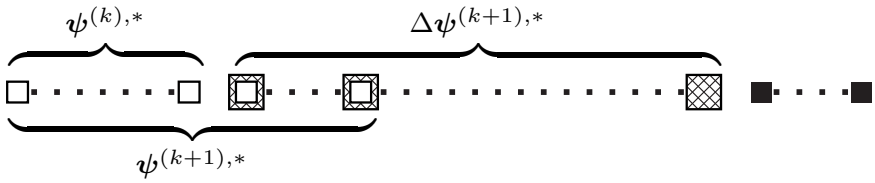

Figure 3.6: Illustration of the Receding Horizon Concept.

This procedure is carried out until the optimization horizon moved from the left side of the panel all the way to the right side, i.e.

$$1 + h_i k + h_o \geq n - 1. \tag{3.21}$$

The last optimization horizon might be smaller than h_o and is entirely appended to the final optimal path. Then, $\boldsymbol{\psi}^* = \boldsymbol{\psi}^{(k),*}$ is final.

In the initial optimization step $k = 0$, the optimal final path is $\boldsymbol{\psi}^{(0),*} = \begin{bmatrix}\psi_1^{l2r}\end{bmatrix}$ and the subproblem to be optimized is $\Delta\boldsymbol{\psi}^{(0)} = \begin{bmatrix}\psi_2^{l2r} & \dots & \psi_{1+h_o}^{l2r}\end{bmatrix}$.

Finally, the patch list $\bar{\mathbf{\Pi}}$ is sorted according to $\boldsymbol{\psi}^*$, denoted as $\bar{\mathbf{\Pi}}^*$. The clustered nodes that were not considered in the path planning algorithm have to be inserted accordingly to receive the optimal patch list $\mathbf{\Pi}^*$. The optimization horizon is set to $h_o = 5h_i$ with the implementation horizon being $h_i = 4$.

Implementation Algorithm 4 outlines the main functionality of the Local Search Receding Horizon Algorithm in pseudo code. Input `xy` of the algorithm are the xy-coordinates of the clustered patch list $\bar{\mathbf{\Pi}}$. For the application of the Receding Horizon Concept, the list needs to be sorted in ascending x-direction. The output `xyOpt` contains the same nodes, but sorted according to their optimal processing sequence $\bar{\mathbf{\Pi}}^*$. Additionally, the optimal path costs `jOpt` are available. To initialize the algorithm, the cost matrix `C` is computed. Since the nodes are already sorted in ascending x-direction, the initial guess of the optimal path `pOpt` is obvious, `pOpt[i] = i`. Each subproblem is defined by its start `iS` and end `iE` index. In case the entire problem instance of size `n` is smaller than the optimization horizon `ho+2`, `iE` has to be adjusted. The start and end node of each optimization horizon are not changed. They are boundary conditions of the optimization problem.

Solving the subproblems is a three-step procedure. First, the subproblem of size `iE-iS+1` is taken out of the entire problem and assigned to separate variables. The subproblem is defined by a partial path `pSub` and the respective part of the cost matrix `CSub`. Locally, i.e. in the context of the subproblem, the initial path is numbered in ascending order, `pSub_lcl`. In order to maintain reference to the overall, i.e. the global, problem, the respective nodes need to be remembered in `pSub_glb`. By this, in the first iteration, the start solution is the from-left-to-right path, while in the following it corresponds to (3.18). Second, the initial solution `pSub_lcl` is locally refined using Local Search. The Local Search Algorithm is

Algorithm 4 Local Search Receding Horizon Algorithm

```
function [xyOpt, jOpt] = RecedingHorizon(xy) //xyL2R
//initialization
C = computeCostMatrix(xy); //(3.5)
for( i=1; i<=n; i++ ) { pOpt[i] = i; } //initial guess = L2R-path

iS = 1; //start/end index of optimization horizon
if( ho+2>n ) { iE = n; } else { iE = ho+2; } //start, end node fixed

//solution of subproblems
while (true) {
   //generate subproblem
   for( i=iS; i<=iE; i++ ) { pSub_glb[i-iS+1] = pOpt[i]; }
   for( i=1; i<=iE-iS+1; i++ ) { pSub_lcl[i] = i; }

   //take out corresponding part of cost matrix
   for( i=iS; i<=iE; i++ ) {
      for( j=i; j<=iE; j++ ) {
         CSub[i-iS+1,j-iS+1] = C[pOpt[i],pOpt[j]];
         CSub[j-iS+1,i-iS+1] = CSub[i-iS+1,j-iS+1]; //C = C'
      }
   }

   //hand over subproblem to Local Search
   pSub_lclOpt = localSearch(CSub, pSub_lcl);

   //rearrange optimal path
   for( i=iS+1; i<=iE-1; i++ ) { //start, end node fixed
      pOpt[i] = pSub_glb[pSub_lclOpt[i-iS+1]];
   }

   //end of optimization
   if( iE>=n ) { break; }

   //shift optimization horizon forward
   iS = iS+iH; if( iS>n ) { iS = n; }
   iE = iE+iH; if( iE>n ) { iE = n; }
}

//generate output
jOpt = pathCost(C, pOpt);
xyOpt = sort(xy, pOpt);
```

basically implemented as described in [65]. The main difference is that the start $\psi^{(k),*}_{1+h_i k}$ and end $\psi^{l2r}_{2+h_i k+h_o}$ node of the subproblem remain unchanged. Third, the optimal solution `pSub_lclOpt` of the subproblem, see (3.19), updates the current optimal path `pOpt`, see (3.20). Note that the entire solution of the subproblem is taken. Thus, the next subproblem in line 12 and 13 is automatically initialized with a good start solution, see (3.18). However, the optimization horizon is only shifted forward by `iH`.

A soon as the optimization horizon reaches the end of the panel `iE>=n`, compare (3.21), all the nodes of the problem instance have been considered and, therefore, the algorithm terminates and generates the final output.

3.2.5 Results and analysis

Three examples demonstrate and compare the capabilities of the presented Ant Colony and Local Search Receding Horizon Algorithm. In the *first example*, the cost of each arc corresponds to the Euclidean distance between the respective two nodes. Figure 3.7 depicts a 1.5m long panel exhibiting 24 small and three big defects (at approximately $x = 0.4$m, 0.8m, 0.9m). The small defects are covered by one patch, whereas the big ones require three patches each. For the path planning algorithm, the respective patches are clustered and considered as one node. The clustering is indicated by the black lines connecting the nodes.

The solution of the simplest possible path planning method from-left-to-right is compared to the results of the Ant Colony Algorithm and the Local Search Receding Horizon Algorithm as presented in Sections 3.2.3 and 3.2.4, respectively. Although the from-left-to-right path already yields quite good results, one can easily identify its weakness. If nodes are very close in x-direction, but far from each other in y-direction, this method generates solutions that permanently jump up and down along the y-axis. Thus, this method becomes less effective as the ratio L/W between panel length L and width W decreases.

Within a computation time of 5ms, the Ant Colony Algorithm manages to improve the start solution from $J^{l2r} = 4.6$m to $J^{aco} = 3.7$m. In comparison, the Local Search Receding Horizon Algorithm finds a path of length $J^{lsrh} = 3.5$m in only 1ms. The reason for the high performance of the Local Search Receding Horizon Algorithm is that it makes perfect use of the peculiarities of the traveling salesman problem at hand. First, there is a dedicated production flow, and second, the panels are much longer than wide. So, it is easy to create good start solutions which only require local refinement[10]. The Local Search Receding Horizon Algorithm is perfectly suited to correct these errors.

The *second example* focuses on the Receding Horizon Concept. In order to

[10]It is worth mentioning that the most suitable start solution for the Local Search Algorithm is the nearest-neighbor path. Usually, the nearest-neighbor path is extremely good except for a few nodes that are left behind, see Figure 3.5.

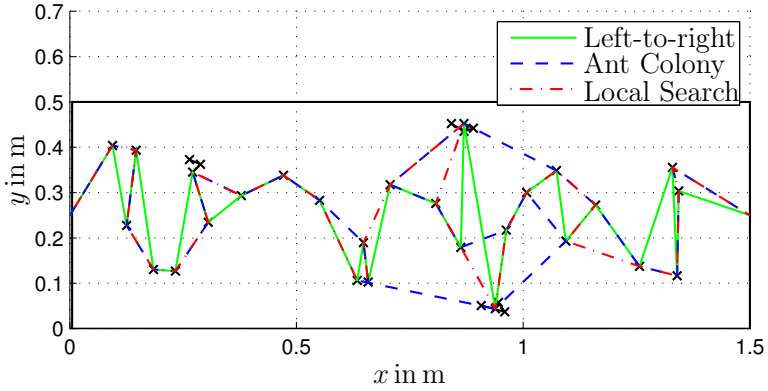

Figure 3.7: Results of three path planning strategies for an exemplary panel.

point out its advantages in this particular path planning problem, a large scale problem, i.e. a panel of 50m × 0.5m exhibiting $n = 1000$ randomly distributed nodes is considered. The from-left-to-right path takes 0.012s to compute and is $J_{l2r} = 185.1$m long. After a computation time of only 0.050s the Local Search Receding Horizon Algorithm yields a path of length $J^{lsrh} = 128.0$m, i.e. an improvement of 30% over the from-left-to-right path. In comparision, the basic Local Search Algorithm takes 105.100s before it terminates because it hits the maximum number of iterations $I = 4n = 4000$. The computed path is $J^{ls} = 149.9$m long, i.e. an improvement of 20%.

Looking at a section of the panel, as depicted in Figure 3.8, reveals the reason for the inferiority of the solution of the Local Search Algorithm. While between $x = 20.3$m and $x = 22$m the Local Search Algorithm and the Local Search Receding Horizon Algorithm compute paths of almost identical quality, the partial path of the Local Search Algorithm between $x = 20$m and $x = 20.3$m is identical to the from-left-to-right path. Some parts of the from-left-to-right path were not even optimized before the Local Search Algorithm reached the maximum number of iterations. As mentioned in [43], the choice of the sets $\boldsymbol{\Xi}_-$ and $\boldsymbol{\Xi}_+$ on which the local exchange is performed is vital for the performance of the algorithm. The optimization horizon greatly restricts this choice by disregarding all the nodes beyond it. Thus, only exchanges with a reasonable chance of improving the path are executed, while a great amount of futile exchanges is prevented. In a way, the optimization horizon works like the candidate sets presented in the algorithm of [43].

The *third example* is taken from real measurement data acquired at the pilot patching plant. Figure 3.9 depicts a 2m-panel exhibiting nine defects that require twelve patches to be covered. Also, the start and end positions as well as two positions the robot must approach for process logic control are marked with a cross. The positions at $x = 0.375$m and $x = 1.642$m need to be approached for the transition from XY-Table 1 moving the panel alone into both xy-tables

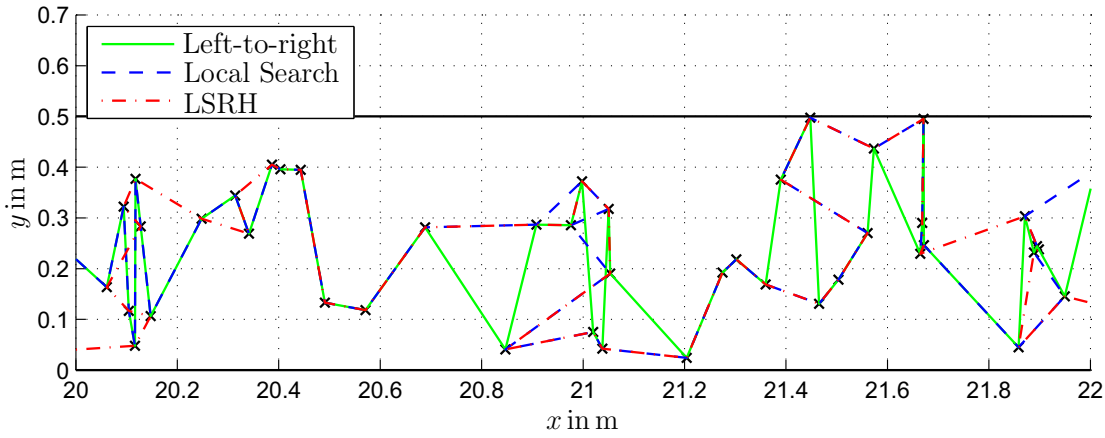

Figure 3.8: A 50m-panel requiring $n = 1000$ patches, section between 20m and 22m.

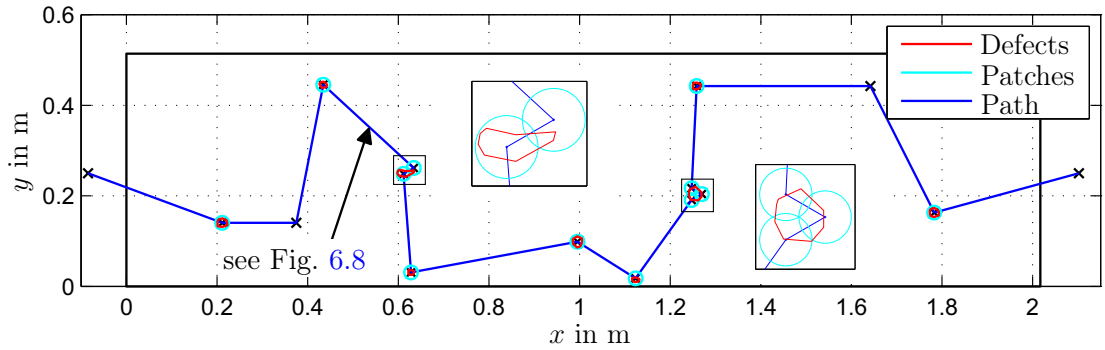

Figure 3.9: Results of the optimization algorithms for a typical shuttering panel obtained at the pilot patching plant.

moving the panel together on to XY-Table 2 moving the panel alone. These nodes are connected with the time optimal path computed by the Local Search Receding Horizon Algorithm.

In the considered application, hard real-time constraints have to be met. Therefore, the possibility to simply stop the algorithm after a given time is a vital backup system. Both, the Ant Colony Algorithm and the Local Search Receding Horizon Algorithm can be terminated any time, since they find good paths very quickly and the best-so-far path is remembered. The drawback of heuristic methods is that optimality of the computed path cannot be proven. Nevertheless, the above mentioned advantages make heuristic methods, in particular the Local Search Receding Horizon Algorithm, the method of choice. The Local Search Receding Horizon Algorithm outperforms the Ant Colony Algorithm because it is tailored to the path planning problem at hand.

Regarding general applicability, it should be noted that these algorithms can be applied to any robot design as nothing, but the cost matrix would change.

Going beyond the traveling salesman problem, both of these algorithms are applicable to any problem that can be stated in the form as described in [23]. Various examples for the application of Ant Colony Algorithms and Local Search Algorithms are given in [18, 27, 39]. The Receding Horizon Concept is restricted to problems exhibiting a distinguished direction of propagation, e.g., in the form of a dedicated process direction as in the application at hand or in a timely fashion as in air traffic management.

3.3 Trajectory generation

The path planning algorithm determines the time optimal processing sequence for the individual patch locations. In the next step, the robot motion between two consecutive patch locations needs to be planned. This motion is described by a trajectory, i.e. the path that the robot (or the panel) follows in the xy-plane given as a function of time. Four requirements are laid down for the choice and the design of the trajectory generator. First, given the patching robot design of Section 2.2, the motion of the panel is decoupled in x- and y-direction, i.e. the panel is moved in each direction independently. Consequently, only one dimensional motion planning is necessary. Second, throughput maximization being the goal, only time optimal trajectory generators are investigated. Third, during the patching action itself, the panel stands still, meaning each positioning action starts at zero initial velocity and acceleration. However, in Section 4.2.2, a control strategy called Trajectory Updating is presented. It requires a trajectory generator that is able to start at an arbitrary initial velocity and acceleration. Fourth, Trajectory Updating also needs the trajectory generator to be executable in real-time. To sum it up, the trajectory generator needs to compute a time optimal, one dimensional trajectory, starting at an arbitrary initial state, coming to a full halt at a given end position, in real-time.

3.3.1 Literature review

Time optimal trajectories fully exploit the dynamical capabilities of the patching robot, i.e. its maximum velocity, acceleration and jerk. However, application of excessive jerk and acceleration is likely to induce vibrations in the machine structure. This puts an unnecessary burden on the machine and, moreover, hampers precise positioning. The choice of sufficiently smooth trajectories is a measure to avoid these problems [8]. In mathematical terms, the desired position profile needs to be at least two times continuously differentiable, meaning the acceleration profile still needs to be continuous, while the jerk profile may already exhibit steps. Therefore, suitable trajectories are derived by three times integrating a piecewise constant jerk profile. Pursuing such an approach to generate time optimal trajectories yields the so-called bang-bang solution, an elementary result

of optimal control theory, see [63]. The jerk signal only assumes one of three values, either maximum or minimum or zero, such that always one of the jerk, acceleration or velocity constraints is active.

The main task is to compute a suitable jerk signal. This can either be accomplished in an offline or online fashion. *Offline trajectory generators* are based on the fact that such bang-bang trajectories can be divided into several phases. For instance, a continuously differentiable trajectory consists of an acceleration phase, a phase of constant velocity and a deceleration phase. A two times continuously differentiable trajectory consists of seven phases. Each of these phases is described by its respective jerk value and duration. The computation of the jerk value and the duration of the phases is achieved in a algorithmic way, taking into account the initial and desired end values of the integrators. In [8], such a two times continuously differentiable, time optimal trajectory generator is presented that considers generic initial and final values of velocity, while the initial and final acceleration values are set to zero. In [62], a three times continuously differentiable, time optimal trajectory generator is proposed. The resulting trajectories consist of fifteen phases. To manage this complexity, a systematic approach is adopted. The entire trajectory is broken down into subproblems, starting at the extremely simple computation of a continuous jerk profile. Based on this, a continuously differentiable acceleration profile is derived and so forth. However, some of the calculations require numerical solvers. Thus, the computational costs are relatively high.

Online trajectory generators typically constitute closed-loop systems. They consist of a chain of integrators, the first one of which receives the control input computed by a variable structure controller, which has the integrator states as inputs. In each time step, based on the current integrator states, this controller yields an input signal that drives the integrators towards their desired reference values in a time optimal way. Continuous time implementations with varying degrees of continuity of the position profile are presented in [33, 93, 94]. A time-discrete implementation, as is usually the case, may pose a problem if the sampling time is too large. Since the controller can only switch its input at discrete points in time, the bang-bang control input computed by the variable structure controller might result in oscillating trajectories. To overcome this drawback, the trajectory generator itself must be designed as a time-discrete system. Fundamental work for discrete time optimal online trajectory generators is presented in [60]. Considering a discrete linear time-invariant system, a control law is developed which drives the system from an arbitrary state to the origin of the state space in minimum time. It is based on the definition of an M-step admissible set, i.e. a subspace of the state space from which the origin can be reached in M time steps by applying an appropriate control input. In turn, this set yields the time optimal control input for a given initial point in the state space. This idea is adopted in [68, 95]. By means of this admissible set, it is possible to define a so-called boundary layer. As soon as the trajectory is in the vicinity of

the desired reference values, i.e. inside the boundary layer, the variable structure controller does not apply a bang-bang signal anymore, but makes use of the whole range available for the control input. Discrete time implementations with varying degrees of continuity are presented in [30, 35, 93]. Further papers [9, 34, 95] consider different kinds of constraints on the integrator states and the control input. A major advantage of the discussed online trajectory generators is that the constraints can be switched in real-time without impairing the stability of the closed-loop system and still yielding time optimal trajectories. On the downside, the transition time of the trajectory cannot be computed without simulating the entire trajectory.

In the following, a two times continuously differentiable, offline Bang-Bang Trajectory Generator is presented. It is able to start at an arbitrary initial system state, i.e. initial velocity and acceleration. Furthermore, a simplified point-to-point version is derived. Finally, this trajectory generator is compared to an online Bang-Bang Trajectory Generator. In Section A.1, a so-called Sine-Square Trajectory Generator is presented. It is based on the offline Bang-Bang Trajectory Generator, but the piecewise constant jerk input is replaced by smoother trigonometric functions. This yields a three times continuously differentiable position profile. In turn, time optimality is lost.

3.3.2 Bang-Bang Trajectory Generator

As an input the trajectory generator receives two consecutive patch locations $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ of the patch list $\mathbf{\Pi}^*$. To keep the notation general, the trajectory generator computes the trajectory between an initial $\mathbf{x}_0 = \begin{bmatrix} x_0 & y_0 \end{bmatrix}^{\mathrm{T}}$ and a desired position $\mathbf{x}_d = \begin{bmatrix} x_d & y_d \end{bmatrix}^{\mathrm{T}}$. Because the dynamics of the patching robot are decoupled in x- and y-direction, only the longitudinal case is considered.

In view of throughput optimization, the robot trajectories need to be time optimal, obeying the dynamical limitations of the patching robot, namely maximum velocity v_M, acceleration a_M and jerk j_M. Moreover, the position trajectory $x_t(t)$ needs to be sufficiently smooth, i.e. at least two times continuously differentiable, $x_t(t) \in \mathcal{C}^2(t)$. The trajectory updating strategy, presented in Section 4.2.2, requires the trajectory generator to be executable in real-time and it has to be able to start at any initial system state $\begin{bmatrix} x_0 & v_0 & a_0 \end{bmatrix}^{\mathrm{T}}$.

A trajectory generator that fulfills all these requirements is derived as a simplified version of [62], see also [47]. The trajectory is generated by three times integrating a jerk signal switching between minimum, zero and maximum, $j_t(t) \in \{-j_M, 0, j_M\}$. The bang-bang solution for the time optimal transition of systems with input and state constraints is a very well established result, see [63]. As can be inferred from Figure 3.10, the trajectory consists of three main phases, an acceleration Phase P1 to P3, a phase of maximum velocity P4, and a deceleration Phase P5 to P7. The acceleration phase itself can be divided into a

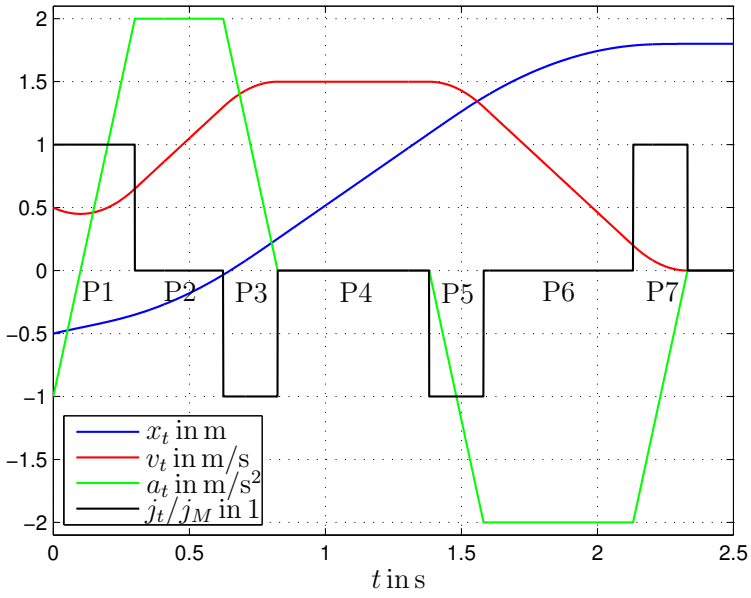

Figure 3.10: Seven phases of a time optimal $\mathcal{C}^2$-trajectory, maximum jerk $j_M = 10\text{m/s}^3$, maximum acceleration $a_M = 2\text{m/s}^2$, maximum velocity $v_M = 1.5\text{m/s}$.

