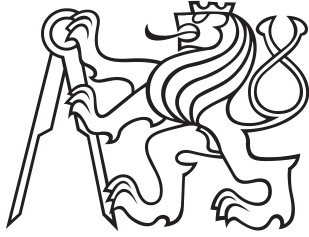


Master Thesis



Czech
Technical
University
in Prague

F2

Faculty of Mechanical Engineering
Department of Mechanics, Biomechanics and Mechatronics

Robotic Painting Cell Design Using Worker's Motion Tracking and Imitation

Bc. Megi Mejdrechová

Supervisor: Ing. Jan Pelikán, Ph.D.

Field of study: Mechatronics

July 2020

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Mejdrechová** Jméno: **Megi** Osobní číslo: **434271**
Fakulta/ústav: **Fakulta strojní**
Zadávací katedra/ústav: **Ústav mechaniky, biomechaniky a mechatroniky**
Studijní program: **Strojní inženýrství**
Studijní obor: **Mechatronika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Návrh robotického lakovacího pracoviště s využitím snímání pohybu a imitace lakýrníka

Název diplomové práce anglicky:

Robotic painting cell design using worker's motion tracking and imitation

Pokyny pro vypracování:

Seznamte se s průmyslovou praxí robotického nanášení nátěrových hmot
Proveďte průzkum dostupných prostředků v oblasti snímání obecného prostorového pohybu tělesa
Navrhněte senzorký systém pro snímání trajektorie lakovacího aplikačního zařízení vedeného lakýrníkem
Proveďte laboratorní experimenty snímání trajektorie s vybranými senzory
Proveďte simulační experimenty snímání předepsané trajektorie pomocí vybraných senzorů pohybu a jejich deklarovaných parametrů
Proveďte simulačně konverzi dat naměřené trajektorie pohybu lakýrníka do dat průmyslového robota pomocí dostupných prostředků
Sestavte koncepční návrh imitačního lakovacího zařízení s průmyslovým robotem

Seznam doporučené literatury:

V. Stejskal, M. Valášek, Kinematics and Dynamics for Machinery, New York, Marcel Dekker 1996

Jméno a pracoviště vedoucí(ho) diplomové práce:

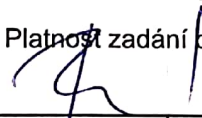
Ing. Jan Pelikán, Ph.D., odbor mechaniky a mechatroniky FS

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

Datum zadání diplomové práce: **30.04.2020**

Termín odevzdání diplomové práce: **07.08.2020**

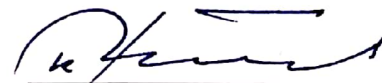
Platnost zadání diplomové práce: _____



Ing. Jan Pelikán, Ph.D.
podpis vedoucí(ho) práce



doc. Ing. Miroslav Španiel, CSc.
podpis vedoucí(ho) ústavu/katedry



prof. Ing. Michael Valášek, DrSc.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

4.5.2020

Datum převzetí zadání

Mejdrechová
Podpis studentky

Declaration

I hereby declare I have written this master thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

In Prague, 31. July 2020

Acknowledgements

I would first like to thank my supervisor Ing. Jan Pelikán, Ph.D. for providing guidance and advice throughout the work on this thesis. The knowledge, experience and ideas he shared with me were always helpful.

Next, I wish to express my gratitude to the Czech Technical University in Prague and especially the Faculty of Mechanical Engineering for giving me opportunity to study and learn and walk the whole way up to my master thesis. I am also grateful for the chance to visit universities abroad that widened my professional skills and ambitions. My gratefulness belongs to the foundation Nadání Josefa, Marie a Zdeňky Hlávkových, that supported my abroad studies.

A special thanks belongs to Ing. Martin Nečas, MSc., Ph.D., who helped me carry out the measurements with laser trackers. Extra thankyou deserves Bc. Ladislav Dvořák for helping me with the construction of slider-crank measurement device and being an unfailing encouragement to me.

My big gratitude goes out to my family, my parents, for their support and love.

Abstract

The topic of this thesis is automation of industrial painting operation and design of a robotic painting cell. A novel concept of robotic trajectory planning based on motion tracking technology is introduced, implemented and tested. This work offers a comparison of possibly used sensors supported by simulations and experiments and verifies the designed concept using a robotic model. Further, it elaborates software for realization of the task of industrial painting automation. The result is presented in context of whole automated robotic painting cell including code for PLC control.

Keywords: Automation, industrial painting, robotic painting, trajectory planning, motion tracking, inertial sensors, PLC control, Moore state machine

Abstrakt

Tématem této práce je automatizace procesu průmyslového lakování a návrh robotického lakovacího pracoviště. V úvodu je představen unikátní koncept plánování robotické trajektorie založený na technologii snímání pohybu, který je dále implementován a otestován. Jsou představeny možné senzory pro snímání pohybu, jejichž porovnání je vytvořeno na základě simulací a experimentů. Ověření navrhovaného konceptu je provedeno pomocí simulačního modelu robota. Dále je navrhnut a vypracován software pro podporu procesu automatizace průmyslového lakování. Výsledné řešení je prezentováno v kontextu celistvého automatizovaného robotického lakovacího pracoviště včetně kódu pro jeho PLC řízení.