Phase P1 where the acceleration increases, a Phase P2 of maximum acceleration and a Phase P3 where it decreases. The same holds true for the deceleration phase.

The jerk signal

$$j_t(t) = \begin{cases} j_1 & 0 \le t \le T_1 \\ j_3 & \bar{T}_2 \le t \le \bar{T}_3 \\ j_5 & \bar{T}_4 \le t \le \bar{T}_5 \\ j_7 & \bar{T}_6 \le t \le \bar{T}_7 = \bar{T}_x \\ 0 & \text{otherwise,} \end{cases} \tag{3.22}$$

where

$$\bar{T}_i = \sum_{j=1}^{i} T_j,\ i \in \{2, 3, \ldots, 7\}$$

is determined by the durations $T_1, \ldots, T_7$ and the jerk values $j_1, \ldots, j_7$ of the seven phases. Note that the jerk values in the phases P2, P4 and P6 are always $j_2 = j_4 = j_6 = 0$, as either the acceleration or the velocity are saturated. The overall trajectory time is denoted by $\bar{T}_x$. The notation shall be simplified by

introducing the following abbreviations

$$\mathbf{j}_i = \begin{bmatrix} j_1 & j_3 & \dots & j_i \end{bmatrix}, i \in \{3, 5, 7\} \tag{3.23a}$$

$$\mathbf{T}_j = \begin{bmatrix} T_1 & T_2 & \dots & T_j \end{bmatrix}, j \in \{2, \dots, 7\}, \tag{3.23b}$$

which are also applied to $(\cdot)^*$, $\tilde{(\cdot)}$ and so forth.

Integration of the jerk signal (3.22) yields the time evolution of acceleration, velocity and position

$$a_t(t; \mathbf{j}_7, \mathbf{T}_7, a_0) = \int_{t_0}^{t} j_t(\tau)\,\mathrm{d}\tau + a_0, \tag{3.24a}$$

$$v_t(t; \mathbf{j}_7, \mathbf{T}_7, v_0, a_0) = \int_{t_0}^{t} a_t(\tau)\,\mathrm{d}\tau + v_0, \tag{3.24b}$$

$$x_t(t; \mathbf{j}_7, \mathbf{T}_7, x_0, v_0, a_0) = \int_{t_0}^{t} v_t(\tau)\,\mathrm{d}\tau + x_0. \tag{3.24c}$$

In the following, it is shown how to compute a $\mathcal{C}^0$-trajectory for the acceleration, a $\mathcal{C}^1$-trajectory for the velocity and, finally, a $\mathcal{C}^2$-trajectory for the position.

A *$\mathcal{C}^0$-trajectory for the acceleration*, described by the parameters[11] $\pi_a = \begin{bmatrix} j_1^* & T_1^* \end{bmatrix} = A_t(a_0, a_d)$, from an initial value a_0 to a desired value a_d can be computed by

$$j_1^* = \begin{cases} -j_M & a_0 > a_d \\ j_M & a_0 < a_d \\ 0 & a_0 = a_d \end{cases} \tag{3.25}$$

and

$$T_1^* = \begin{cases} 0 & j_1^* = 0 \\ (a_d - a_0)/j_1^* & j_1^* \neq 0. \end{cases} \tag{3.26}$$

A *$\mathcal{C}^1$-trajectory for the velocity* consists of three phases and, consequently, is described by the parameter vector $\pi_v = \begin{bmatrix} \mathbf{j}_3^* & \mathbf{T}_3^* \end{bmatrix} = V_t(v_0, a_0, v_d)$ with the initial velocity v_0 and the desired value v_d. First, the jerk sequence $\begin{bmatrix} j_1^* & 0 & j_3^* \end{bmatrix}$ needs to be determined. This is accomplished by computing a $\mathcal{C}^0$-trajectory for the acceleration $\tilde{\pi}_a = \begin{bmatrix} \tilde{j}_1 & \tilde{T}_1 \end{bmatrix} = A_t(a_0, 0)$ and the constant velocity at the end of it

$$\tilde{v}_1 = v_0 + a_0 \tilde{T}_1 + \tilde{j}_1 \tilde{T}_1^2/2. \tag{3.27}$$

[11] A remark regarding the nomenclature: The function A_t takes the input arguments a_0, a_d and computes the parameter vector π_a describing a $\mathcal{C}^0$-trajectory for the acceleration. The symbol $*$ refers to the corresponding optimal value.

Based on $\tilde{v}_1$, the jerk sequence is

$$\begin{bmatrix} j_1^* & 0 & j_3^* \end{bmatrix} = \begin{cases} \begin{bmatrix} -j_M & 0 & j_M \end{bmatrix} & \tilde{v}_1 > v_d \\ \begin{bmatrix} j_M & 0 & -j_M \end{bmatrix} & \tilde{v}_1 < v_d \\ \begin{bmatrix} \tilde{j}_1 & 0 & 0 \end{bmatrix} & \tilde{v}_1 = v_d. \end{cases} \tag{3.28}$$

If $\tilde{v}_1 = v_d$, the desired velocity has been reached and, consequently, $\pi_v = \begin{bmatrix} \tilde{j}_1 & 0 & \tilde{T}_1 & 0 & 0 \end{bmatrix}$ describes the desired trajectory.

Otherwise, the second step is to determine the maximum achieved acceleration a^* in Phase P2 of the velocity trajectory. To do that, ignore the acceleration limit a_M, which implicitly reduces P2 to zero, and let $\hat{a}$ denote the unlimited maximum achieved acceleration. Consequently, the time intervals follow from

$$T_1 = (\hat{a} - a_0)/j_1^*, \tag{3.29a}$$
$$T_2 = 0, \tag{3.29b}$$
$$T_3 = (0 - \hat{a})/j_3^*. \tag{3.29c}$$

Furthermore, the following equations, which are simply derived by integration of the jerk signal (3.28), are needed: acceleration a_1 and velocity v_1 after P1, a_2 and v_2 after P2 and v_3 after P3

$$a_1 = a_0 + j_1^* T_1, \tag{3.30a}$$
$$v_1 = v_0 + a_0 T_1 + j_1^* T_1^2/2, \tag{3.30b}$$
$$a_2 = a_1 = \hat{a}, \tag{3.30c}$$
$$v_2 = v_1 + a_1 T_2, \tag{3.30d}$$
$$v_3 = v_2 + a_2 T_3 + j_3^* T_3^2/2 = v_d. \tag{3.30e}$$

Solving (3.30) and (3.29) for $\hat{a}$ yields, see also [62],

$$\hat{a} = \pm \frac{1}{j_1^* - j_3^*} \sqrt{(j_1^* - j_3^*)\, j_3^* \,(2v_0 j_1^* - 2v_d j_1^* - a_0^2)}. \tag{3.31}$$

The sign of $\hat{a}$ is chosen so that $T_1 > 0$ and $T_3 > 0$. Now, a^* can be determined considering the bounds on the acceleration

$$a^* = \begin{cases} -a_M & \hat{a} < -a_M \\ a_M & \hat{a} > a_M \\ \hat{a} & \text{otherwise.} \end{cases} \tag{3.32}$$

Third, it remains to compute the time parametrization $\begin{bmatrix} T_1^* & T_2^* & T_3^* \end{bmatrix}$. Substituting a^* for $\hat{a}$ in (3.29) yields T_1^* and T_3^*; T_2^* follows from

$$T_2^* = (v_d - v_3^*)/a^*, \tag{3.33}$$

where v_3^* is given by substituting $T_1 = T_1^*$, $T_2 = 0$ and $T_3 = T_3^*$ in (3.30). Note that (3.33) implicitly covers both possible cases. Assume in (3.32) $a^* = \hat{a}$, then $v_d = v_3^*$ and, consequently, the phase of constant maximum acceleration P2 is not needed, $T_2^* = 0$. Otherwise, a_M is reached and P2 does not vanish.

The computation of a $\mathcal{C}^2$*-trajectory for the position* is performed in a similar way. It consists of seven phases, i.e. $\pi_x = \begin{bmatrix} \mathbf{j}_7^* & \mathbf{T}_7^* \end{bmatrix} = X_t(x_0, v_0, a_0, x_d)$ with the initial position x_0 and the desired value x_d. However, based on the $\mathcal{C}^1$-trajectory for the velocity, only its three main phases, acceleration P1 to P3, maximum velocity P4 and deceleration P5 to P7 need to be considered.

The major task is the computation of the acceleration and deceleration phase. To this end, the maximum achieved velocity $v^* = v_3 = v_4$ during P4 needs to be determined using binary search. Initially $v^- = -v_M$ and $v^+ = v_M$ is chosen. In every iteration, the following steps are executed, see [62],

1. Determine new $v^* = (v^- + v^+)/2$.

2. Compute the acceleration phase, i.e. a $\mathcal{C}^1$-trajectory for the velocity $\tilde{\pi}_{v,13} = \begin{bmatrix} \tilde{j}_1 & \tilde{j}_3 & \tilde{T}_1 & \tilde{T}_2 & \tilde{T}_3 \end{bmatrix} = V_t(v_0, a_0, v^*)$.

3. Compute the deceleration phase, i.e. a $\mathcal{C}^1$-trajectory for the velocity $\tilde{\pi}_{v,57} = \begin{bmatrix} \tilde{j}_5 & \tilde{j}_7 & \tilde{T}_5 & \tilde{T}_6 & \tilde{T}_7 \end{bmatrix} = V_t(v^*, 0, 0)$.

4. Compute the total distance covered during the acceleration and deceleration phase, $\tilde{x}_7 = x_t(t; \tilde{\mathbf{j}}_7, \tilde{\mathbf{T}}_7, x_0, v_0, a_0)$ with $\tilde{T}_4 = 0$.

5. Shift the search interval according to

$$\begin{bmatrix} v^- & v^+ \end{bmatrix} = \begin{cases} \begin{bmatrix} v^- & v^* \end{bmatrix} & \tilde{x}_7 > x_d \\ \begin{bmatrix} v^* & v^+ \end{bmatrix} & \tilde{x}_7 < x_d \\ \begin{bmatrix} v^* & v^* \end{bmatrix} & \tilde{x}_7 = x_d. \end{cases} \tag{3.34}$$

6. If $\tilde{x}_7 = x_d$ or the maximum number of iterations [12] is reached, the search stops, otherwise go to step 1).

At the end of the binary search, the maximum achieved velocity v^* is found and the respective acceleration and deceleration phase are already computed, i.e. the parameters are considered final, $(\cdot)^* = \tilde{(\cdot)}$. It remains to compute the length of the phase of maximum velocity P4

$$T_4^* = (x_d - \tilde{x}_7)/v^*. \tag{3.35}$$

[12] Regarding the convergence of the binary search, note that a small number of eleven iterations, already determines v^* with an accuracy of $2v_M/2^{11} = v_M/1024$.

This way, given an arbitrary initial state $\begin{bmatrix} x_0 & v_0 & a_0 \end{bmatrix}$ and the desired position x_d, the optimal $\mathcal{C}^2$-trajectory can be parametrized by $\begin{bmatrix} \mathbf{j}_7^* & \mathbf{T}_7^* \end{bmatrix} = X_t(x_0, v_0, a_0, x_d)$ and the trajectory signals follow from (3.24).

The requirement to start at an arbitrary initial state makes the task of trajectory generation much more complex - a level of complexity that is not required, for the computation of the cost matrix (3.5) of the path planning problem, since only the point-to-point transition times are required. Therefore, a point-to-point version of the Bang-Bang Trajectory Generator is presented in the following.

3.3.2.1 Point-to-point version

In the following, $x_d > x_0$ is assumed. If this is not the case, the start point is exchanged for the end point, $x_0 \leftrightarrow x_d$. This is possible because the system dynamics are equal in forward and backward direction and thus the problem is symmetric. Furthermore, $x_0 = 0$ and $x_d \leftarrow x_d - x_0$ is set without loss of generality.

Because of the symmetry of the problem, the trajectory parametrization $\pi_x = \begin{bmatrix} \mathbf{j}_7^* & \mathbf{T}_7^* \end{bmatrix}$ becomes much simpler. The phases of constant jerk, acceleration and velocity are of equal respective length,

$$T_j/2 = T_1 = T_3 = T_5 = T_7, \tag{3.36a}$$
$$T_a = T_2 = T_6, \tag{3.36b}$$
$$T_v = T_4. \tag{3.36c}$$

Also, since $x_d > x_0$, the jerk sequence is always

$$j_M = j_1 = j_7 \tag{3.37a}$$
$$-j_M = j_3 = j_5. \tag{3.37b}$$

The overall trajectory time calculates as

$$\bar{T}_x = 2T_j + 2T_a + T_v. \tag{3.38}$$

It remains to compute the reduced parameter vector $\pi_{x,p2p} = \begin{bmatrix} T_j & T_a & T_v \end{bmatrix}$. The actuator needs the time $T_j/2$ to build up its maximum acceleration. This time span is consequently determined by the maximum jerk

$$T_j = 2a_M/j_M. \tag{3.39}$$

Moreover, T_a computes from the time it takes to accelerate up to maximum velocity. Maximum velocity v_M is reached at the end of Phase P3, i.e. at $t = 2T_j/2 + T_a$. Thus, solving $v_t(t)|_{t=T_j+T_a} = v_M$ yields

$$T_a = v_M/a_M - T_j/2. \tag{3.40}$$

Similarly, T_v is determined by the distance x_d to be covered. Solving $x_t(t)|_{t=\bar{T}_x} = x_d$ with $\bar{T}_x$ according to (3.38) yields

$$T_v = \frac{2x_d/a_M - T_j^2 - 3T_jT_a - 2T_a^2}{T_j + 2T_a}. \tag{3.41}$$

Looking at (3.40) and (3.41), in general, it might happen that $T_a < 0$ or $T_v < 0$ or both. These three cases have to be distinguished as in Algorithm 5.

Algorithm 5 Point-to-point trajectory

$T_j = 2a_M/j_M$ {always non-negative}
$T_a = v_M/a_M - T_j/2$

if $T_a < 0$ **then**
  $T_a = 0$ {Case 1}
  $a_M = \sqrt{j_M v_M}$
  $T_j = 2a_M/j_M$
end if

$T_v = \frac{2x_d/a_M - T_j^2 - 3T_jT_a - 2T_a^2}{T_j + 2T_a}$

if $T_v < 0$ **then**
  $T_v = 0$ {Case 2}
  $v_M = \frac{-a_M^2 + \sqrt{a_M^4 + 4j_M^2 a_M x_d}}{2j_M}$
  $T_a = v_M/a_M - T_j/2$

  if $T_a < 0$ **then**
    $T_v = 0$ {Case 3}
    $T_a = 0$
    $a_M = (x_d/2)^{1/3} j_M^{2/3}$
    $v_M = (x_d/2)^{2/3} j_M^{1/3}$
    $T_j = 2a_M/j_M$
  end if
end if

$\pi_{x,p2p} = \begin{bmatrix} T_j & T_a & T_v \end{bmatrix}$ {reduced parameter vector}

In Case 1, $T_a < 0$ indicates that maximum velocity v_M is already reached before maximum acceleration. Therefore, a_M needs to be adjusted. Inserting (3.39) into (3.40), setting $T_a = 0$ and solving for a_M yields

$$a_{M0} = \sqrt{j_M v_M}. \tag{3.42}$$

Then, of course, (3.39) needs to be evaluated with the newly obtained maximum acceleration a_{M0}.

In Case 2, $T_v < 0$ indicates that the desired distance x_d is reached before maximum velocity. Therefore, v_M needs to be adjusted. Setting $T_v = 0$ and solving (3.39), (3.40) and (3.41) for v_M yields

$$v_{M0} = \frac{-a_M^2 + \sqrt{a_M^4 + 4j_M^2 a_M x_d}}{2j_M}. \tag{3.43}$$

Then, (3.40) needs to be evaluated with the newly obtained maximum velocity v_{M0}.

In Case 3, $T_a < 0$ and $T_v < 0$, both are set to zero, $T_a = T_v = 0$, and solving (3.39), (3.40) and (3.41) yields

$$a_{M00} = (x_d/2)^{1/3} j_M^{2/3}, \tag{3.44a}$$
$$v_{M00} = (x_d/2)^{2/3} j_M^{1/3}. \tag{3.44b}$$

Then, (3.39) needs to be evaluated with the newly obtained maximum acceleration a_{M00}.

These extremely simple computations yield the time optimal point-to-point trajectory.

3.3.3 Results and analysis

For the sake of comparison, a time optimal, time-discrete online trajectory generator is implemented, see Section A.2. Obviously, it also employs the bang-bang principle and thus yields the identical two times continuously differentiable trajectories as the offline approach of Section 3.3.2. The offline version computes the seven individual time segments of maximum, zero and minimum jerk all at once. In contrast, the online approach computes each time step separately.

Figure 3.11 shows trajectories of both generators that approach multiple desired end positions. They start at the origin of the state space $\begin{bmatrix} x_t(0) & v_t(0) & a_t(0) \end{bmatrix} = \mathbf{0}$. At $t_0 = 0$ the reference value jumps to 5m. Shortly before this value is reached, i.e. at $t_1 = 2.3$s, the reference value increases to 6.5m. Consequently, the deceleration phase, $a_t(t_1) = -a_M$, is canceled and a short phase of maximum acceleration is introduced. Again, during the final approach to the reference value, the desired position is dropped to 1.5m. This time, no immediate reaction of the trajectory generators can be seen. This is because the acceleration is already at its minimum $-a_M$ and it takes some time until the velocity signal changes its sign.

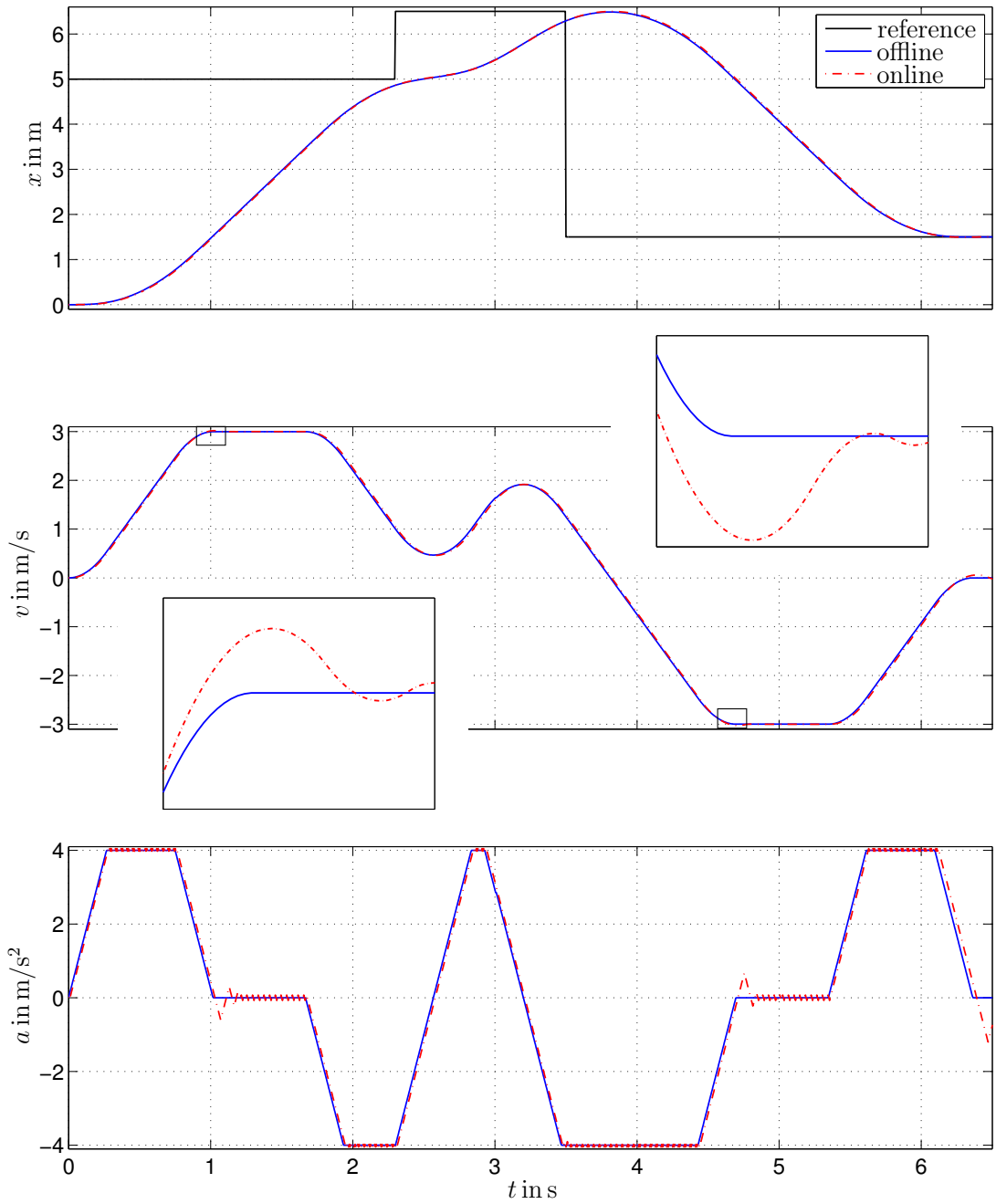

Figure 3.11: Comparison of the offline versus the online Bang-Bang Trajectory Generator with maximum jerk $j_M = 15\text{m/s}^3$, maximum acceleration $a_M = 4\text{m/s}^2$, maximum velocity $v_M = 3\text{m/s}$ and a sampling time of 5ms.

The effects of the above mentioned conceptual difference between the two approaches are small, but noteworthy. In the (quasi-) time-continuous case, sampling time $T_s = 1\text{ms}$ and below, both generate strictly time optimal motion profiles. However, in the depicted case of 5ms, the acceleration profile of the online version exhibits chatter, which causes overshoots in the velocity signal. This is because the online Bang-Bang Trajectory Generator is only able to switch the jerk at the sampling instants. Naturally, for higher sampling times, this results in oscillating behavior, which clearly counteracts precise and fast positioning. The more recent papers [30, 34] present solutions to this problem. The offline approach avoids this problem *in the first place* by planning a continuous trajectory, i.e. computing the parameters for a continuous time trajectory. Given these parameters, the time-continuous motion profiles can be evaluated (correctly) at any point in time. Still, the offline Bang-Bang Trajectory Generator can be executed within a fraction of a Millisecond, which clearly makes it real-time capable. Therefore, the presented offline version of the Bang-Bang Trajectory Generator is the tool of choice and shall be used for the real-time control of the patching robot, which is the topic of the next chapter.

Finally, it shall be noted that both trajectory generators can be applied to any 1D-motion-planning-task, translational or rotational. This naturally includes multidimensional motion that can be decomposed accordingly.

CHAPTER 4

Real-time control

In Chapter 3, the optimal processing sequence for each panel is determined. To execute this sequence, a real-time controller for the patching robot has to be developed, see [47]. The patching action itself is accomplished by a patching tool already in operation at the plant for many years. Positioning of the panel underneath the patching tool is performed by two xy-tables, see Section 2.2. As already discussed, friction forces within the drivetrain of the rubber belts of the xy-tables are high. Furthermore, in certain process stages the panel is moved by both xy-tables cooperatively. In this case, the xy-tables have to be synchronized to ensure good positioning performance. Thus, a suitable control structure needs to handle the challenging friction conditions and support both, independent and coupled positioning, see Section 4.1.

The position control is accomplished under the assumption that the absolute position of the panel is continuously available, i.e. at the sampling rate of the position controller. However, in longitudinal direction, there occurs slip between the panel and the rubber belts of the patching robot. Consequently, the incremental encoders of the motors of the rubber belts of the patching robot do not yield the exact longitudinal panel position, which is why the absolute panel position needs to be tracked separately, e.g., by a camera system. The position signal of the camera is not available at the high sampling rate of the encoder signals and thus a strategy for sensor data fusion called Trajectory Updating is introduced in Section 4.2.

4.1 Control structure

In this section, the control structure for independent and coupled positioning is discussed. The longitudinal and lateral degree of freedom of the xy-tables are decoupled. In the following, it is assumed that slip between the rubber belts of

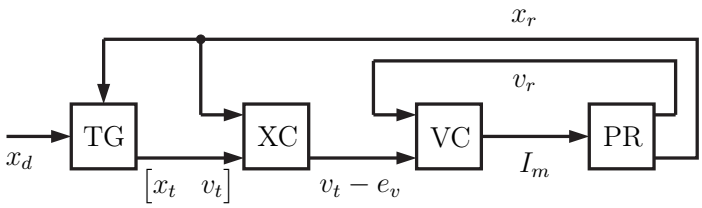

Figure 4.1: Structure of the longitudinal position controller. Since the control structure is identical for both xy-tables, the index $i \in \{1, 2\}$ is suppressed.

the xy-tables and the panel is negligible. Thus, the longitudinal panel position x_p and the robot position x_r coincide, i.e. $x_p = x_r = r_r\,\varphi_r/i_x$ with r_r being the effective radius of the sprockets driving the rubber belts, φ_r the angular position of the sprockets and i_x being the transmission ratio of the gearbox.

To begin with, independent positioning, i.e. when only one xy-table moves the panel, is discussed. Exemplarily, the longitudinal direction is considered; in lateral direction, it is exactly the same. The corresponding control structure is depicted in Figure 4.1. The inner velocity controller (VC) consists of a PI- and a feedforward controller with the translational velocity $v_r = r_r\,\dot{\varphi}_r/i_x$ of the rubber belts as control variable, the desired velocity $v_t - e_v$ as reference input and the motor current I_m of the synchronous servo motor driving the rubber belts of the patching robot (PR) as control input. This controller is directly implemented at the frequency converters. The feedforward control input is derived by numerical differentiation of the smooth signal of the desired velocity $v_t - e_v$. The outer position control loop, i.e. the position controller (XC) and the trajectory generator (TG), is implemented at an industrial computer. The position controller, a high-gain P-controller, ensures that the position x_r of the rubber belts follows the trajectory $\begin{bmatrix} x_t & v_t \end{bmatrix}$ leading to a desired end position x_d. The reasoning behind the partitioning of the controllers between frequency converter and industrial computer is the following: On the one hand, the inner velocity control loop and its subordinate current control loop[1] are implemented at a low level in order to facilitate a high sampling rate. This is necessary to be able to deal with the high friction forces in longitudinal direction. On the other hand, the outer position control loop requires more computational power to meet the demands of the Trajectory Updating Strategy proposed in Section 4.2.2.

This partitioning also allows to reset the longitudinal position of the rubber belts without interfering with the velocity control. The reset of the rubber belt position is accomplished by resetting an offset value at the industrial computer. The motor encoder itself retains its value. This feature is required for the Trajectory Updating Algorithm. Furthermore, the entire plant control is implemented at this computer.

If both xy-tables move the panel cooperatively, x- and y-direction need to

[1]The current controller cannot be accessed and thus is not shown in the flow chart.

be treated separately. In longitudinal direction, the coupling is firm, meaning $F_{px} \approx \mu F_c$, where F_{px} is the longitudinal force exerted on the panel by the rubber belts, F_c is the clamping force and μ is the friction coefficient of the rubber belts. Therefore, a master-slave control is utilized. XY-Table 1, the master, controls the panel position as presented above, while XY-Table 2, the slave, merely follows the speed of the master. This means that, while the position controller of XY-Table 1 controls the panel position, the velocity controller of XY-Table 2 is fed the same desired velocity as XY-Table 1, $v_{t,2} - e_{v,2} = v_{t,1} - e_{v,1}$. Thus, XY-Table 2 interferes with the positioning action as little as possible, while helping accelerate the panel.

In lateral direction, both xy-tables are positioned independently. On the one hand, this is necessary because the coupling of the xy-tables via the panel is rather loose, meaning $F_{py} \gg F_c$, where F_{py} is the lateral force exerted on the panel. On the other hand, this is possible because the friction forces in lateral direction are very low and, therefore, trajectory tracking is very accurate. Consequently, when fed the same desired trajectory, both xy-tables perform the almost identical motion independently of each other.

Finally, it shall be pointed out that the real-time controllers were heuristically tuned directly at the plant. The stick-slip effects within the drive train of the patching robot as well as between the rubber belts and the panels are unpredictable. Thus, it is difficult to derive an accurate mathematical model, which is why heuristic tuning is deemed the most viable approach. The presented control structure is implemented at the pilot patching plant. All measurement results obtained from the plant are documented in Chapter 6.

4.2 Sensor data fusion

In lateral direction, positioning of the panel below the patching tool is a straight forward task using the synchronous linear motor to move the entire slide of the xy-table. In longitudinal direction, the rubber belt conveyors exert acceleration forces on the panel only via friction. The transmittable forces before slip occurs depend on many factors, like clamping pressure, wear of the rubber belts, roughness and curvature of the (ideally flat) wood panels. Also, the overlap between the rubber belts and the panel depends on panel length and position. This slip between the panel and the rubber belt induces odometry errors. Consequently, the position of the panel cannot be determined accurately using the motor encoders. Separate absolute position tracking, e.g., by a camera is required.

4.2.1 Literature review

In literature, systematic and non-systematic odometry errors are distinguished, see [13]. Examples for systematic errors are unknown or unequal effective radii of

the wheels driving a robot or their misalignment. Non-systematic errors are, e.g., slip caused by too high acceleration and changing properties of stick-slip friction or, in general, unpredictable features of the environment, such as sandy terrain.

In order to deal with these errors, sensor data fusion is required. According to [78], the fusion strategies can be classified into three main approaches: probabilistic models, least-squares techniques and intelligent fusion. The probabilistic model methods are Bayesian reasoning, evidence theory, robust statistics and recursive operators. The least-squares techniques are Kalman Filtering, optimal theory, regularization and uncertainty ellipsoids. The intelligent fusion methods are fuzzy logic, neural networks and genetic algorithms.

The most common strategy in engineering is Kalman Filtering, see [10, 13, 17, 42, 50, 85]. To this end, a dynamical odometry and measurement model of the machine have to be derived. If systematic odometry errors are known, they can be considered in the model of the machine. In [13, 85], unknown effective wheel base, unequal wheel diameters and side slip due to wheel deflection are taken into account. Eliminating these dominant sources of systematic errors greatly increases the accuracy of the odometry system. In the Kalman Filter, odometry is used for the prediction step. Whenever the absolute position of the robot can be determined, e.g., by visual position tracking, magnetic markers on the floor, etc., the correction step of the Kalman Filter is executed.

In return, odometry information can be used to shrink the search region for visual position tracking algorithms, which shortens their computation time. Using odometry to support visual position tracking is a concept already proposed in [12]. It is called Verification Vision and characterized by two features. First, the vision system has a great deal of prior knowledge about the scene and, second, the vision system is only used to verify and refine the location of objects in the scene. This synergy between odometry and absolute position measurement using map-matching is also applied in [20]. In the following, a different approach to sensor data fusion in odometry is presented.

4.2.2 Trajectory Updating

In longitudinal direction, high friction forces in the drivetrain of the rubber belts and slip between the rubber belts and the shuttering panel turn out to be the main challenges. The control structure presented in Section 4.1 is suited to deal with the first one of these two challenges.

In Figure 4.2, this structure is extended to account for the second challenge, slippage of the panel. The proposed strategy for position control in the presence of slip utilizes two different sources of position information. First, the incremental motor encoders are used in the velocity and position control loop, see Figure 4.1. They support a high sampling rate $1/T_s$ to achieve high positioning dynamics and accuracy, despite high friction forces. Second, the panel position provided by the position tracking camera (CM) is used to detect the slip between robot x_r

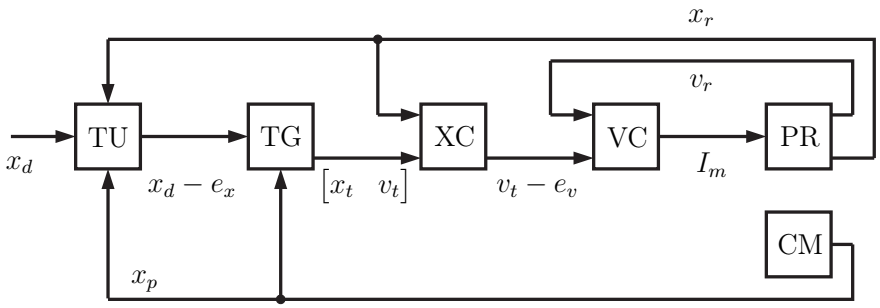

Figure 4.2: Extension of the control structure in longitudinal direction, compare Figure 4.1.

and panel position x_p. However, due to the high computational costs of computer vision, the panel position is only available at a low sampling rate $1/T_{sv}$ and with a time delay T_{dv}.

In order to compensate for the slip that accumulates over time, in the Trajectory Update Block (TU), the position deviation $e_x(t-T_{dv}) = x_r(t-T_{dv}) - x_p(t-T_{dv})$ is determined at a sampling rate of $1/T_{sv}$. After the time delay T_{dv}, during which the change in the position deviation is assumed to be negligible, i.e. $e_x(t) \approx e_x(t-T_{dv})$, a new trajectory leading to a corrected desired end position $x_d(t) - e_x(t)$ is computed in real-time. Starting at $x_0 = x_t(t)$, $v_0 = v_t(t)$, $a_0 = a_t(t)$ ensures the smoothness of the piecewise generated overall trajectory.

This procedure is summarized in Algorithm 6. After processing of a patch

Algorithm 6 Trajectory Updating

1: $t = k\,T_s$ {current time step}
2: **if** $x_d(t) \neq x_d((k-1)\,T_s)$ **then**
3: $\quad x_r(t) = x_p(t)$ {reset robot position}
4: $\quad \begin{bmatrix} \mathbf{j}_7^* & \mathbf{T}_7^* \end{bmatrix} = X_t(x_r(t), 0, 0, x_d(t))$ {p2p-trajectory}
5: **end if**
6: **if** $x_p(t-T_{dv}) \neq x_p(t-T_{sv}-T_{dv})$ **then**
7: $\quad e_x(t) = x_r(t-T_{dv}) - x_p(t-T_{dv})$
8: $\quad x_0 = x_t(t),\ v_0 = v_t(t),\ a_0 = a_t(t)$
9: $\quad \begin{bmatrix} \mathbf{j}_7^* & \mathbf{T}_7^* \end{bmatrix} = X_t(x_0, v_0, a_0, x_d(t) - e_x(t))$ {update}
10: **end if**
11: $a_t(t; \mathbf{j}_7^*, \mathbf{T}_7^*, a_0)$
12: $v_t(t; \mathbf{j}_7^*, \mathbf{T}_7^*, v_0, a_0)$
13: $x_t(t; \mathbf{j}_7^*, \mathbf{T}_7^*, x_0, v_0, a_0)$

position x_{i-1}, the next one x_i needs to be approached according to the optimal processing sequence of the patch list $\mathbf{\Pi}^* = \{\begin{bmatrix} x_1 & y_1 \end{bmatrix}^{\mathrm{T}}, \ldots, \begin{bmatrix} x_n & y_n \end{bmatrix}^{\mathrm{T}}\}$. This change of the set point $x_d(t) = x_i \neq x_d((k-1)\,T_s) = x_{i-1}$ triggers the computation of a new parametrization of a point-to-point trajectory, see line 4. The trajectory

generator is described in Section 3.3.2. Note that during the patching action, the panel does not move for $T_p = 2$s. Also, the controllers of the patching robot are switched off for safety reasons. Therefore, it is possible to reset the robot position to the panel position. Whenever there is a new absolute position for the panel available, a trajectory update is performed, see line 9. The current values of the trajectory for each time step are computed in lines 11–13, see (3.24).

The Trajectory Updating Strategy is implemented at the pilot patching plant. All measurement results obtained from the plant are documented in Chapter 6.

CHAPTER 5

Process logic control

In Chapter 4, real-time controllers for the patching robot are developed. They move the xy-tables according to the patch list to position the panel underneath the patching tool. In this chapter, the task is to develop a suitable process logic control for the patching robot. An overview of the patching robot and the prototype plant is given in Sections 2.2 and 2.3. Based on this, a state-machine for the patching robot is presented in Section 5.1. In Section 5.2, Simulation studies demonstrate the feasibility of this approach.

5.1 State-machine for the patching robot

A finite-state-machine or finite-state-automaton, see [49], is an abstract computing device, modeling, e.g., computer programs, sequential logic circuits or the behavior of real devices and machines. The behavior of these abstract machines is determined by a finite set of states. The machine can only be in one state at a time, called the current state. Transitions from the current state into another are triggered by meeting some specific transition criteria.

Each state, as implemented in this process logic control, is defined by entry actions, state actions and transition conditions. The entry actions are only executed once, when entering the state. By contrast, the state actions are executed as long as the machine remains in this state. One or more transition conditions define criteria, events or inputs, that trigger a change to different subsequent states. By this, the current state fully determines the behavior of the machine at the current time. Executing the state-machine in a cyclic thread, a flow chart of a single generic state[1] looks like Figure 5.1. In each sampling interval, one of the

[1]Note that a state can be structured in various ways, e.g., the entry actions of one state could also be modeled as exit actions of the previous state. Or, the entry and state actions of this generic state could be implemented as individual states themselves.

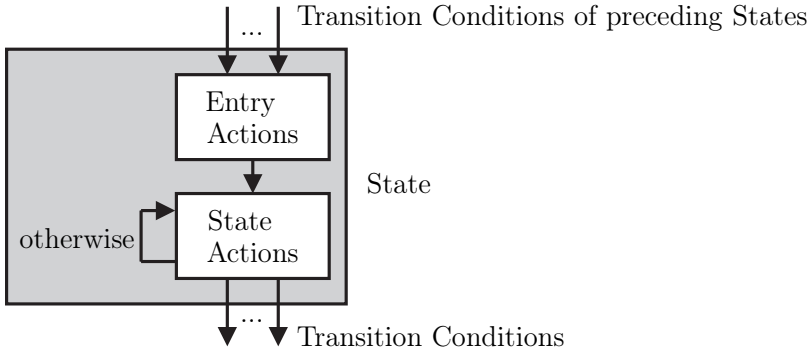

Figure 5.1: Flow chart of a generic state.

transitions, indicated by the arrows, is executed. Each arrow is labeled with its transition condition. The transition from the entry actions into the state actions is unconditional and thus got no label.

The purpose of a state-machine is to provide a certain robustness of the control against unexpected deviations from the nominal process, e.g., faulty sensor signals, time delays etc.. Unexpected occurrences, that might affect the machine's behavior in different states, are ignored, since they are irrelevant to the current state.

In the following, the patching process roughly described in Section 2.2.2 is implemented using state-machines. To this end, four specific positions of the xy-tables, respectively the panel, have to be introduced. If XY-Table 1 is in *Feed-In Position* $\mathbf{x}_{fi} = \begin{bmatrix} 0 & y_{cnv1} \end{bmatrix}^{\mathrm{T}}$, it is laterally aligned with Conveyor 1 at $y_{cnv1} = 0.250\text{m}$ in front of the patching robot. Thus, panels can be fed from the conveyor to the xy-table. The longitudinal position is irrelevant. Similarly, a *Move-Out Position* for XY-Table 2 is defined. In Move-Out Position $\mathbf{x}_{mo} = \begin{bmatrix} L + x_{lb21} & y_{cnv2} \end{bmatrix}^{\mathrm{T}}$, XY-Table 2 is laterally aligned with Conveyor 2 at $y_{cnv2} = 0.250\text{m}$. The longitudinal position $L + x_{lb21}$ of the panel is chosen such that XY-Table 2 can be lowered. If Light Barrier 21 at $x_{lb21} = 0.175\text{m}$ is not interrupted, the panel does not occupy the workspace of the patching tool anymore and XY-Table 2 can be lowered. Furthermore, two handover positions are added to the patch list of each panel. *Handover Position 1* $\mathbf{x}_{ho1} = \begin{bmatrix} x_{ho} & y_{i-1} \end{bmatrix}^{\mathrm{T}}$ is required to move the panel to the workspace of XY-Table 2 and thus prepares the transition from XY-Table 1 moving the panel independently into coupled positioning. The lateral coordinate of $\mathbf{x}_{ho1}$ is irrelevant, so it is just copied from the previous entry of the patch list y_{i-1}, where i is the index of the handover position in the patch list. The longitudinal coordinate x_{ho} is chosen, such that there is sufficient overlap between XY-Table 2 and the panel. In case of Figure 3.9, $x_{ho} = 0.375\text{m}$, i.e. $\mathbf{x}_{ho1} = \begin{bmatrix} 0.375 & 0.141 \end{bmatrix}^{\mathrm{T}}$. In analogy, *Handover Position 2* $\mathbf{x}_{ho2} = \begin{bmatrix} L - x_{ho} & y_{i-1} \end{bmatrix}^{\mathrm{T}}$ is defined for the transition from coupled positioning into XY-Table 2 positioning independently. In case of Figure 3.9,

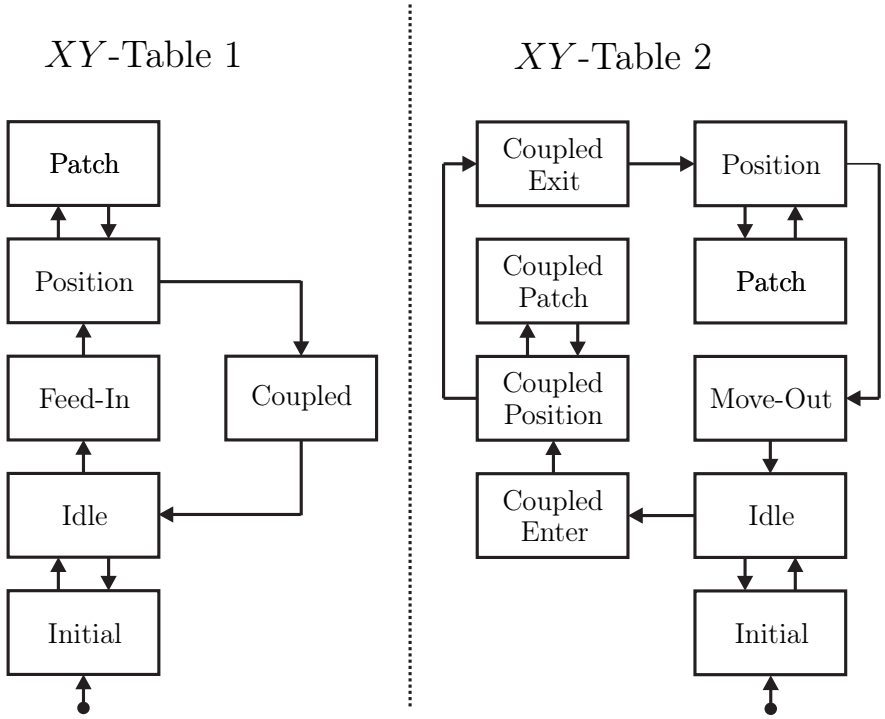

Figure 5.2: State-machine of the patching robot. To preserve clarity, the individual transition conditions are not explicitly described in this chart, only in the text.

$\mathbf{x}_{ho2} = \begin{bmatrix} 2.017 - 0.375 & 0.443 \end{bmatrix}^{\mathrm{T}}$.

Now, that these prerequisites are established, the process logic control is described in detail. It is divided into two state-machines, one for each xy-table. Figure 5.2 depicts the individual states and transitions of the state-machines of the xy-tables[2]. In the following, each state is described by listing its entry and state actions as well as transition conditions.

XY-Table 1, Initial State State at power-up or reset.

- Entry actions: None.
- State actions: Do not move.
- Transition conditions: If the process logic control is released, transition into Idle State.

XY-Table 1, Idle State XY-Table 1 moves to the Feed-In Position and waits for the next panel to approach.

[2]For brevity, the state-machine of XY-Table 1 is simply referred to as XY-Table 1 in the remainder of the document.

- Entry actions: Delete panel data.
- State actions:
  1. Release the clamping mechanism.
  2. If the clamping mechanism is released, lower XY-Table 1.
  3. If XY-Table 1 is lowered, move to the (lateral) Feed-In position and set longitudinal position to zero.
  4. If XY-Table 1 is positioned correctly, activate Conveyor 1.
- Transition conditions:
  1. If XY-Table 1 is positioned correctly and Light Barrier 11[3] is interrupted, transition into Feed-In State.
  2. If XY-Table 1 is positioned correctly and the process logic control is locked, transition into Initial State.

XY-Table 1, Feed-In State The panel approaching from Conveyor 1 is fed into XY-Table 1.

- Entry actions:
  1. Request the dataset of the respective panel from the wood knowledge repository and assign it to XY-Table 1.
  2. Engage the clamping mechanism.
- State actions:
  1. The rubber belts and Conveyor 1 move the panel forward. The rubber belts are velocity controlled, since their position, respectively the panel position, is not defined yet.
  2. If Light Barrier 13 is interrupted, the rubber belts and Conveyor 1 stop.
  3. If XY-Table 2 is in either Idle or Move-Out State, XY-Table 1 is elevated.
- Transition conditions: If XY-Table 1 is elevated, transition into Position State.

[3] A CAD drawing of the patching robot and, in particular, the numbering of the light barriers are given in Figure 2.2.

XY-Table 1, Position State XY-Table 1 positions the panel according to the position yielded by Camera 1.

- Entry actions:
  1. If the real-time controllers are switched off, reset them and switch them on.
  2. Reset the longitudinal position of XY-Table 1 to the position yielded by Camera 1.
  3. Determine the next position to approach from the panel data set.
- State actions:
  1. If this position is Handover Position 1 and XY-Table 2 is in Idle State, XY-Table 2 is sent Handover Position 1 and triggered to transition into Coupled Enter State.
  2. Position the panel.
- Transition conditions: If the panel is positioned correctly and it is not Handover Position 1, transition into Patch State.

XY-Table 1, Patch State XY-Table 1 is switched off and the patching action is executed.

- Entry actions:
  1. Switch off the real-time controllers.
  2. Activate the patching tool.
- State actions: If the patching action is finished, mark the respective patch position in the patch list of the panel data set.
- Transition conditions: If the patching action is finished, transition into Position State.

XY-Table 1, Coupled State This state represents the cooperative motion of the two xy-tables. XY-Table 1 receives instructions from XY-Table 2, Coupled Position State or XY-Table 2, Coupled Patch State, regarding both actions and transitions.

- Entry actions: None.
- State actions: None.
- Transition conditions: None.

XY-Table 2, Initial State Equivalent to XY-Table 1, Initial State.

XY-Table 2, Idle State XY-Table 2 waits until it is triggered to transition into Coupled Enter State.

- Entry actions: Delete panel data.
- State actions:
  1. Release the clamping mechanism.
  2. If the clamping mechanism is released, lower XY-Table 2.
  3. If XY-Table 2 is lowered, stay at the current position[4].
- Transition conditions: If XY-Table 2 is positioned correctly and the process logic control is locked, transition into Initial State.

XY-Table 2, Coupled Enter State XY-Table 2 moves to Handover Position 1 and clamps the panel.

- Entry actions: None.
- State actions:
  1. Move to Handover Position 1 as received from XY-Table 1, Position State.
  2. If both xy-tables are positioned correctly, XY-Table 2 is elevated.
  3. If XY-Table 2 is elevated and Light Barrier 21 is interrupted, its clamping mechanism is engaged.
  4. If XY-Table 2 clamped the panel, the panel data set of XY-Table 1 is also assigned to XY-Table 2, Handover Position 1 is marked and XY-Table 1 is triggered to change from Position State to Coupled State.
- Transition conditions: If XY-Table 2 clamped the panel, transition into Coupled Position State.

[4]The current position usually is the Move-Out Position. Handover Position 1 of the next panel is not known yet.

XY-Table 2, Coupled Position State Both xy-tables position the panel cooperatively according to the position yielded by Camera 1 (of XY-Table 1).

- Entry actions:
  1. If the real-time controllers of both xy-tables are switched off, reset them and switch them on.
  2. Reset the longitudinal position of both xy-tables to the position yielded by Camera 1.
  3. Determine the next position to approach from the panel data set.
- State actions:
  1. Position the panel.
  2. If the panel is positioned correctly and it is Handover Position 2, XY-Table 1 is triggered to transition into Idle State.
- Transition conditions:
  1. If the panel is positioned correctly and it is not Handover Position 2, transition into Coupled Patch State.
  2. If the panel is positioned correctly and it is Handover Position 2, transition into Coupled Exit State.

XY-Table 2, Coupled Patch State Both xy-tables are switched off and the patching action is executed.

- Entry actions:
  1. Switch off the real-time controllers of both xy-tables.
  2. Activate the patching tool.
- State actions: If the patching action is finished, mark the respective patch position in the patch list of the panel data set.
- Transition conditions: If the patching action is finished, transition into Coupled Position State.

XY-Table 2, Coupled Exit State Wait until XY-Table 1 has left the workspace of XY-Table 2.

- Entry actions: None.
- State actions:

1. Hold Handover Position 2, do not move.
2. If XY-Table 1 is lowered, Handover Position 2 is marked.

- Transition conditions: If XY-Table 1 is lowered, XY-Table 2 transitions into Position State.

XY-Table 2, Position State XY-Table 2 positions the panel according to the position yielded by Camera 2.

- Entry actions:
  1. If the real-time controllers are switched off, reset them and switch them on.
  2. Reset the longitudinal position of XY-Table 2 to the position yielded by Camera 2.
  3. Determine next position to approach from the panel data set.
- State actions: Position the panel.
- Transition conditions:
  1. If the panel is positioned correctly and it is not the Move-Out Position, transition into Patch State.
  2. If the panel is positioned correctly and it is the Move-Out Position and Light Barrier 21 is not interrupted, transition into Move-Out State.

XY-Table 2, Patch State Equivalent to XY-Table 1, Patch State.

XY-Table 2, Move-Out State The panel which has just been processed is moved out to Conveyor 2.

- Entry actions: Send the data set of the processed panel back to the wood knowledge repository.
- State actions:
  1. Lower XY-Table 2.
  2. If XY-Table 2 is lowered, the rubber belts and Conveyor 2 move the panel forward. The rubber belts are velocity controlled.
  3. If Light Barrier 21 and Light Barrier 22 and Light Barrier 23 are not interrupted, the rubber belts stop.
- Transition conditions: If Light Barrier 21 and Light Barrier 22 and Light Barrier 23 are not interrupted, transition into Idle State.

This section shall be concluded with two important remarks regarding the state-machines. First, the real-time controllers are switched off while in Patch State. This is possible because the xy-tables do not (and are not supposed to) move in this state. It is to make sure, they do not interfere with the patching process. When switching them back on again, they require a reset.

Second, a critical phase of the process logic control clearly is entering the Coupled State. It is executed as follows: XY-Table 1, Position State triggers the transition of XY-Table 2 from Idle into Coupled Enter State. Then, after XY-Table 2 engaged its clamping mechanism, XY-Table 2 transitions into Coupled Position state and at the same time triggers the transition of XY-Table 1 into Coupled State. This procedure is called handshake, as both xy-tables have to consent to this transition. Leaving Coupled State follows a similar pattern. In XY-Table 2, Coupled Position State, if the position to approach is Handover Position 2 and the panel is positioned accurately, XY-Table 1 is triggered to transition into Idle State. At the same time, XY-Table 2 transitions into Coupled Exit State and waits until XY-Table 1 is lowered. Only then, XY-Table 2 transitions into Position State.

The process logic control is implemented in C++ according to design patterns for state-machines presented in [28].

5.2 Simulation

In this section, the process logic control is applied to a simulation model of the prototype patching plant. This model consists of the patching robot itself as well as of a preceding and a subsequent conveyor used to feed in and move out the panels. This implies that the (unprocessed) panels already went through the defect scanner and their data is already saved in the wood knowledge repository, in particular the panel dimensions and their patch lists are already known. Before simulation results are presented, a quick overview of the dynamical model of the patching robot is given.

5.2.1 Mathematical model of the patching robot

The velocity-controlled conveyors in front and behind the patching robot are described by their maximum velocity v_γ and a rise time T_γ from zero to maximum velocity. The patching robot consists of a patching tool and two xy-tables. Since the patching tool is decoupled from the positioning process and it is already operational and working reliably, it is merely described by the time delay T_p, i.e. the time it takes to finish one patching action. The xy-tables are lowerable. Their transition between the low and the high position is also modeled by a time delay T_z. Engaging as well as releasing the clamping mechanism takes T_c seconds. It remains to derive the equations of motion of the xy-tables and the panel in the

xy-plane. To this end, Figure 2.2 should be recalled.

The state vector of an xy-table is given by $\mathbf{z}_{r,i} = \begin{bmatrix}\varphi_{r,i} & y_{r,i} & \dot{\varphi}_{r,i} & \dot{y}_{r,i}\end{bmatrix}^{\mathrm{T}}$, where $\varphi_{r,i}$ is the roller angle of its rubber belt and $y_{r,i}$ is the lateral position of the slide of the xy-table. The index $i \in \{1, 2\}$ stands for XY-Table 1 and XY-Table 2, respectively. Assuming viscous bearing friction, the equations of motion read as,

$$\begin{bmatrix}\ddot{\varphi}_{r,i} \\ \ddot{y}_{r,i}\end{bmatrix} = \begin{bmatrix}J_{r,i}^{-1}\left(M_{r,i} - d_{r\varphi,i}\dot{\varphi}_{r,i} - r_{r,i}F_{px,i,j}\right) \\ m_{r,i}^{-1}\left(F_{r,i} - d_{ry,i}\dot{y}_{r,i} - F_{py,i,j}\right)\end{bmatrix} \tag{5.1}$$

with $r_{r,i}$, $J_{r,i}$, $m_{r,i}$, $d_{r\varphi,i}$ and $d_{ry,i}$ denoting the effective radius of the sprockets driving the rubber belts, the total inertia in longitudinal direction, the total moving mass in lateral direction and the viscous friction parameters in longitudinal and lateral direction, respectively. The actuator inputs are the torque $M_{r,i}$ of the synchronous servo motor and the force $F_{r,i}$ of the synchronous linear motor. The force applied by XY-Table $i \in \{1, 2\}$ to Panel $j \in \{1, 2\}$ is denoted by $\begin{bmatrix}F_{px,i,j} & F_{py,i,j}\end{bmatrix}^{\mathrm{T}}$. In the same manner, the equations of motion for the panel take the form

$$\begin{bmatrix}\ddot{x}_{p,j} \\ \ddot{y}_{p,j}\end{bmatrix} = \begin{bmatrix}m_{p,j}^{-1}\left(F_{px,1,j} + F_{px,2,j}\right) \\ m_{p,j}^{-1}\left(F_{py,1,j} + F_{py,2,j}\right)\end{bmatrix}, \tag{5.2}$$

where $m_{p,j}$ is the mass of the panel j. The state vector of Panel j is given by $\mathbf{z}_{p,j} = \begin{bmatrix}x_{p,j} & y_{p,j} & \dot{x}_{p,j} & \dot{y}_{p,j}\end{bmatrix}^{\mathrm{T}}$.

Formally, the xy-tables and their respective panels are separate dynamical systems, each with its own inert mass and position. They are only connected via friction forces. Thus, the physically correct way to describe the interaction between these dynamical systems would be to establish an appropriate stick-slip model, whose parameters are not known and generally are prone to large variations, see Sections 2.2 and 4.2. However, the clamping mechanism of the rubber belts ensures high friction forces. Consequently, a stick-slip model between rubber belt and panel would only add unnecessary complexity to the mathematical model. Therefore, a simpler description is favorable. For this, the connection[5] between panel and robot is assumed to be fixed, i.e.

$$\begin{bmatrix}x_{p,j} \\ y_{p,j}\end{bmatrix} = \begin{bmatrix}r_{r,i}\varphi_{r,i} \\ y_{r,i}\end{bmatrix} + \begin{bmatrix}\epsilon_{x,i,j} \\ \epsilon_{y,i,j}\end{bmatrix}. \tag{5.3}$$

All the slip effects are summarized in an additive, two times continuously differentiable disturbance $\begin{bmatrix}\epsilon_{x,i,j} & \epsilon_{y,i,j}\end{bmatrix}^{\mathrm{T}} \in \left[\mathcal{C}^2(t)\right]^2$ between XY-Table i and Panel j. In lateral direction, the panels are guided almost perfectly, i.e. $\epsilon_{y,i,j} = 0$. Still it is kept in order to maintain generality.

[5] Note that in this model, the dynamics of the motor is neglected and a direct torque or force input is assumed. Therefore, also the transmission ratio of the gearbox i_x is not considered, i.e. $i_x = 1$.

Choosing (5.3), the model exhibits a switching behavior which depends on the longitudinal panel position. Three modes have to be distinguished. First, the xy-table is empty. Then, naturally the force acting on the panel disappears. Inserting $\begin{bmatrix} F_{px,i,j} & F_{py,i,j} \end{bmatrix}^{\mathrm{T}} = \mathbf{0}$ into (5.1) yields the dynamics for the empty xy-table.

Second, both xy-tables are coupled via one moving panel. Solving (5.3) for $\begin{bmatrix} \varphi_{r,i} & y_{r,i} \end{bmatrix}^{\mathrm{T}}$ and inserting into (5.1) yields the forces acting on the panel as functions of the panel coordinates, the actuator inputs and the disturbances, $\begin{bmatrix} F_{px,i,j}(x_{p,j},\, M_{r,i},\, \epsilon_{x,i,j}) & F_{py,i,j}(y_{p,j},\, F_{r,i},\, \epsilon_{y,i,j}) \end{bmatrix}^{\mathrm{T}}$. Inserting again into (5.2), the panel dynamics read as

$$\begin{bmatrix} \ddot{x}_{p,j} \\ \ddot{y}_{p,j} \end{bmatrix} = \begin{bmatrix} \bar{m}_{px,j}^{-1} \left(\frac{M_{r,1}}{r_{r,1}} + \frac{M_{r,2}}{r_{r,2}} - \left(\frac{d_{r\varphi,1}}{r_{r,1}^2} + \frac{d_{r\varphi,2}}{r_{r,2}^2} \right) \dot{x}_{p,j} + \bar{\epsilon}_{x,j} \right) \\ \bar{m}_{py,j}^{-1} \left(F_{r,1} + F_{r,2} - (d_{ry,1} + d_{ry,2})\, \dot{y}_{p,j} + \bar{\epsilon}_{y,j} \right) \end{bmatrix} \tag{5.4a}$$

with

$$\bar{m}_{px,j} = m_{p,j} + \frac{J_{r,1}}{r_{r,1}^2} + \frac{J_{r,2}}{r_{r,2}^2}, \tag{5.4b}$$

$$\bar{m}_{py,j} = m_{p,j} + m_{r,1} + m_{r,2}, \tag{5.4c}$$

$$\bar{\epsilon}_{x,j} = \frac{d_{r\varphi,1}}{r_{r,1}^2}\dot{\epsilon}_{x,1,j} + \frac{d_{r\varphi,2}}{r_{r,2}^2}\dot{\epsilon}_{x,2,j} + \frac{J_{r,1}}{r_{r,1}^2}\ddot{\epsilon}_{x,1,j} + \frac{J_{r,2}}{r_{r,2}^2}\ddot{\epsilon}_{x,2,j}, \tag{5.4d}$$

$$\bar{\epsilon}_{y,j} = d_{ry,1}\dot{\epsilon}_{y,1,j} + d_{ry,2}\dot{\epsilon}_{y,2,j} + m_{r,1}\ddot{\epsilon}_{y,1,j} + m_{r,2}\ddot{\epsilon}_{y,2,j}. \tag{5.4e}$$

The current states of the xy-tables are computed by solving (5.3) for the robot coordinates.

Third, an xy-table moves a panel alone. The corresponding equations of motion are easily obtained by neglecting the terms of the respectively other patching robot in (5.4).

This simple mathematical model is used for the simulation of the process logic control. The parameters of this model are derived from a CAD design of the manufacturer Springer AG. They are given in Chapter *Symbols and Parameters*. The real-time controllers are implemented as described in Section 4.1. Details are omitted for brevity. This simulation model is mainly used for verification of the process logic control.

5.2.2 Results and analysis

Figure 5.3 shows the way of three panels through the production line. Only the longitudinal motion x is shown, since the longitudinal position of the panels drives the state-machine. The origin of the coordinate system is placed at the center of the drill of the patching tool, as described in Section 2.4. The black dash-dotted lines indicate the position of the different machines, i.e. the patching tool *tool*, the xy-tables xyT_i and the conveyors cnv_i, $i \in \{1, 2\}$. In between the xy-tables

XY-Table 1		XY-Table 2	
0	Idle	0	Idle
1	Feed-In	1	Coupled Enter
2	Position	2	Coupled Position
3	Patch	3	Coupled Patch
4	Coupled	4	Coupled Exit
		5	Position
		6	Patch
		7	Move-Out

Table 5.1: Numbering of the states as used in Figure 5.4. The Initial State is not considered in the simulation model.

and the patching tool a small gap of 5cm is explicitly considered. At time $t = 0$ three panels are at Conveyor 1. The solid lines represent their front end $x_{p,j}$, the dashed-dotted lines $x_{p,j} - L_j$, $j \in \{1, 2, 3\}$ their rear end.

The position tracking cameras have a rather narrow field of view in terms of x-range. The cameras cam_i, $i \in \{1, 2\}$ track the panels as long as they are partly in the x-range of $[-0.350, -0.175]$ and $[0.175, 0.350]$, respectively. The gap $[-0.175, 0.175]$ in between the camera coverage is caused by the patching tool. However, it is only 0.35m wide, so that even the shortest panel (of 1m length) is permanently tracked by at least one of the cameras during positioning. The vertical lines of the camera signals, as marked by t_{c1s}, t_{c1e}, t_{c2s} and t_{c2e}, indicate the time intervals in which the cameras track the panel position. They are clearly overlapping. Take, for example, Panel 1. Its front end $x_{p,1}$ enters the field of view of Camera 1 at t_{c1s}, i.e. $x_{p,1} \geq -0.350$m. From then on, its position is tracked until its rear end $x_{p,1} - L_1$ leaves the field of view at t_{c1e}, i.e. $x_{p,1} - L_1 \geq -0.175$m. Camera 2 tracks $x_{p,1}$ in the time interval $[t_{c2s}, t_{c2e}]$. The intervals $[t_{c1s}, t_{c1e}]$ and $[t_{c2s}, t_{c2e}]$ overlap.

Whenever the panels come to a halt at the xy-tables, it happens for one of the following five reasons: First, because of the Feed-In Procedure ①, second, because of the transition from XY-Table 1 positioning independently into coupled positioning ②, third, because of the transition from coupled positioning into XY-Table 2 positioning independently ③, fourth, because of the Move-Out Procedure ④ and, fifth, because of patching, which are all the remaining positions not marked.

Figure 5.4 lists the essential signals of the process logic control. The signals st_1 and $-st_2$ show the state transitions of XY-Table 1 and XY-Table 2 according to Table 5.1. The minus indicates that the state of XY-Table 2 is plotted on the negative y-axis; the same holds true for the signals $-elv_2$ and $-clp_2$.

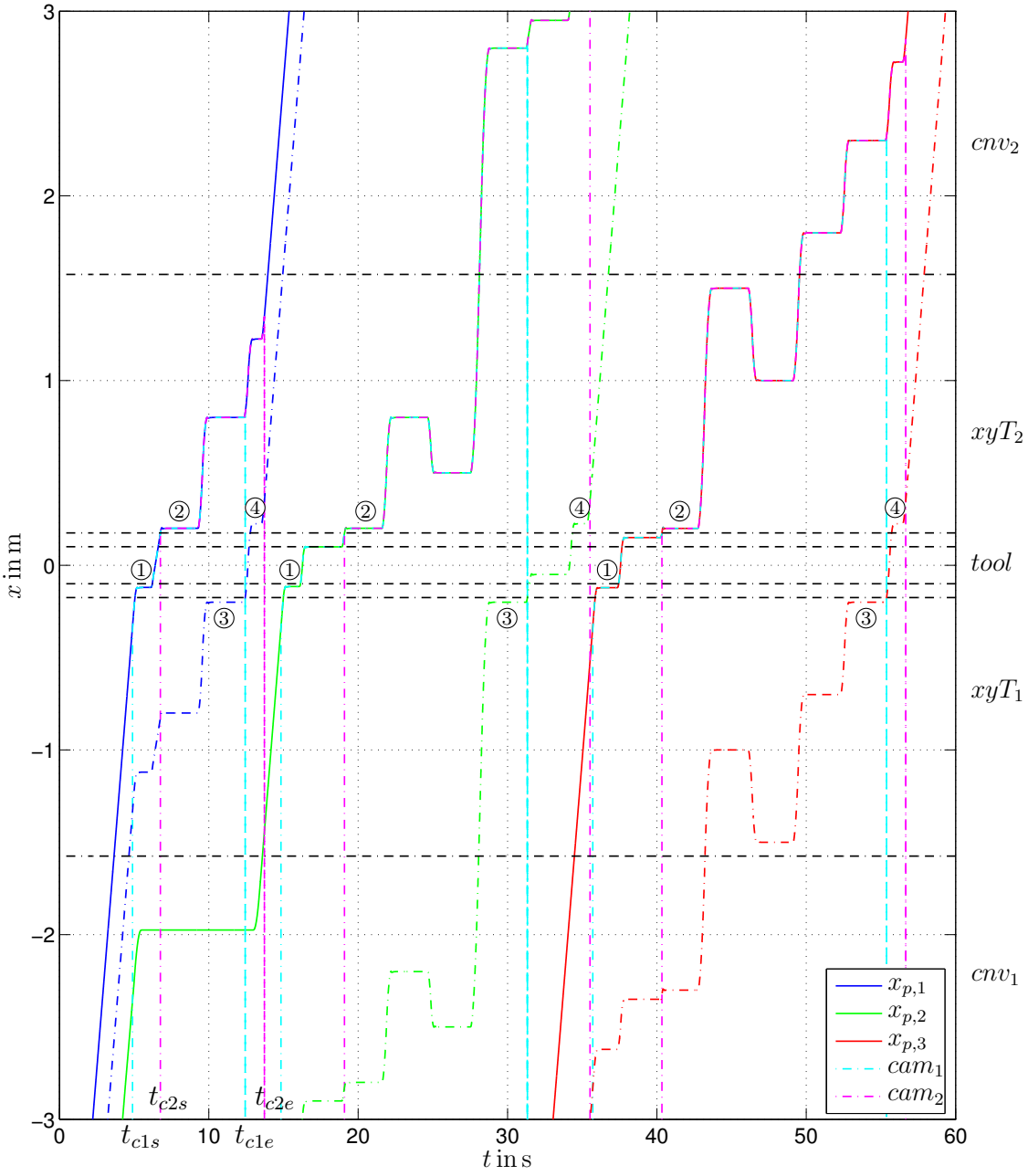

Figure 5.3: Three panels on their way through the production line. Panel 1,2,3 are of length 1m, 3m and 2.5m, respectively. Panel 1 requires zero patches, Panel 2 requires four at $x_{p,2} = \{0.10, 0.80, 0.50, 2.95\}$ and Panel 3 also requires four at $x_{p,3} = \{0.15, 1.50, 1.00, 1.80\}$. Abbreviations: *tool* ... patching tool, xyT_1 ... XY-Table 1, cnv_1 ... Conveyor 1, $x_{p,1}$ ... front end of Panel 1, cam_1 ... Position Tracking Camera 1.

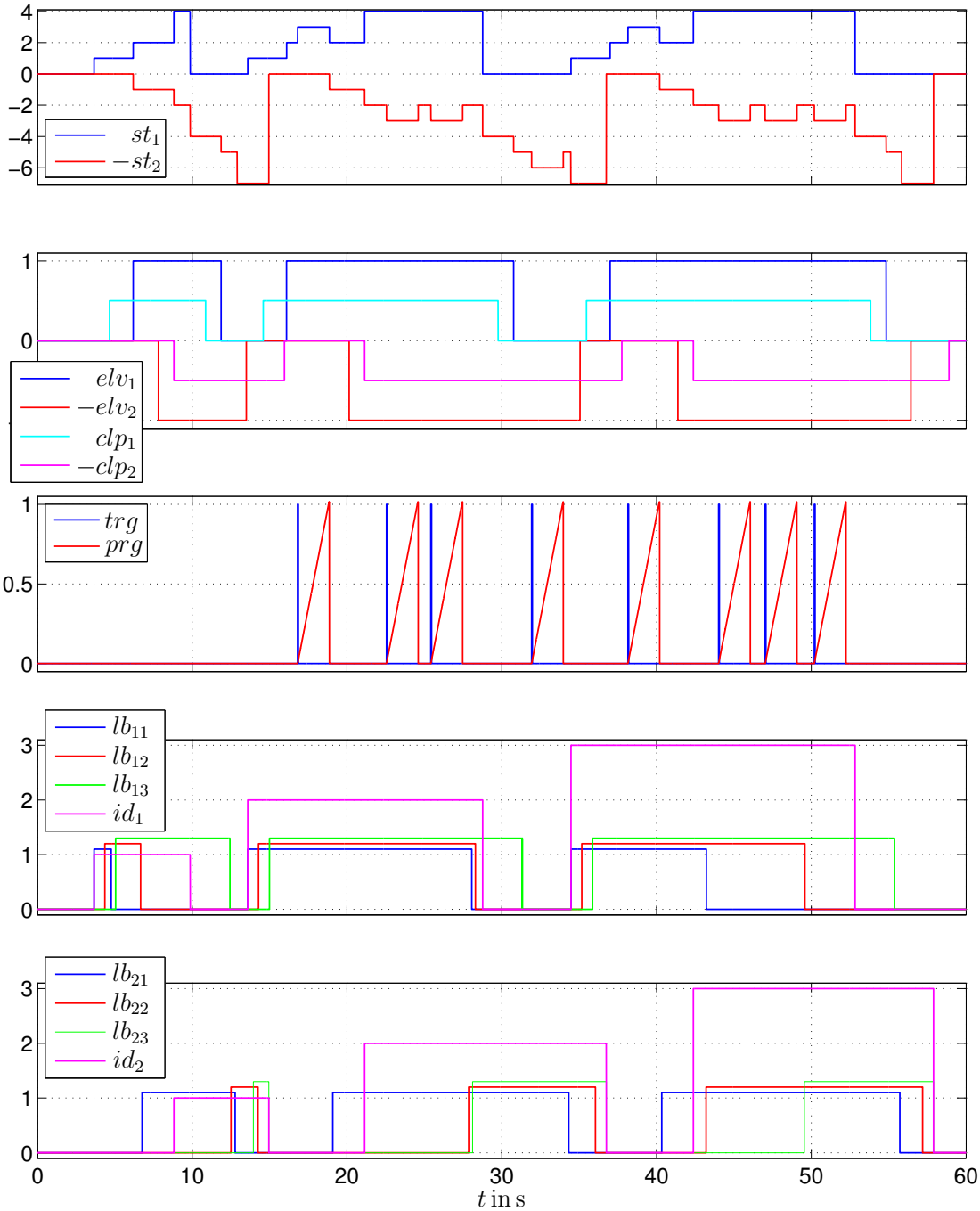

Figure 5.4: Signals of the process logic control. Abbreviations: st_1 ... state of xyT_1, elv_1 ... xyT_1 is elevated, clp_1 ... clamping mechanism of xyT_1 is engaged, trg ... patching tool triggered, prg ... progress of the patching action, lb_{12} ... Light Barrier 2 of xyT_1, id_1 ... panel assigned to xyT_1.

The signals elv_i, $i \in \{1, 2\}$ and clp_i show if the xy-tables are elevated and if the clamping mechanism of the rubber belts is engaged[6]. The signal trg shows when the patching tool is triggered and prg shows the progress of the patching action. The signals lb_k, $k \in \{11, 12, 13, 21, 22, 23\}$ show if the light barriers are interrupted. They are numbered according to Figure 2.2. The signals id_i show which panel is currently assigned to XY-Table i. Note that the high level of the logical signals, i.e. elv_i, clp_i, lb_k, are scaled to avoid overlapping.

With the help of Figure 5.3 and 5.4, the process logic control shall be explained exemplarily. To begin with, it is described how Panel 1 is moved from one side of the patching robot to the other. It exhibits zero defects, so the focus is on Feed-In, the transitions from independent into coupled positioning and Move-Out.

In Idle State both xy-tables have their clamping mechanism released, they are lowered and in Feed-In or Move-Out Position, respectively. Panel 1 approaches XY-Table 1 from the Feed-In Conveyor. As soon as it interrupts Light Barrier 11, the state-machine of XY-Table 1 transitions into Feed-In State. On entering this state, the respective panel data is requested from the wood knowledge repository and assigned to XY-Table 1, see id_1. The clamping mechanism is engaged, which takes T_c seconds, as can be inferred from the time delay between $st_1 = 1$ and $clp_1 = 1$. During this time, XY-Table 1 adjusts to the Feed-In Velocity of Conveyor 1. In Feed-In State, the rubber belts are velocity-controlled, since the cameras do not track the panel position yet and the exact position is irrelevant anyway. The panel is moved forward until it reaches the end of XY-Table 1, i.e. Light Barrier 13 is interrupted. Then, XY-Table 1 is ready to be elevated. However, first, XY-Table 2 has to transition into either Idle or Move-Out State. This indicates that processing of the panel at XY-Table 2 is finished and, consequently, it is not going to occupy the patching tool in the future. Elevation takes T_z seconds, as can be inferred from the time delay between $lb_{13} = 1$ and $elv_1 = 1$.

As soon as XY-Table 1 is elevated, it transitions into Position State. Panel 1 does not require any patches and thus the next position to approach is Handover Position 1 at $x = 0.2$m. Therefore, after positioning the panel accurately, XY-Table 1 just waits, since there is no transition condition for this case in XY-Table 1, Position State. If XY-Table 2 is idle, which it is, Handover Position 1 is sent to its state-machine and it is triggered to switch to Coupled Enter State.

In Coupled Enter State, XY-Table 2 moves to Handover Position 1, elevates itself and clamps the panel. As a safety precaution, the positioning of both xy-tables is checked. If the clamping mechanism is engaged, the panel data set of XY-Table 1 is assigned to XY-Table 2 as well. Also, XY-Table 1 is triggered to transition into Coupled State and XY-Table 2 transitions into Coupled Position State. In XY-Table 1, Coupled State nothing is done. All the compu-

[6] As already mentioned in Section 5.2.1, the actions of elevating/lowering and engaging/releasing the clamping mechanism take T_z and T_c seconds, respectively.

tations for coupled positioning and patching are carried out in the state-machine of XY-Table 2. On entering Coupled Position State, first, the longitudinal belt positions of both xy-tables are reset to the position yielded by Camera 1. For coupled positioning, the master-slave concept is employed. Both, the reset of the belt position and the master-slave concept are discussed in Section 4.1. The next position is Handover Position 2 at $x_{ho2} = 0.8$m. After positioning the panel accurately, XY-Table 2 switches to Coupled Exit State and XY-Table 1 is triggered to switch to Idle State.

As soon as XY-Table 1 enters Idle State, the panel data is deleted from the state-machine. XY-Table 1 releases the clamping mechanism, lowers itself and moves to the Feed-In Position, ready to accept Panel 2.

In Coupled Exit State, XY-Table 2 does nothing, but wait until XY-Table 1 is lowered. Then, XY-Table 2 switches to Position State, where the Move-Out Position of Panel 1 is approached. As soon as Panel 1 left the workspace of the patching tool, i.e. Light Barrier 21 is not interrupted anymore, XY-Table 2 switches to Move-Out State.

In Move-Out State, XY-Table 2 lowers itself and the rubber belt adjusts to the Move-Out Velocity of the conveyor. The panel is moved forward until no light barrier of XY-Table 2 is interrupted anymore. Conveyor 2 is activated permanently. However, it only moves the panel, if XY-Table 2 is lowered and the panel actually contacts the conveyor. Then, XY-Table 2 transitions into Idle State and awaits the next Handover Position 1 from the subsequent Panel 2.

After discussing the positioning process, the integration of the patching process into the positioning process is explained with the help of Panel 2. It requires four patches at $x_{p,2} = \{0.10, 0.80, 0.50, 2.95\}$. Considering the handover positions at $x_{ho1} = 0.20$m and $x_{ho2} = 2.80$m, XY-Table 1 approaches Patch Position 1. Positions 2 and 3 are processed in the Coupled States, which leaves Position 4 to XY-Table 2. Processing a patch location consists of two steps, first, positioning the panel and, second, the patching action itself. Positioning is already discussed above. In XY-Table 1, Position State, XY-Table 2, Coupled Position State and XY-Table 2, Position State, the transition conditions distinguish between Patch Positions, Handover Position 1, 2 and Move-Out Position. If it is a Patch Position, the state-machines transition into their respective Patch States, i.e. XY-Table 1, Patch State, XY-Table 2, Coupled Patch State and XY-Table 2, Patch State. The Patch State shall be explained exemplarily for XY-Table 1, however, it works in the same way for the other two. As soon as the first patch position $x = 0.10$m is approached correctly, XY-Table 1 transitions from Position to Patch State. For safety reasons, the real-time controllers of XY-Table 1 are switched off before the patching tool is activated, see trg. The patching process takes T_p seconds, as can be inferred from prg. After successful completion, XY-Table 1 switches back to Position State. When returning to Position State, the real-time controllers are reset to the current panel position, before they are turned back on again, see Section 5.1.

CHAPTER 6

Measurement results of the patching robot

In Chapters 4 and 5, the control strategies for the patching robot are described. This chapter documents their feasibility in an industrial environment. During an extensive test phase[1], several hundred panels were patched successfully, while making final adjustments to both, the design of the patching robot and its control algorithms. Measurement data collected during these tests are presented.

To begin with, key data of the real-time system, the position sensors and the actuators of the patching robot shall be listed. The velocity controllers of the electrical drives and their underlying current controllers, which cannot be accessed, are provided by the manufacturer SEW Eurodrive and directly implemented at the frequency converters. This enables a sampling time of $T_s = 1\text{ms}$ for the velocity and $T_{si} = 0.125\text{ms}$ for the current controllers. The position control algorithms and the process logic control are implemented in C++ at an industrial computer. The computer communicates with the patching plant via Beckhoff field bus hardware. TwinCAT V3 real-time software grants the Windows-based computer real-time capability with a sampling time of $T_s = 1\text{ms}$.

Position measurement is accomplished using two sources. First, taking into account the transmission ratio $i_x = 3$ of the gearbox and the effective radius $r_r = 0.036\text{m}$ of the sprockets driving the rubber belts, the incremental encoders of the synchronous servo motors measure the longitudinal position of the rubber belts x_r with a resolution of $\rho_{xr} = (4096 i_x)/(2 r_r \pi) = 5.4\text{e}4\text{m}^{-1}$. In lateral direction y_r, the absolute position encoders of the linear motors have a resolution of $\rho_{yr} = 1.0\text{e}5\text{m}^{-1}$. Second, the absolute longitudinal position of the panel x_p shall be visually tracked by a smart camera at each xy-table. However, in order to decouple the development of the real-time control from the development of the algorithm for visual position tracking, two laser interferometers shall be used instead. As soon as the panel to be tracked interrupts Light Barrier 13 (of

[1] A video of the entire patching process is available at
www.acin.tuwien.ac.at/fileadmin/cds/videos/hofmairBoeckKugi2015_woodPatchingRobot.wmv

XY-Table 1) for the first time, the position of the interferometer is reset to the respective position, similar to Section 2.4. It tracks the panel with an accuracy of $\rho_{xp} = 1.0\text{e}4\text{m}^{-1}$.

The synchronous servo motor of the rubber belts provides a nominal torque of 2.9N m. Considering the transmission ratio of the gearbox $i_x = 3$, a torque of $M_n = 8.7\text{N m}$ is available at the sprocket of the rubber belt. In lateral direction, the slide of an xy-table can be moved with a nominal force of $F_n = 560\text{N}$.

In order to maximize the throughput, the trajectories shall optimally exploit the potential of the actuators. The following parameters for the Bang-Bang Trajectory Generator, presented in Section 3.3.2, are chosen. In longitudinal direction, the maximum velocity[2] is $v_{Mx} = 1\text{m/s}$. Given an effective radius $r_r = 0.036\text{m}$ of the sprockets, this yields $\dot{\varphi}_M = 27.778\text{s}^{-1}$. Utilizing 70% of the nominal torque for acceleration yields a maximum angular acceleration of

$$\ddot{\varphi}_M = \frac{0.7 M_n}{J_r + m_p r_r^2} = 219.380\text{s}^{-2}, \tag{6.1}$$

where $J_r = 0.015\text{kg m}^2$ is the moment of inertia of the entire drive shaft and $m_p = 10\text{kg}$ is the mass of an average panel. Assuming a rise time for the actuator torque of $T_\varphi = 20\text{ms}$ provides an estimate of the maximum jerk $\dddot{\varphi}_M = 10969\text{s}^{-3}$. Similar considerations yield the parameters of the trajectory generator in lateral direction, $\dot{y}_M = 0.500\text{m/s}$, $\ddot{y}_M = 0.7F_n/(m_r + m_p) = 2.896\text{m/s}^2$ and $\dddot{y}_M = 144.800\text{m/s}^3$, where the mass of the slide of an xy-table, i.e. the moving mass in lateral direction, is $m_r = 125.375\text{kg}$. All this key data of the plant is summarized in Chapter *Symbols and Parameters*.

In the following, four experiments are conducted at the prototype plant. First, friction in the drivetrain of the rubber belts is documented. Second, the master-slave control is analyzed. Third, measurement results of the patching process of an entire panel demonstrate the feasibility of the real-time as well as the process logic control. Fourth, the functionality of the Trajectory Updating Algorithm is assessed based on these measurements.

6.1 Friction in the drivetrain of the rubber belts

In order to establish the challenging friction conditions in the drivetrain of the rubber belts, Figure 6.1 exemplarily shows a friction curve that was recorded moving a 3m-panel in longitudinal direction. The torque $M_{r,1}$ converted to the output shaft of the drivetrain is plotted versus the stationary translational velocity $\dot{x}_{r,1} = r_r\,\dot{\varphi}_{r,1}$ of the rubber belts. The red lines indicate the nominal torque of the synchronous servo motor.

[2] In this chapter, in order to avoid unnecessary indices, the following notation is used, $v_x = \dot{x} = r_r\dot{\varphi}_r$, $v_y = \dot{y}$, in particular, $v_{Mx} = \dot{x}_M = r_r\dot{\varphi}_M$ and $v_{ty} = \dot{y}_t$, etc.

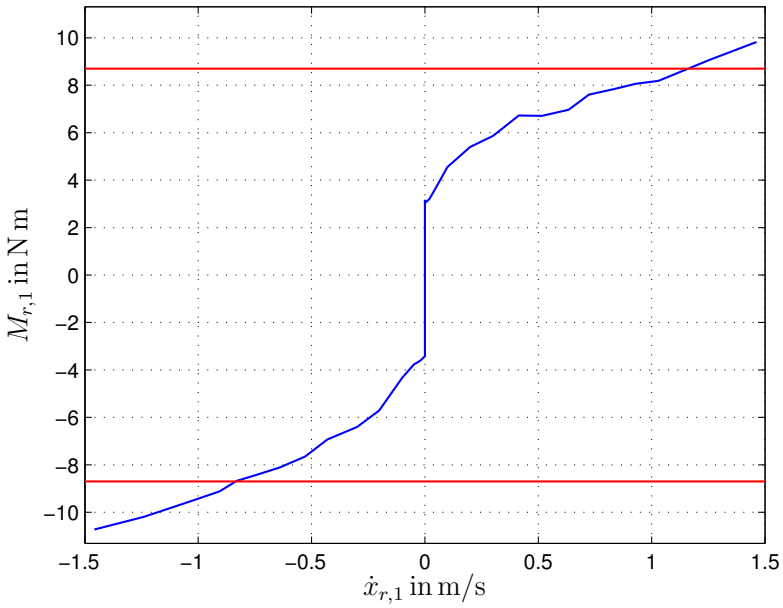

Figure 6.1: Friction characteristic of longitudinal motion of XY-Table 1 moving a 3m-panel.

Static friction values are in the range of 40% of the nominal torque, but might go up to 60% in less common cases. At the envisaged positioning velocity of 1m/s the actuator already utilizes 100% of its nominal torque.

6.2 Coupled positioning

Another challenge for the real-time control is coupled positioning by means of a master-slave concept. The following measurement results demonstrate its feasibility. Figure 6.2 shows the longitudinal motion of a positioning action in Coupled State. It starts at $x_{r,1} = 1.1$m and leads to $x_{r,1} = 1.6$m. Since this experiment is about synchronization behavior of the xy-tables (and not about panel positioning), $x_{p,1} = x_{r,1}$ is assumed and merely the robot is positioned. The desired and the actual trajectory of XY-Table 1 are denoted by x_t, $x_{r,1}$ and $\dot{x}_t$, $\dot{x}_{r,1}$ for position and velocity, respectively. Also, the applied motor torques $M_{r,1}$, $M_{r,2}$ and the difference between the actual positions $x_{r,1} - x_{r,2}$ and velocities $\dot{x}_{r,1} - \dot{x}_{r,2}$ of the xy-tables are depicted.

As described in Section 4.1, during coupled positioning, XY-Table 1 is the master and XY-Table 2 the slave. In longitudinal direction, the master is position controlled. The slave merely follows the velocity of the master, $\dot{x}_{t,2} - e_{\dot{x},2} = \dot{x}_{t,1} - e_{\dot{x},1}$. Thus, ideally, they move at exactly the same velocity, i.e. $\dot{x}_{r,1}(t) - \dot{x}_{r,2}(t) \equiv 0$.

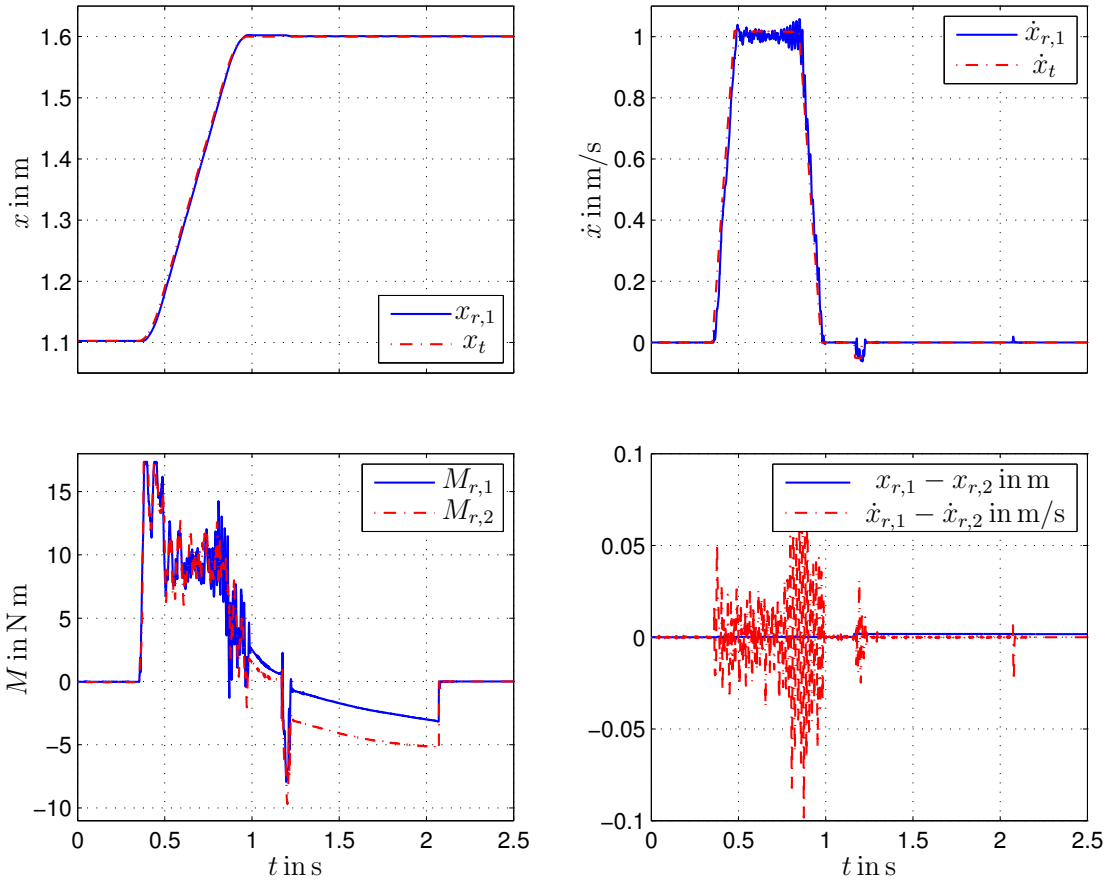

Figure 6.2: Coupled positioning, longitudinal direction.

In reality, small high-frequency oscillations of the velocity signal $\dot{x}_{r,1}$ also cause slight deviations in the velocity signals $\dot{x}_{r,1} - \dot{x}_{r,2}$ of the two xy-tables. On average, they are insignificant. Consequently, also the difference in their positions $x_{r,1} - x_{r,2}$ is virtually zero over the entire period.

The same holds true for the torque signals; ideally they are identical. In reality, since the friction properties are not identical, there are also deviations between the torque signals. However, their sign is always the same, meaning the rubber belts cooperate in accelerating the panel. They do not work against each other. Furthermore, one can see that during the phase of constant velocity $\dot{x}_t = 1\text{m/s}$, i.e. between $t = 0.5\text{s}$ to $t = 0.8\text{s}$, the actuator utilization is approximately 9N m or 100% for both xy-tables, compare Figure 6.1.

Figure 6.3 shows the lateral motion of this positioning action in coupled mode, starting at $y_{r,1} = 0.05\text{m}$ leading to $y_{r,1} = 0.25\text{m}$. Again, the desired and the actual trajectory of XY-Table 1 are denoted by y_t, $y_{r,1}$ and $\dot{y}_t$, $\dot{y}_{r,1}$ for position and velocity, respectively. Additionally, the applied motor forces $F_{r,1}$, $F_{r,2}$ and

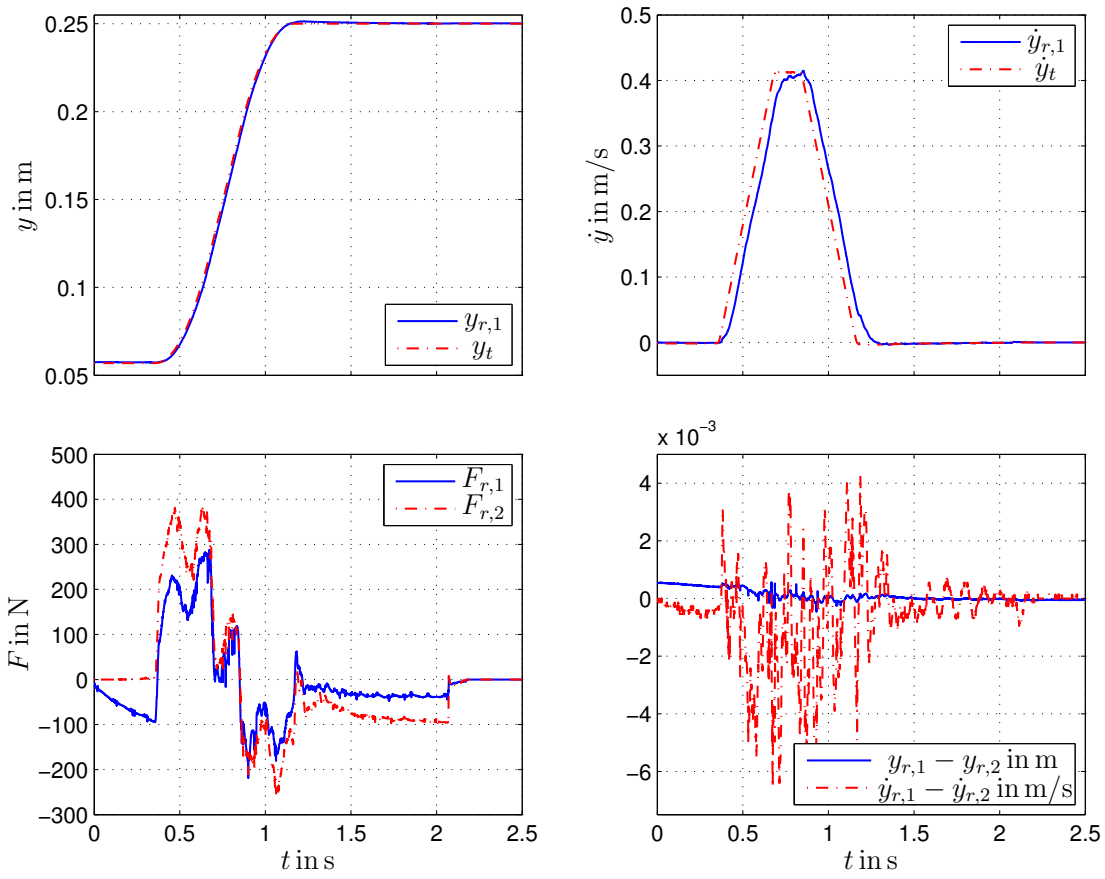

Figure 6.3: Coupled positioning, lateral direction. Note that in this experiment the trajectories are planned with $\dot{y}_M = 0.400\text{m/s}$, $\ddot{y}_M = 0.4F_n/(m_r + m_p) = 1.655\text{m/s}^2$ and $\dddot{y}_M = 82.750\text{m/s}^3$.

the difference between the actual positions $y_{r,1} - y_{r,2}$ and velocities $\dot{y}_{r,1} - \dot{y}_{r,2}$ of the xy-tables are depicted.

Laterally, both xy-tables move independently during coupled positioning because the coupling is less firm in this direction. Also, friction forces in the rails of the slides are much smaller than in the drivetrain of the rubber belts, so trajectory tracking is very accurate. Thus, the slides move in an almost identical fashion independently of each other. A lag error of approximately 0.05s in tracking the velocity trajectory can be observed for both xy-tables. Apart from that, the velocity difference $\dot{y}_{r,1} - \dot{y}_{r,2}$ exhibits amplitudes of approximately $0.01\dot{y}_M$. On average, they are almost zero, therefore, negligible. There is a small initial position difference of approximately $y_{r,1} - y_{r,2} = 0.5\text{mm}$. It vanishes at the end of the positioning action.

However, this small difference already causes a counter action of the actu-

ator force $F_{r,1}$ of XY-Table 1. In general, there is a noticeable difference in the amplitudes of the force signals $F_{r,1}$ and $F_{r,2}$. Still, their sign is always identical and so the slides always cooperate in accelerating the panel.

After approximately 1.4s the panel is positioned correctly. After further 0.5s settling time, controllers are switched off and the patching action is triggered. Thus, it is guaranteed that the real-time controllers of the xy-tables do not interfere with the patching action. Considering the positioning accuracy of 1mm and the high friction forces, the settling time of the controllers of approximately 0.5s is deemed necessary.

6.3 Positioning performance and actuator utilization

To demonstrate the efficiency of the proposed control concept, this section analyzes the measurement results of the patching process of the 2m-panel depicted in Figure 3.9. Apart from the twelve patch positions, two handover positions have to be approached, one for entering Coupled State and one for leaving it again, see Section 5.1.

Figure 6.4 presents the measurement data of longitudinal positioning. The panel position $x_{p,1}$ at XY-Table 1 as measured by its laser interferometer is compared to the position trajectory x_t. Also, the translational velocity of the rubber belts $\dot{x}_{r,1}$ is compared to the velocity trajectory $\dot{x}_t$. Finally, the actuator torque $M_{r,1}$ is given. The horizontal red lines indicate the nominal torque M_n. Subplots zoom in on three positioning actions. The scalings of the axis of these subplots are given in the respective units.

Apart from positioning accuracy, which will be discussed in Section 6.4, the first thing to notice is the duration of the positioning actions. On average, they take no longer than a few tenths of a second. The major part of the processing time is consumed by the patching tool, but also by lowering/elevating the xy-tables and releasing/engaging the clamping mechanism, see Section 5.1. This can be inferred from the long rest periods. Particularly challenging are small traveling distances in the range of centimeters or even less. Trajectory times of less than 0.1s can be the result. This is a very short time for a mechanical system of this size, especially in regard of the high friction forces. Trajectory tracking is less accurate in these cases. However, the stationary positioning accuracy is unaffected by trajectory length, $\lim_{t\to\infty} |x_{r,i}(t) - x_t(t)| \leq 1\text{mm}$, $i \in \{1, 2\}$. Generally speaking, positioning speed is satisfactory for the whole range of traveling distances from 0.001m to 3m for both, independent and coupled positioning.

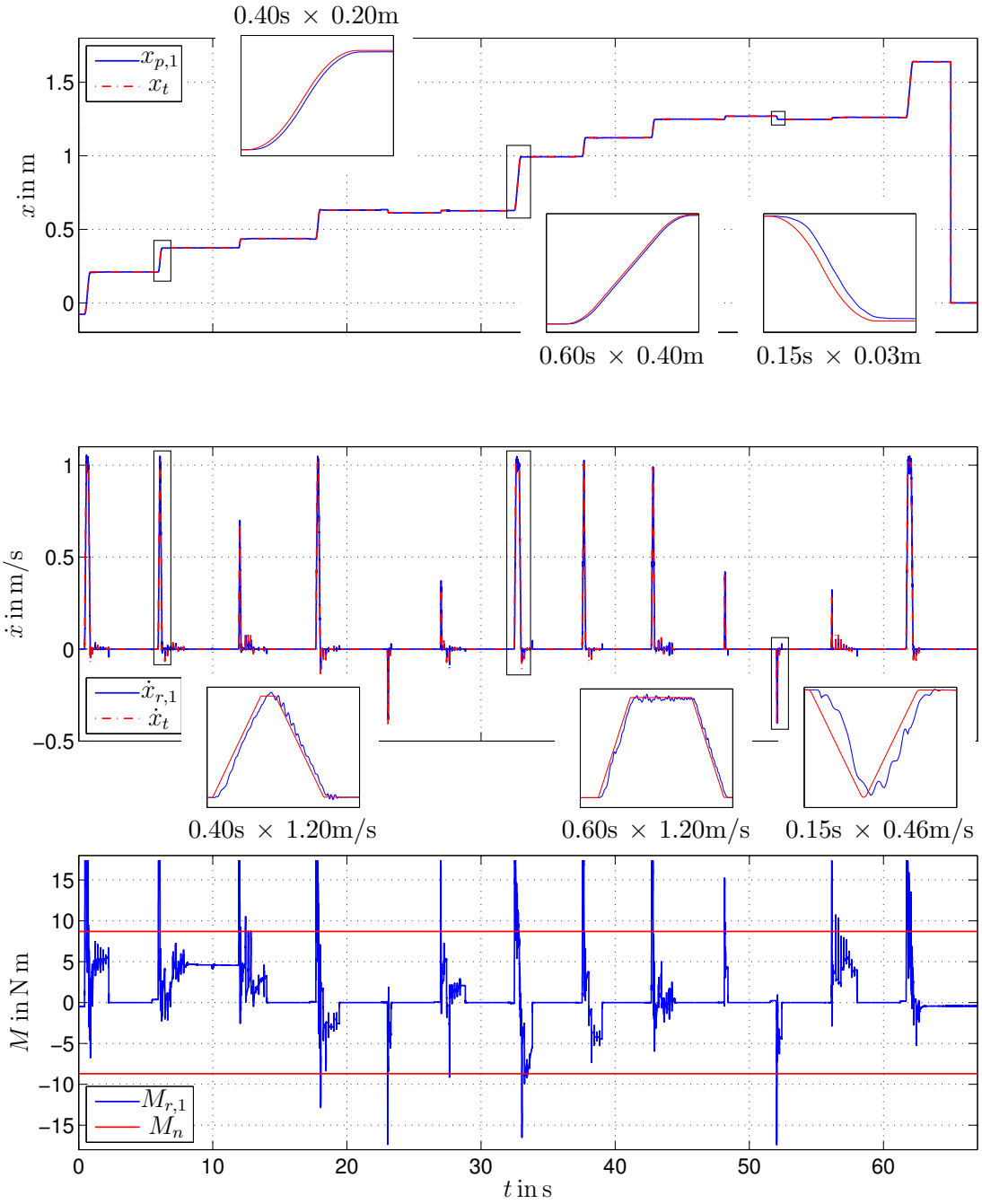

Figure 6.4: Time evolutions of processing the 2m-panel depicted in Figure 3.9, x-direction. The measurement data of XY-Table 1 is presented. Similar results are achieved for XY-Table 2.

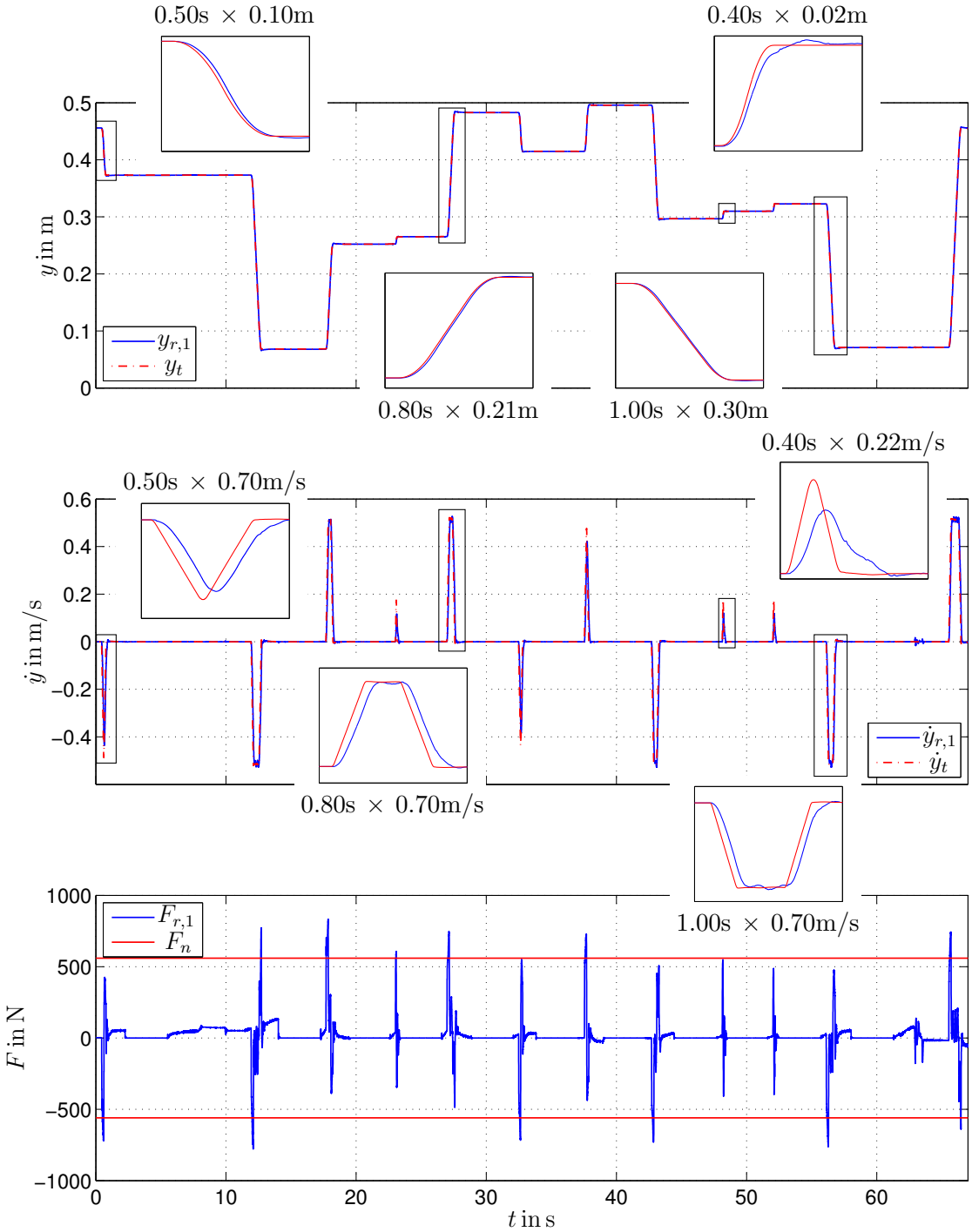

Figure 6.5: Time evolutions of processing the 2m-panel depicted in Figure 3.9, y-direction. The measurement data of XY-Table 1 is presented. Similar results are achieved for XY-Table 2.

Overall, tracking of the velocity trajectory is quite good. However, small ripples of the velocity signal cannot be avoided. This effect becomes more evolved at low speeds.

Consequently, high amplitudes around an already high mean value of the torque signal are observed, even at constant velocities. At constant maximum velocity v_{xM}, the actuator utilization oscillates between 50% to 150%. During acceleration, the nominal torque is exceeded by a factor of 2. This is possible because the relative duty cycle is no higher than 33%. Positioning actions rarely take longer than one second, i.e. $\max(\bar{T}_x, \bar{T}_y) < 1\text{s}$, whereas a patching action takes $T_p = 2\text{s}$. During patching, the controllers are switched off and the actuators have time to cool off. Using the motors to their full potential is necessary to maximize the throughput of the plant.

Moreover, two important details require mention. First, during the transition into Coupled State, at approximately $t = 10\text{s}$, the real-time controllers are not switched off. During this time, the motor torque is at almost $M_{r,1} = 5\text{N m} = 0.57M_n$ while $\dot{x}_{r,1} = 0\text{m/s}$, which again illustrates the friction in the drivetrain of the rubber belts.

Second, the instantaneous change in the longitudinal position x_p of the panel at the end of the measurement is a result of the process logic control. As soon as the panel reaches Handover Position 2 at $x_p = 1.642\text{m}$, at approximately $t = 63\text{s}$, it is handed over to XY-Table 2. Thus, the panel's position at XY-Table 1 is not defined any longer and the position of the rubber belt $x_{r,1}$ and the laser interferometer $x_{p,1}$ of XY-Table 1 are reset to zero.

Figure 6.5 presents the measurement data of lateral positioning. Position $y_{r,1}$ and velocity $\dot{y}_{r,1}$ of the slide are compared to their respective trajectory y_t and $\dot{y}_t$. Also, the actuator force $F_{r,1}$ is given with the horizontal red lines indicating the nominal force F_n.

Positioning performance in lateral direction is also very good. A stationary positioning accuracy of $\lim_{t\to\infty} |y_{r,i}(t) - y_t(t)| \leq 1\text{mm}$, $i \in \{1, 2\}$ is achieved within a positioning time of a few tenths of a second and with a maximum overshoot of 2mm over the nominal target value, i.e. 1mm outside the required positioning accuracy. Tracking of the velocity trajectory is characterized by a lag error of approximately 50ms, which is already observed in Section 6.2. For long traveling distances, this does not play a role. However, for short distances, this error can make up 30% of the trajectory time and thus impairs the tracking behavior. This becomes apparent when comparing the four zoomed trajectories. The two long trajectories, that include a phase of constant maximum velocity, show very good tracking of the velocity profile, while the tracking performance visibly goes down as the traveling distance is reduced. In this case, the slide of the xy-table is still accelerating, while the trajectory already reduces the velocity. The maximum force of the linear motors is around $1.5F_n$.

6.4 Trajectory Updating

In the previous section, measurement data is analyzed without covering the most critical issue, longitudinal positioning accuracy. It is directly related to the Trajectory Updating Strategy presented in Section 4.2.2, see also [47].

The final patching robot design is equipped with two smart cameras, one for each xy-table. They are used to track the longitudinal panel position because slippage occurs between the rubber belts and the panel. For reasons mentioned above, at the prototype patching robot, the cameras are replaced by laser interferometers. They track the absolute panel position continuously, i.e. at a sampling time of $T_s = 1$ms. In order to emulate the computationally expensive visual position tracking algorithm, the signal of the interferometer is only accessed at a sampling time of $T_{sv} = 200$ms and with a time delay of $T_{dv} = 100$ms, meaning the envisaged camera takes a picture every 200ms, which takes 100ms to process. Considering that only one-dimensional motion needs to be tracked and the algorithm tracks a known object in a known scene, this is a very conservative estimate. This way, the laser interferometers can be replaced by the visual position tracking algorithm without modifications or loss of performance. In the following, two experiments are presented.

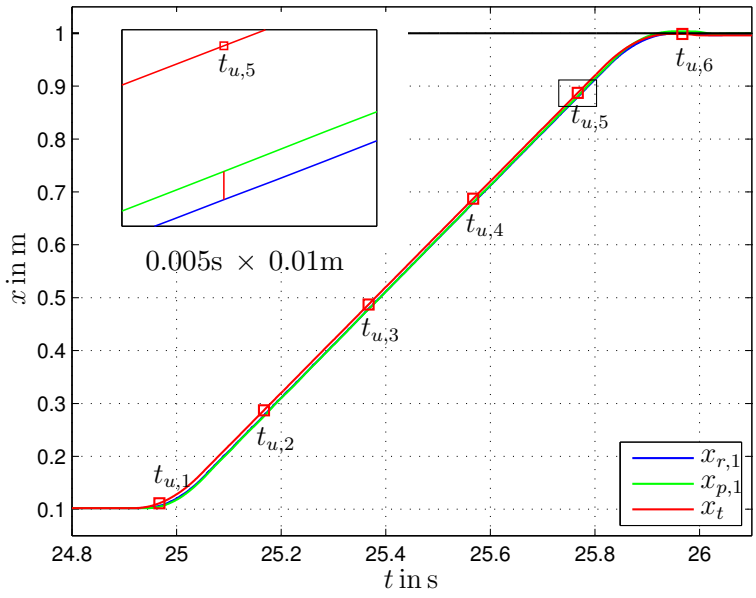

Figure 6.6: Experiment 1: Position trajectories, overview.

In the *first experiment*, the clamping pressure is reduced to half of its nominal value in order to induce high slip between the panel and the rubber belts. Positioning a 3m-panel of 20kg over a distance of 0.9m is depicted in Figure 6.6. As one can see, there are slight deviations between robot position $x_{r,1}$ and panel

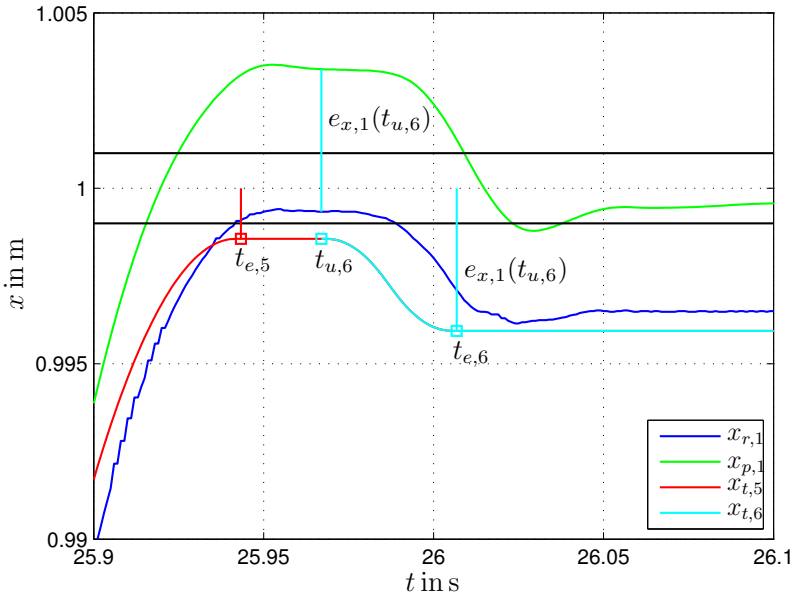

Figure 6.7: Experiment 1: Position trajectories, detail of Figure 6.6. The solid black lines mark the target region.

position $x_{p,1}$. Typically, the panel lags behind during the acceleration phase and there is a small overshoot of the panel position during the deceleration phase. The overall trajectory x_t is composed of seven trajectories, one point-to-point trajectory $x_{t,0}$ and six updates of the trajectory $x_{t,i}$, $i = \{1, \ldots, 6\}$. The red squares indicate the times $t_{u,i}$ of the respective update.

Figure 6.7 depicts the trajectory updates $x_{t,5}$ and $x_{t,6}$ in detail. The start of the fifth trajectory $t_{u,5}$ is depicted in the zoom of Figure 6.6 and $t_{e,5}$ marks the end of the trajectory. Due to the aggressive deceleration phase, the panel slips forward even more and trajectory $x_{t,5}$ needs to be corrected a final time. Trajectory $x_{t,6}$ stops 4mm short of the nominal desired end value because the panel is ahead of the rubber belts. Note that in this simplified experiment, the current panel position is used to compute the position deviation, meaning $T_{dv} = 0$s. The trajectory updates $x_{t,i}$ are performed using the current offset $e_{x,1}(t_{u,i} - T_{dv})|_{T_{dv}=0} = (x_{r,1}(t_{u,i} - T_{dv}) - x_{p,1}(t_{u,i} - T_{dv}))|_{T_{dv}=0}$ between robot and panel position, see, e.g., $i = 6$.

The *second experiment* corresponds to the exact production scenario, i.e. 100% clamping pressure, the most common 2m-panel with the patch locations and the robot path according to Figure 3.9 and Figure 6.4, and a time delay $T_{dv} = 0.1$s. Figure 6.8 shows the final approach of the positioning action starting at $x_0 = 0.434$m and leading to $x_d = 0.634$m. The absolute panel position is acquired at time instants $t_{u,i} - T_{dv}$, $i = \{1, 2\}$. After T_{dv}, i.e. at $t_{u,i}$, this information is used to compute new trajectories leading to the respective modified end points

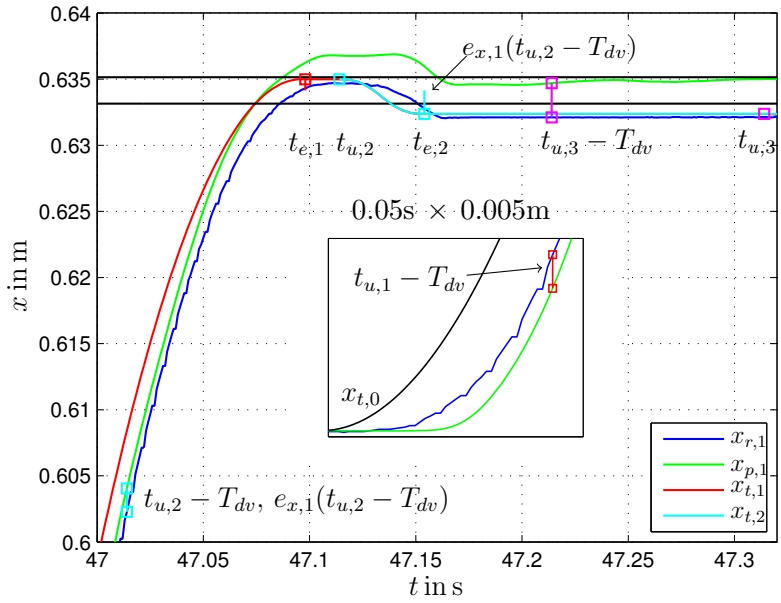

Figure 6.8: Experiment 2: Position trajectories, detail. The window inside the graph shows the beginning of the positioning action, where the black curve depicts the original point-to-point trajectory.

$t_{e,i}$, see $e_{x,1}(t_{u,2} - T_{dv})$. The first absolute position measurement is acquired during the acceleration phase, the panel is behind the rubber belts and, therefore, the trajectory is 1mm longer. The second measurement is acquired during the deceleration phase, where the panel is ahead of the rubber belts. Consequently, the patching robot moves 2mm backwards to drive the panel into the desired region. Only after the third absolute position measurement is received at $t_{u,3}$, the controller actually realizes that the panel is positioned correctly and the positioning action is finished.

CHAPTER 7

Conclusions and Outlook

Up to now, patching of wood defects involves a lot of manual labor. It is the most expensive production step in inline timber manufacturing. Yet, it is unavoidable, since wood, as a natural material, always exhibits imperfections and defects that either undermine its structural integrity or appearance. The need for automation is apparent. This work encompasses design, process optimization and control of a patching plant for wooden shuttering panels.

To begin with, the patching robot and, subsequently, the patching plant are designed based on an analysis of the raw material. The state of the art for wood patching in Europe is a semi-automatic patching tool that automates the patching process itself. It eliminates the defect by drilling the respective area and, then, inserting a unisize wooden patch to seal the hole. This tool is integrated in the patching robot. Positioning of the panel underneath the tool, up to now performed by a human, is accomplished by two xy-machine-tables, one to the left and one to the right of the tool, see Figure 2.2. The panels are clamped by the rubber belts of the xy-tables that move them in longitudinal direction. In lateral direction, the entire slide of the xy-tables is moved. For the most part, the panels are clamped and moved by both xy-tables together. This patching robot forms the core of the patching plant. Before the panels are fed to the robot, they go through an optical defect scanner, that describes the border of the defects by polygons. Based on the acquired list of defects, each unique panel is processed in a time optimal way. The algorithms for optimizing the process are executed at an industrial computer. It holds a database of the scanned panels and forwards the data to the patching robot.

In case of big defects, several of the circular, unisize patches might be required. Therefore, in an initial step towards the time optimal processing program, a patch placement algorithm determines the minimum number of patches and their arrangement to fully cover a defect. The proposed algorithm is based on

hexagonally closest packing. It moves the defect polygon relatively to a hexagon tessellation until a minimum number of covering hexagons, i.e. patch circles, is found. Since the tessellation has a periodic pattern, the solution space of this optimization problem is closed. By going through the closed solution space, one is guaranteed to find the global minimum of patches. This algorithm is computationally quite expensive, however, among all optimization steps, this one has the largest contribution to savings of both production time and wood. The wood savings mainly result from the fact that the maximum number of patches per defect is limited due to quality reasons and, therefore, finding the minimum number of patches significantly contributes to waste reduction.

Since the defect scanner sometimes yields faulty data, i.e. several overlapping defects are in one place, a sanity check algorithm preprocesses the data prior to patch placement. An arbitrary number of overlapping defects are merged by recursively calling the algorithm until no overlaps exist anymore. In each call of the algorithm, a pair of overlapping polygons is removed from the defect list. Instead, the union of these two polygons is added.

The patch placement algorithm is applied to every defect of one panel side. This yields a list of patch locations that need to be approached in a time optimal sequence, called robot path. The path planning problem is very similar to the traveling salesman problem. In the context of a path planning algorithm, every patch location is called a node that needs to be visited exactly once, while the panel moves from one side of the patching robot to the other. Because of this dedicated production flow and the elongated, rectangular shape of the panels, it is easy to construct good initial guesses for the time optimal robot path, such as the from-left-to-right path or the nearest-neighbor path. The two presented sophisticated path planning algorithms make use of these initial guesses to speed up computation.

The first one is an Ant Colony Algorithm, which is inspired by the foraging behavior of ants. It is determined by pheromone trails they deposit on their way between food and nest. The Ant Colony Algorithm mimics this behavior by computing paths from a given start to a given end node on a fully connected graph. Each arc of this graph is associated with its traveling cost, i.e. time, and its pheromone level. In every iteration, each ant of a colony individually constructs feasible paths by making random decisions that are biased by the traveling costs and the pheromone levels. Also, it deposits pheromone on the arcs crossed. Conversely, pheromone evaporates. Thus, after some time, quick paths achieve high pheromone levels and the ants converge to the minimum-time path.

The second strategy is called Local Search Receding Horizon Algorithm. The core of this algorithm is Local Search, which is based on 3-optimality. Local Search improves a given initial path, i.e. the from-left-to-right path or the nearest-neighbor path, by performing 3-exchanges until no increase in solution quality can

be achieved anymore. Since the path planning problem considers the production flow, the Receding Horizon Concept is utilized to split the entire problem into a series of smaller subproblems. After solving a subproblem using Local Search, the first part of the solution is appended to the final (minimum-time) path. The rest of the solution serves as a high-quality start solution for the next optimization horizon. Thus, the horizon is moved forward until the entire problem is optimized. Comparing the Ant Colony Algorithm to the Local Search Receding Horizon Algorithm, the latter one is clearly favorable. Its computation time is shorter and the solution quality better because it perfectly exploits the properties of the path planning problem at hand.

After the optimal processing sequence is found, the final step of process optimization is the computation of a two times continuously differentiable, time optimal robot trajectory between two patch locations. Moreover, the trajectory generator shall be able to start from an arbitrary initial velocity and acceleration and it needs to be executable in real-time. A time optimal $\mathcal{C}^2(t)$-trajectory is derived from a step-shaped jerk signal that is integrated three times. The trajectory consists of seven phases during each of which either jerk, acceleration or velocity are saturated. The jerk amplitude, which either takes its maximum or minimum value or zero, and the length of each phase parametrize the trajectory. The computation is done algorithmically. This offline approach, in contrast to online Bang-Bang Trajectory Generators, inherently ensures the trajectories do not exhibit overshoots or oscillations, even in a time-discrete implementation. Still, the offline approach is executable in a fraction of a millisecond. The generation of a time optimal $\mathcal{C}^2(t)$-trajectory that starts at initial velocity and acceleration zero can be accomplished even more efficiently by merely calculating three algebraic equations and distinguishing between three simple special cases.

The next task is the design of the real-time control of the patching robot. It needs to support independent as well as coupled positioning, meaning synchronization of the two xy-tables is required. Further challenges are high friction in the drivetrain of the rubber belts and slip between the belts and the panels. A cascaded control structure turns out to be useful. The PI-velocity controllers and their underlying current controllers are implemented directly at the frequency converters to support high sampling rates. This is necessary to deal with the high friction forces. The high-gain P-position controllers, the trajectory generators as well as the process logic control for the entire plant are implemented at an industrial computer.

During coupled positioning, a master-slave control is employed. In longitudinal direction, XY-Table 1, the master, positions the panel while XY-Table 2, the slave, merely follows the master's velocity. In lateral direction, the xy-tables are controlled independently because, first, the coupling via the panel is quite weak in this direction and, second, because friction is low and thus trajectory tracking for both xy-tables is almost identical.

In longitudinal direction, slip occurs between the panel and the rubber belts. Therefore, tracking of the absolute position of the panel is required. Using computer vision, this is only possible at a very low sampling rate and with a time delay. Every time the panel position is refreshed, an update of the reference trajectory is performed. To this end, the offset between panel and robot position is computed and a new robot trajectory accounting for the detected offset is calculated in real-time.

Based on the real-time control, a robust process logic control for the patching tool, both xy-tables and their neighboring conveyors is developed. Key issues are that both xy-tables are able to handle their own panel without mutual interference, both xy-tables are able to handle one panel together and the safe implementation of the patching action. The process logic control is implemented in C++ as two interacting state-machines, one for each xy-table. To test the proposed process logic control, a simple dynamical model of the patching robot is derived. Using this model, the patching of several panels is simulated and analyzed in detail.

At the end of this work, the effectiveness of the proposed control strategies, in particular of the real-time control, is documented by measurement results of the pilot patching plant. In the first experiment, the challenging friction conditions in the drivetrain of the rubber belts are investigated. Approximately $40 - 60\%$ of the nominal torque of the servo motors driving the belts are required just to set them in motion. At stationary positioning speed, the full nominal torque has to be utilized. In the second experiment, coupled positioning is analyzed. In longitudinal direction, trajectory tracking is very accurate. The velocity signals of the rubber belts exhibit small, high-frequency osciallations around the reference trajectory. This is due to friction. The master-slave concept effectively keeps the difference in the velocities of the xy-tables at a negligible level. More importantly, the applied motor torques of the xy-tables always show the same sign, i.e. they work together in moving the panel. In lateral direction, similar observations are made, although the xy-tables are controlled independently of each other. In the third experiment, measurement results of a real production scenario are discussed. To sum it up, (i) an actuator utilization of up to 200% and the time optimal trajectory generator guarantee minimum positioning times, (ii) a positioning accuracy of one millimeter is achieved and (iii) most production time is consumed by the patching tool. Finally, the data of the third experiment is used to investige the effectiveness of the trajectory updating strategy. Although the absolute panel position is only measured at a very low sampling rate and with a high time delay, this approach ensures accurate panel positioning in the presence of slip.

Going beyond the considered patching plant, the developed algorithms are applicable to a wide range of industrial optimization and control problems.

- Similar problems to patch placement arise, e.g., in coverage of wireless communication or sensor networks.
- Different path planning or process sequence optimization tasks arise in industrial plants and logistics.
- Sophisticated feedforward control strategies and by this trajectory generators are part of most advanced real-time motion control systems.
- Trajectory updating is an efficient way of sensor data fusion applicable to a variety of position tracking problems in mobile robotics and industrial plants.

APPENDIX A

Alternative trajectory generators

In this appendix, two further trajectory generators shall be outlined. First, a so-called Sine-Square Trajectory Generator that is based on the offline Bang-Bang Trajectory Generator presented in Section 3.3 and, second, the online Bang-Bang Trajectory Generator mentioned in Section 3.3.3 are given.

A.1 Sine-Square-Trajectory-Generator

The Sine-Square Trajectory Generator is based on the Bang-Bang Trajectory Generator presented in Section 3.3. In comparison to (3.22), the jerk signal is given by

$$j_t(t) = \begin{cases} 2j_1 \sin\left(\frac{\pi}{2}\frac{t}{T_1}\right)\cos\left(\frac{\pi}{2}\frac{t}{T_1}\right) & 0 \leq t \leq T_1 \\ 2j_3 \sin\left(\frac{\pi}{2}\frac{t-\bar{T}_2}{T_3}\right)\cos\left(\frac{\pi}{2}\frac{t-\bar{T}_2}{T_3}\right) & \bar{T}_2 \leq t \leq \bar{T}_3 \\ 2j_5 \sin\left(\frac{\pi}{2}\frac{t-\bar{T}_4}{T_5}\right)\cos\left(\frac{\pi}{2}\frac{t-\bar{T}_4}{T_5}\right) & \bar{T}_4 \leq t \leq \bar{T}_5 \\ 2j_7 \sin\left(\frac{\pi}{2}\frac{t-\bar{T}_6}{T_7}\right)\cos\left(\frac{\pi}{2}\frac{t-\bar{T}_6}{T_7}\right) & \bar{T}_6 \leq t \leq \bar{T}_7 = \bar{T}_x \\ 0 & \text{otherwise,} \end{cases} \tag{A.1}$$

where

$$\bar{T}_i = \sum_{j=1}^{i} T_j,\ i \in \{2, 3, \ldots, 7\}.$$

Integration of (A.1) yields an acceleration profile, whose transient phases P1, P3, P5 and P7 are of shape $a_t(t) \sim \sin^2(t)$, hence the name. The Sine-Square Trajectory Generator computes $\mathcal{C}^3(t)$-trajectories, see Figure A.1, while the simplicity of the Bang-Bang Trajectory Generator is preserved. This is the main advantage

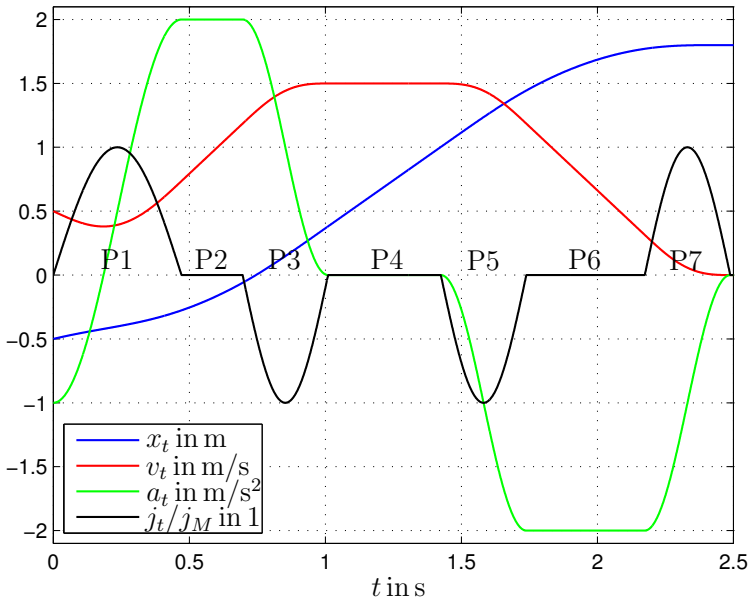

Figure A.1: Seven phases of a Sine-Square Trajectory, maximum jerk $j_M = 10\text{m/s}^3$, maximum acceleration $a_M = 2\text{m/s}^2$, maximum velocity $v_M = 1.5\text{m/s}$. Compare to Figure 3.10.

of this approach; in turn, time optimality is lost. In practice, however, the difference is rather small, since the transient phases of the acceleration profile are typically very short for electrical drives.

Taking into account the trigonometric jerk profile (A.1), all the remaining computations are equivalent. Thus, in the following only a brief overview is presented. A $\mathcal{C}^1$*-trajectory for the acceleration* $\pi_a = \begin{bmatrix} j_1^* & T_1^* \end{bmatrix} = A_t(a_0, a_d)$ follows from

$$j_1^* = \begin{cases} -j_M & a_0 > a_d \\ j_M & a_0 < a_d \\ 0 & a_0 = a_d \end{cases} \tag{A.2}$$

and

$$T_1^* = \begin{cases} 0 & j_1^* = 0 \\ \pi/2\,(a_d - a_0)/j_1^* & j_1^* \neq 0. \end{cases} \tag{A.3}$$

The computation of a $\mathcal{C}^2$*-trajectory for the velocity* begins with the determination of the jerk sequence $\begin{bmatrix} j_1^* & 0 & j_3^* \end{bmatrix}$. This first step is accomplished by computing a $\mathcal{C}^1$-trajectory for the acceleration $\tilde{\pi}_a = \begin{bmatrix} \tilde{j}_1 & \tilde{T}_1 \end{bmatrix} = A_t(a_0, 0)$ and the constant velocity at the end of it

$$\tilde{v}_1 = v_0 + a_0\,\tilde{T}_1 + \tilde{j}_1\,\tilde{T}_1^2/\pi. \tag{A.4}$$

Based on $\tilde{v}_1$, the jerk sequence is

$$[j_1^*, 0, j_3^*] = \begin{cases} \begin{bmatrix} -j_M & 0 & j_M \end{bmatrix} & \tilde{v}_1 > v_d \\ \begin{bmatrix} j_M & 0 & -j_M \end{bmatrix} & \tilde{v}_1 < v_d \\ \begin{bmatrix} \tilde{j}_1 & 0 & 0 \end{bmatrix} & \tilde{v}_1 = v_d. \end{cases} \tag{A.5}$$

Second, the maximum achieved acceleration a^* is determined. To this end, the unlimited maximum achieved acceleration takes the form

$$\hat{a} = \pm \frac{1}{\pi\,(j_1^* - j_3^*)} \sqrt{\pi\,(j_1^* - j_3^*)\, j_3^*\,(4 v_0 j_1^* - 4 v_d j_1^* - \pi\, a_0^2)}. \tag{A.6}$$

Now, a^* considers the bounds on acceleration

$$a^* = \begin{cases} -a_M & \hat{a} < -a_M \\ a_M & \hat{a} > a_M \\ \hat{a} & \text{otherwise.} \end{cases} \tag{A.7}$$

Third, it remains to compute the time parametrization

$$T_1^* = \pi/2\,(a^* - a_0)/j_1^*, \tag{A.8a}$$
$$T_2^* = (v_d - v_3^*)/a^*, \tag{A.8b}$$
$$T_3^* = \pi/2\,(0 - a^*)/j_3^*, \tag{A.8c}$$

where

$$v_3^* = \frac{\pi\,(a^*)^2\, j_3^* + 4 v_0 j_1^* j_3^* - \pi\, a_0^2 j_3^* - \pi\,(a^*)^2\, j_1^*}{4 j_1^* j_3^*} \tag{A.9}$$

is the velocity at the end of Phase P3 with T_1 and T_3 according to (A.8) and $T_2 = 0$.

The computation of the $\mathcal{C}^3$*-trajectory for the position* is performed in an analogous fashion as the computation of the $\mathcal{C}^2$-trajectory for the position of the Bang-Bang Trajectory Generator.

A.1.1 Point-to-point version

Similarly to Section 3.3.2.1, a point-to-point version of the Sine-Square Trajectory Generator can be derived. As before, $x_0 = 0$ and $x_d > 0$ is assumed without loss of generality. Also, the parametriztion of the trajectory is identical, see (3.36), (3.37) and (3.38).

It remains to compute the reduced parameter vector $\pi_{x,p2p} = \begin{bmatrix} T_j & T_a & T_v \end{bmatrix}$ from

$$T_j = \pi a_M / j_M, \tag{A.10a}$$
$$T_a = v_M / a_M - T_j/2, \tag{A.10b}$$
$$T_v = \frac{2x_d/a_M - T_j^2 - 3T_jT_a - 2T_a^2}{T_j + 2T_a}. \tag{A.10c}$$

Looking at (A.10), in general, it might happen that $T_a < 0$ or $T_v < 0$ or both. These three cases have to be distinguished similar to Algorithm 5.

In Case 1, i.e. $T_a < 0$, it is set to $T_a = 0$ and a_M needs to be adjusted

$$a_{M0} = \sqrt{2 j_M v_M / \pi}. \tag{A.11}$$

Then, (A.10a) needs to be evaluated with the newly obtained maximum acceleration a_{M0}.

In Case 2, i.e. $T_v < 0$, it is set to $T_v = 0$ and v_M needs to be adjusted

$$v_{M0} = \frac{-\pi a_M^2 + \sqrt{\pi^2 a_M^4 + 16 j_M^2 a_M x_d}}{4 j_M}. \tag{A.12}$$

Then, (A.10b) needs to be evaluated with the newly obtained maximum velocity v_{M0}.

In Case 3, i.e. $T_a < 0$ and $T_v < 0$, both are set to $T_a = T_v = 0$ and both, a_M and v_M, need to be adjusted

$$a_{M00} = (2x_d)^{1/3} (jM/\pi)^{2/3}, \tag{A.13a}$$
$$v_{M00} = (x_d)^{2/3} (jM/(2\pi))^{1/3}. \tag{A.13b}$$

Then, (A.10a) needs to be evaluated with the newly obtained maximum acceleration a_{M00}.

These extremely simple computations yield the three times continuously differentiable point-to-point trajectory.

A.2 Discretized online Bang-Bang Trajectory Generator

The discretized online Bang-Bang Trajectory Generator that is used for comparison purposes in Section 3.3.3 is presented in detail in [8, 94]. Basically, it consists of a chain of three integrators and a sliding mode controller (SMC) driving them towards their respective reference values, see Figure A.2. In this section, merely the formula necessary for its implementation shall be given. All signals are discretized with k denoting the time step, i.e. $(\cdot)_k = (\cdot)(k\,T_s)$.

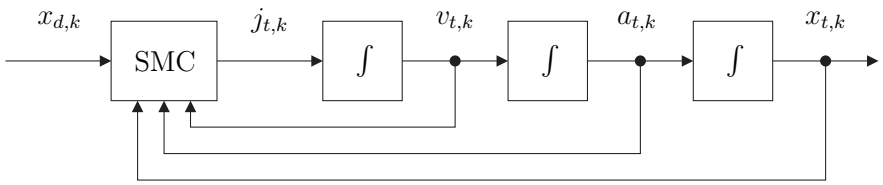

Figure A.2: Schematics of the time-discrete, online Bang-Bang Trajectory Generator, where k denotes the time step, i.e. $(\cdot)_k = (\cdot)(k\,T_s)$.

The integrators are implemented by means of rectangular approximation for the acceleration

$$a_{t,k} = a_{t,k-1} + T_s j_{t,k-1} \tag{A.14a}$$

and by means of trapezoidal approximation for the velocity and the position

$$v_{t,k} = v_{t,k-1} + T_s \frac{a_{t,k} + a_{t,k-1}}{2}, \tag{A.14b}$$

$$x_{t,k} = x_{t,k-1} + T_s \frac{v_{t,k} + v_{t,k-1}}{2}. \tag{A.14c}$$

For the sliding mode controller,

$$\operatorname{sign}(x) = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases} \tag{A.15}$$

and, in particular, $\mathrm{s}_\delta = \operatorname{sign}(\delta)$ is defined. Furthermore, the errors in position, velocity and acceleration are introduced

$$e_{x,k} = \frac{x_{t,k} - x_{d,k}}{j_M}, \quad e_{v,k} = \frac{v_{t,k} - \dot{x}_{d,k}}{j_M}, \quad e_{a,k} = \frac{a_{t,k} - \ddot{x}_{d,k}}{j_M} \tag{A.16}$$

as well as the constraints on the velocity error

$$e_{vm,k} = \frac{v_m - \dot{x}_{d,k}}{j_M}, \quad e_{vM,k} = \frac{v_M - \dot{x}_{d,k}}{j_M} \tag{A.17}$$

and the acceleration error

$$e_{am,k} = \frac{a_m - \ddot{x}_{d,k}}{j_M}, \quad e_{aM,k} = \frac{a_M - \ddot{x}_{d,k}}{j_M}, \tag{A.18}$$

all normalized to the maximum jerk j_M. Note that this Bang-Bang Trajectory Generator is designed for asymmetric bounds on velocity and acceleration, but symmetric bounds on the jerk; $(\cdot)_m$ and $(\cdot)_M$ refer to the minimum and maximum

values, respectively. It is possible to change these values online. Based on this, the sliding mode controller is given by [8, 94],

$$\delta = e_{v,k} + \frac{e_{a,k}\,|e_{a,k}|}{2} \tag{A.19a}$$

$$\sigma = e_{x,k} + e_{v,k}e_{a,k}\mathrm{s}_\delta - \frac{e_{a,k}^3}{6}\left(1 - 3\,|\mathrm{s}_\delta|\right) + \frac{\mathrm{s}_\delta}{4}\sqrt{2\,\left[e_{a,k}^2 + 2e_{v,k}\mathrm{s}_\delta\right]^3} \tag{A.19b}$$

$$\nu^+ = e_{x,k} - \frac{e_{aM,k}\,(e_{a,k}^2 - 2e_{v,k})}{4} - \frac{(e_{a,k}^2 - 2e_{v,k})^2}{8e_{aM,k}} - \frac{e_{a,k}(3e_{v,k} - e_{a,k}^2)}{3} \tag{A.19c}$$

$$\nu^- = e_{x,k} - \frac{e_{am,k}\,(e_{a,k}^2 + 2e_{v,k})}{4} - \frac{(e_{a,k}^2 + 2e_{v,k})^2}{8e_{am,k}} - \frac{e_{a,k}(3e_{v,k} + e_{a,k}^2)}{3} \tag{A.19d}$$

$$\Sigma = \begin{cases} \nu^+ & \text{if } e_{a,k} \le e_{aM,k} \text{ and } e_{v,k} \le \frac{e_{a,k}^2}{2} - e_{aM,k}^2 \\ \nu^- & \text{if } e_{a,k} \ge e_{am,k} \text{ and } e_{v,k} \ge e_{am,k}^2 - \frac{e_{a,k}^2}{2} \\ \sigma & \text{otherwise} \end{cases} \tag{A.19e}$$

$$j_c = -\,j_M\,\mathrm{sign}\left(\Sigma + (1 - |\mathrm{sign}(\Sigma)|)\,[\delta + (1 - |\mathrm{s}_\delta|)\;e_{a,k}]\right) \tag{A.19f}$$

$$j_{t,k} = \max\{j_v(e_{vm,k}),\,\min\{j_c, j_v(e_{vM,k})\}\}, \tag{A.19g}$$

where $j_v(v)$ follows from

$$j_v(v) = \max\{j_a(e_{am,k}),\,\min\{j_{cv}(v), j_a(e_{aM,k})\}\} \tag{A.20a}$$

$$j_{cv}(v) = -\,j_M\,\mathrm{sign}\left(\delta_v(v) + (1 - |\mathrm{sign}(\delta_v(v))|)\;e_{a,k}\right) \tag{A.20b}$$

$$\delta_v(v) = e_{a,k}|e_{a,k}| + 2\,(e_{v,k} - v) \tag{A.20c}$$

$$j_a(a) = -\,j_M\,\mathrm{sign}(e_{a,k} - a). \tag{A.20d}$$

Bibliography

[1] F. Allgöwer, T. A. Badgwell, J. S. Qin, J. B. Rawlings, and S. J. Wright, "Nonlinear predictive control and moving horizon estimation – An introductory overview," in *Advances in Control: Highlights of ECC '99*, P. M. Frank, Ed. London, UK: Springer, 1999, pp. 391–449.

[2] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton, NJ, USA: Princeton University Press, 2006.

[3] X. Bai, Z. Yun, D. Xuan, T. H. Lai, and W. Jia, "Optimal patterns for four-connectivity and full coverage in wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 9, no. 3, pp. 435–448, 2010.

[4] E. Baradit, R. Aedo, and J. Correa, "Knots detection in wood using microwaves," *Wood Science and Technology*, vol. 40, no. 2, pp. 118–123, 2006.

[5] A. Berglund, "Process control and production strategies in the sawmill industry," Ph.D. dissertation, Department of Engineering Sciences and Mathematics, Luleå University of Technology, Skellefteå, SE, 2013.

[6] A. Berglund, O. Broman, A. Grönlund, and M. Fredriksson, "Improved log rotation using information from a computed tomography scanner," *Computers and Electronics in Agriculture*, vol. 90, pp. 152–158, 2013.

[7] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2010.

[8] L. Biagiotti and C. Melchiorri, *Trajectory Planning for Automatic Machines and Robots*. Berlin, GER: Springer, 2008.

[9] L. Biagiotti and R. Zanasi, "Online trajectory planner with constraints on velocity, acceleration and torque," in *Proc. IEEE International Symposium on Industrial Electronics*, Bari, IT, Jul. 2010, pp. 274–279.

[10] B. Bischoff, D. Nguyen-Tuong, F. Streichert, M. Ewert, and A. Knoll, "Fusing vision and odometry for accurate indoor robot localization," in *Proc. International Conference on Control Automation Robotics & Vision*, Guangzhou, CN, Dec. 2012, pp. 347–352.

[11] T. Bock, *Robotics and Automation in Construction.* Rijeka, Croatia: InTech, 2008, ch. 2, pp. 21–42.

[12] R. C. Bolles, "Verification vision for programmable assembly," in *Proc. International Joint Conference on Artificial Intelligence*, Cambridge, MA, USA, Aug. 1977, pp. 569–575.

[13] J. Borenstein and L. Feng, "Measurement and correction of systematic odometry errors in mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 6, pp. 869–880, 1996.

[14] M. Brännström, "Integrated strength grading," Ph.D. dissertation, Division of Wood Technology, Luleå University of Technology, Skellefteå, SE, 2009.

[15] Z. Cai, H. Huang, Y. Qin, and X. Ma, "Ant colony optimization based on adaptive volatility rate of pheromone trail," *International Journal of Communications, Network and System Sciences*, vol. 2, no. 8, pp. 792–796, 2009.

[16] S. Chand, V. N. Hsu, and S. Sethi, "Forecast, solution, and rolling horizons in operations management problems: A classified bibliography," *Manufacturing & Service Operations Management*, vol. 4, no. 1, pp. 25–43, 2002.

[17] F. Chenavier and J. L. Crowley, "Position estimation for a mobile robot using vision and odometry," in *Proc. IEEE International Conference on Robotics and Automation*, vol. 3, Nice, FR, May 1992, pp. 2588–2593.

[18] S.-C. Chu, J. F. Roddick, C.-J. Su, and J.-S. Pan, "Constrained ant colony optimization for data clustering," in *Proc. Pacific Rim International Conference on Artificial Intelligence*, Auckland, NZ, Aug. 2004, pp. 534–543.

[19] W. J. Cook, *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation.* Princeton, NJ, USA: Princeton University Press, 2012.

[20] I. J. Cox, "Blanche - an experiment in guidance and navigation of an autonomous robot vehicle," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 2, pp. 193–204, 1991.

[21] B. Cărbunar, A. Grama, J. Vitek, and O. Cărbunar, "Redundancy and coverage detection in sensor networks," *ACM Transactions on Sensor Networks*, vol. 2, no. 1, pp. 94–128, 2006.

[22] G. K. Das, S. Das, S. C. Nandy, and B. P. Sinha, "Efficient algorithm for placing a given number of base stations to cover a convex region," *Journal of Parallel and Distributed Computing*, vol. 66, no. 11, pp. 1353–1358, 2006.

[23] M. Dorigo and G. Di Caro, "Ant colony optimization: A new meta-heuristic," in *Proc. Congress on Evolutionary Computation*, vol. 2, Washington D.C., USA, Jul. 1999, pp. 1470–1477.

[24] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA, USA: MIT Press, 2004.

[25] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Berlin, GER: Springer, 2003.

[26] M. Fredriksson, "Log sawing position optimization using computed tomography scanning," *Wood Material Science & Engineering*, vol. 9, no. 2, pp. 110–119, 2014.

[27] L. M. Gambardella and M. Dorigo, "An ant colony system hybridized with a new local search for the sequential ordering problem," *INFORMS Journal on Computing*, vol. 12, no. 3, pp. 237–255, 2000.

[28] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*. New York City, NY, USA: Pearson Education, 2009.

[29] M. Gendreau and J.-Y. Potvin, Eds., *Handbook of Metaheuristics*. New York City, NY, USA: Springer, 2010.

[30] O. Gerelli and C. Guarino Lo Bianco, "A discrete-time filter for the on-line generation of trajectories with bounded velocity, acceleration, and jerk," in *Proc. IEEE International Conference on Robotics and Automation*, Anchorage, AK, USA, May 2010, pp. 3989–3994.

[31] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing, 1989.

[32] I.-H. Gu, H. Andersson, and R. Vicen, "Wood defect classification based on image analysis and support vector machines," *Wood Science and Technology*, vol. 44, no. 4, pp. 693–704, 2010.

[33] C. Guarino Lo Bianco, A. Tonielli, and R. Zanasi, "Nonlinear trajectory generator for motion control systems," in *Proc. International IEEE Conference on Industrial Electronics, Control, and Instrumentation*, vol. 1, Taipei, TWN, Aug. 1996, pp. 195–201.

[34] C. Guarino Lo Bianco and F. Ghilardelli, "Third order system for the generation of minimum-time trajectories with asymmetric bounds on velocity, acceleration, and jerk," in *Proc. International Conference on Intelligent Robots and Systems*, Vilamoura, Algarve, PT, Oct. 2012, pp. 137–143.

[35] C. Guarino Lo Bianco and R. Zanasi, "Smooth profile generation for a tile printing machine," *IEEE Transactions on Industrial Electronics*, vol. 50, no. 3, pp. 471–477, 2003.

[36] G. Gutin and A. P. Punnen, *The Traveling Salesman Problem and its Variations*. Philadelphia, NY, USA: Springer, 2002.

[37] W. J. Gutjahr, "A graph-based ant system and its convergence," *Future Generation Computer Systems*, vol. 16, no. 8, pp. 873–888, 2000.

[38] ——, "ACO algorithms with guaranteed convergence to the optimal solution," *Information Processing Letters*, vol. 82, no. 3, pp. 145–153, 2002.

[39] W. J. Gutjahr and M. S. Rauner, "An ACO algorithm for a dynamic regional nurse-scheduling problem in Austria," *Computers & Operations Research*, vol. 34, no. 3, pp. 642–666, 2007.

[40] O. Hagman and E. Åstrand, "Automatic grading: a multisensor approach," in *Proc. IUFRO XX World Congress*, Tampere, FI, Aug. 1995, pp. 195–201.

[41] Z. Hao, H. Huang, X. Zhang, and K. Tu, "A time complexity analysis of aco for linear functions," in *Simulated Evolution and Learning*, T.-D. Wang, X. Li, S.-H. Chen, X. Wang, H. Abbass, H. Iba, G.-L. Chen, and X. Yao, Eds. Berlin, GER: Springer, 2006, pp. 513–520.

[42] D. Helmick, Y. Cheng, D. Clouse, L. Matthies, and S. Roumeliotis, "Path following using visual odometry for a mars rover in high-slip environments," in *Proc. IEEE Aerospace Conference*, vol. 2, Big Sky, MT, USA, Mar. 2004, pp. 772–789.

[43] K. Helsgaun, "An effective implementation of the Lin-Kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.

[44] ——, *An effective implementation of K-opt moves for the Lin-Kernighan TSP heuristic*. Roskilde, DK: Writings on Computer Science Roskilde University, 2006.

[45] ——, "General k-opt submoves for the Lin-Kernighan TSP heuristic," *Mathematical Programming Computation*, vol. 1, no. 2–3, pp. 119–163, 2009.

[46] A. Heppes and H. Melissen, "Covering a rectangle with equal circles," *Periodica Mathematica Hungarica*, vol. 34, no. 1–2, pp. 65–81, 1997.

[47] M. Hofmair, M. Böck, and A. Kugi, "Time-optimal trajectory generation, path planning and control for a wood patching robot," in *Proc. IEEE Multi-Conference on Systems and Control*, Sydney, New South Wales, AUS, Sep. 2015, pp. 459–465.

[48] M. Hofmair, M. Melik-Merkumians, M. Böck, M. Merdan, G. Schitter, and A. Kugi, "Patching process optimization in an agent-controlled timber mill," *Journal of Intelligent Manufacturing*, pp. 1–16, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10845-014-0962-z

[49] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* New York City, NY, USA: Pearson Education, 2001.

[50] H. Hu and D. Gu, "Landmark-based navigation of mobile robots in manufacturing," in *Proc. IEEE International Conference on Emerging Technologies and Factory Automation*, vol. 1, Barcelona, Catalonia, ES, Oct. 1999, pp. 121–128.

[51] X.-B. Hu and W.-H. Chen, "Genetic algorithm based on receding horizon control for arrival sequencing and scheduling," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 5, pp. 633–642, 2005.

[52] ——, "Receding horizon control for aircraft arrival sequencing and scheduling," *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 2, pp. 189–197, 2005.

[53] X.-B. Hu, W.-H. Chen, and E. Di Paolo, "Multiairport capacity management: Genetic algorithm with receding horizon," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 2, pp. 254–263, 2007.

[54] H. Huang, C.-G. Wu, and Z.-F. Hao, "A pheromone-rate-based analysis on the convergence time of ACO algorithm," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 39, no. 4, pp. 910–923, 2009.

[55] E. Johansson, D. Johansson, J. Skog, and M. Fredriksson, "Automated knot detection for high speed computed tomography on Pinus sylvestris L. and Picea abies (L.) Karst. using ellipse fitting in concentric surfaces," *Computers and Electronics in Agriculture*, vol. 96, pp. 238–245, 2013.

[56] D. S. Johnson, "Local optimization and the traveling salesman problem," in *Automata, Languages and Programming*, M. S. Paterson, Ed. Berlin, GER: Springer, 1990, pp. 446–461.

[57] D. S. Johnson and L. A. McGeoch, *Local Search in Combinatorial Optimization*. London, UK: John Wiley and Sons, 1997.

[58] J. T. Karsulovic, L. A. León, and L. Gaete, "The use of linear attenuation coefficients of gamma radiation for detecting knots in pinus radiata," *Forest Products Journal*, vol. 49, no. 2, pp. 73–76, 1999.

[59] ——, "Ultrasonic detection of knots and annual ring orientation in pinus radiata lumber," *Wood and Fiber Science*, vol. 32, no. 3, pp. 278–286, 2000.

[60] S. S. Keerthi and E. G. Gilbert, "Computation of minimum-time feedback control laws for discrete-time systems with state-control constraints," *IEEE Transactions on Automatic Control*, vol. 32, no. 5, pp. 432–435, 1987.

[61] R. Kershner, "The number of circles covering a set," *American Journal of Mathematics*, vol. 61, no. 3, pp. 665–671, 1939.

[62] K. Knierim and O. Sawodny, "Real-time trajectory generation for three-times continuous trajectories," in *Proc. IEEE Conference on Industrial Electronics and Applications*, Singapore, Jul. 2012, pp. 1462–1467.

[63] J. P. La Salle, "Time optimal control systems," in *Proc. of the National Academy of Sciences of the United States of America*, vol. 45, Baltimore, Maryland, USA, Feb. 1959, pp. 573–577.

[64] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[65] S. Lin, "Computer solutions of the traveling salesman problem," *Bell System Technical Journal*, vol. 44, no. 6, pp. 2245–2269, 1965.

[66] F. Longuetaud, F. Mothe, B. Kerautret, A. Krähenbühl, L. Hory, J. Leban, and I. Debled-Rennesson, "Automatic knot detection and measurements from x-ray CT images of wood: A review and validation of an improved algorithm on softwood samples," *Computers and Electronics in Agriculture*, vol. 85, pp. 77–89, 2012.

[67] H. Melissen, "Loosest circle coverings of an equilateral triangle," *Mathematics Magazine*, vol. 70, no. 2, pp. 119–125, 1997.

[68] R. Morselli, "Nonlinear trajectory generators for motion control systems," Ph.D. dissertation, Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, Modena, IT, 2003.

[69] R. Mulligan and H. M. Ammari, "Coverage in wireless sensor networks: A survey," *Network Protocols and Algorithms*, vol. 2, no. 2, pp. 27–53, 2010.

[70] G. E. Murphy, H. D. Marshall, and C. M. Bolding, "Adaptive control of bucking on harvesters to meet order book constraints," *Forest products journal*, vol. 54, no. 12, pp. 114–121, 2004.

[71] U. Nordmark, "Value recovery and production control in the forestry-wood chain using simulation technique," Ph.D. dissertation, Division of Wood Technology, Luleå University of Technology, Skellefteå, SE, 2005.

[72] K. J. Nurmela, "Conjecturally optimal coverings of an equilateral triangle with up to 36 equal circles." *Experimental Mathematics*, vol. 9, no. 2, pp. 241–250, 2000.

[73] K. J. Nurmela and P. R. J. Östergård, "Covering a square with up to 30 equal circles," Helsinki University of Technology, Laboratory for Theoretical Computer Science, Helsinki, FI, 2000.

[74] T. Pahlberg, O. Hagman, and M. Thurley, "Recognition of boards using wood fingerprints based on a fusion of feature detection methods," *Computers and Electronics in Agriculture*, vol. 111, pp. 164–173, 2015.

[75] T. Pahlberg, E. Johansson, O. Hagman, and M. Thurley, "Wood fingerprint recognition using knot neighborhood K-plet descriptors," *Wood Science and Technology*, vol. 49, no. 1, pp. 7–20, 2015.

[76] G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 96–101, 1994.

[77] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach*. Saddle River, NJ, USA: Pearson Education, 2010.

[78] J. Sasiadek, "Sensor fusion," *Annual Reviews in Control*, vol. 26, no. 2, pp. 203–228, 2002.

[79] L. M. Schmitt, "Theory of genetic algorithms," *Theoretical Computer Science*, vol. 259, no. 1–2, pp. 1–61, 2001.

[80] ——, "Theory of genetic algorithms ii: models for genetic operators over the string-tensor representation of populations and convergence to global optima for arbitrary fitness function under scaling," *Theoretical Computer Science*, vol. 310, no. 1–3, pp. 181–231, 2004.

[81] L. M. Schmitt, C. L. Nehaniv, and R. H. Fujii, "Linear analysis of genetic algorithms," *Theoretical Computer Science*, vol. 200, no. 1–2, pp. 101–134, 1998.

[82] P. A. Shenga, P. Bomark, O. Broman, and O. Hagman, "3D phase-shift laser scanning of log shape," *Journal of Bioresources*, vol. 9, no. 4, pp. 7593–7605, 2014.

[83] M. Sipser, *Introduction to the Theory of Computation*. Boston, MA, USA: Cengage Learning, 2012.

[84] T. Stützle and M. Dorigo, "A short convergence proof for a class of ant colony optimization algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 358–365, 2002.

[85] Y. K. Tham, H. Wang, and E. K. Teoh, "Multi-sensor fusion for steerable four-wheeled industrial vehicles," *Control Engineering Practice*, vol. 7, no. 10, pp. 1233–1248, 1999.

[86] C. Todoroki and M. Rönnqvist, "Dynamic control of timber production at a sawmill with log sawing optimization," *Scandinavian Journal of Forest Research*, vol. 17, no. 1, pp. 79–89, 2002.

[87] G. F. Toth, "Thinnest covering of a circle by eight, nine or ten congruent circles," in *Combinatorial and Computational Geometry*, J. E. Goodman, J. Pach, and E. Welzl, Eds. New York City, NY, USA: Cambridge University Press, 2005, pp. 361–376.

[88] T. Tulokas and J. Vuorilehto, "Improvement potential in log rotation," *Baltic Forestry*, vol. 13, no. 2, pp. 221–228, 2007.

[89] J. Vuorilehto and T. Tulokas, "On log rotation precision," *Forest Products Journal*, vol. 57, no. 1–2, pp. 91–96, 2007.

[90] R. Williams, *The Geometrical Foundation of Natural Structure: A Source Book of Design*. New York City, NY, USA: Dover Publications, 1979.

[91] G. Xing, X. Wang, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated coverage and connectivity configuration for energy conservation in sensor networks," *ACM Transactions on Sensor Networks*, vol. 1, no. 1, pp. 36–72, 2005.

[92] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," in *Proc. Annual International Conference on Mobile Computing and Networking*, Rome, IT, Jul. 2001, pp. 70–84.

[93] R. Zanasi, C. Guarino Lo Bianco, and A. Tonielli, "Nonlinear filters for the generation of smooth trajectories," *Automatica*, vol. 36, no. 3, pp. 439–448, 2000.

[94] R. Zanasi and R. Morselli, "Third order trajectory generator satisfying velocity, acceleration and jerk constraints," in *Proc. International Conference on Control Applications*, vol. 2, Glasgow, Scotland, UK, Sep. 2002, pp. 1165–1170.

[95] ——, "Discrete minimum time tracking problem for a chain of three integrators with bounded input," *Automatica*, vol. 39, no. 9, pp. 1643–1649, 2003.

[96] Z.-H. Zhan, J. Zhang, Y. Li, O. Liu, S. Kwok, W. Ip, and O. Kaynak, "An efficient ant colony system based on receding horizon control for the aircraft arrival sequencing and scheduling problem," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 2, pp. 399–412, 2010.

[97] H. Zhang and J. C. Hou, "Maintaining sensing coverage and connectivity," *Ad Hoc & Sensor Wireless Networks*, vol. 1, no. 1, pp. 89–124, 2005.