Klíčová slova: Automatizace, průmyslové lakování, robotické lakování, plánování trajektorie, snímání pohybu, inerciální senzory, PLC řízení, Moorův stavový stroj

Překlad názvu: Návrh robotického lakovacího pracoviště s využitím snímání pohybu a imitace lakýrníka

Contents

1 Introduction	1
2 Background and literature review	3
3 Concept of the solution	7
4 Thesis targets	11
5 Motion tracking task	13
5.1 Overview of motion tracking techniques	14
5.2 Coordinate frames and transformations	17
5.3 Dynamics	20
5.4 Laser Trackers	21
5.4.1 Method	22
5.4.2 Hardware specification	22
5.4.3 Software specification	24
5.4.4 Post-processing	25
5.4.5 Simulation	29
5.4.6 Measurement scenarios	30
5.4.7 Results	30
5.5 Inertial Sensor	32
5.5.1 Method	33
5.5.2 Hardware specification	35
5.5.3 Software specification	36
5.5.4 Post-processing	40
5.5.5 Simulation	45
5.5.6 Measurement scenarios	46
5.5.7 Results	47
5.6 Visual and inertial sensor fusion	49
5.6.1 Method	50
5.6.2 Hardware specification	51
5.6.3 Software specification	52
5.6.4 Post-processing	53
5.6.5 Measurement scenarios	56
5.6.6 Results	56
5.7 Comparison	57

6 Robot control program synthesis	59
6.1 Transformation	59
6.2 Inverse kinematic problem	60
6.3 Control commands generation	62
6.4 Simulation	63
6.4.1 PyBullet	63
6.4.2 ROS - Descartes	65
6.5 Conclusions from simulations	66
7 Software project design	69
7.1 Introduction	69
7.2 Software specification	70
7.2.1 User roles, agents and actions	71
7.2.2 Hardware and software requirements	73
7.3 Software analysis	74
7.4 Software testing	75
8 Robotic painting cell design and control	85
8.1 Task description	86
8.1.1 The painting operation description	86
8.1.2 Fault diagnosis and isolation	89
8.1.3 Emergency stop	89
8.2 Task analysis	89
8.2.1 Hardware design	90
8.3 State model	91
8.4 State machine algorithm design	91
8.4.1 Moore state machine algorithm design	92
8.5 State machine algorithm implementation	93
9 Discussion and conclusions	111
10 Future work	115
Bibliography	117
A Motion tracking measurements results	125
A.1 Laser Trackers	126
A.2 MMR IMU sensor	146
A.3 T 265 tracking camera	179
A.4 MMR IMU sensor and T265 tracking camera	187

B Robot simulation results	195
B.1 Simulation data	196
B.2 PyBullet	201
B.3 ROS - Descartes	204
C Source code	207
C.1 Laser Trackers post-processing	207
C.2 MMR IMU sensor control and post-processing	207
C.3 T265 tracking camera post-processing	207
C.4 PyBullet KUKA robot simulation	207
C.5 ROS - Descartes ABB robot simulation	207
C.6 PLC control program	207

List-of-Figures

2.1 The world’s first industrial robot by Norwegian company Trallfa. The robot was developed for industrial painting and later bought by ASEA (ABB) [2].	3
5.1 The absolute laser trackers AT901 (front) and AT960 (back) by Leica. Dimensions ca. 250x250x500 millimeters, weight 15 kilograms. [37].	23
5.2 Two Super CatEye reflectors mounted on a solid bar and fixed on a rotary drive during experiment.	24
5.3 The algorithm of dead-reckoning. Orientation and position are determined based on measured angular velocity and external specific force [23].	34
5.4 The MetaMotionR+ sensor by Mbientlab. The PCB and battery are enclosed in a plastic case (now open). Dimensions 27x27x4 millimeters, weight 6 grams. The orientation of the body coordinate frame is shown (following the RGB-xyz convention).	35
5.5 The slider-crank measurement device built to perform experiments with Mbientlab MMR sensor and RealSense T265 tracking camera. The device consists of three revolute joints and one sliding joint realized by four wheels. The MMR sensor can be placed in the paper boxes in three different configurations, the camera can be mounted with two screws at the end of the sliding part in three different configurations as well.	47
5.6 The tracking camera T265 by Intel RealSense. Dimensions 108x24.5x12.5 millimeters, weight 60 grams [60]. The orientation of the body coordinate frame is shown (following the RGB-xyz convention).	51
6.1 Simulation of KUKA LBR iiwa in PyBullet.	64
6.2 Simulation of ABB IRB 2400 in ROS.	65
7.1 Context model of the Robo-Painter software.	72
7.2 Context model of the Robo-Painter software, detail.	76
7.3 Entity relationship diagram of the Robo-Painter software.	77
7.4 Data flow diagram of the Robo-Painter software at level 0, system overview. . .	78
7.5 Data flow diagram of the Robo-Painter software at level 1, software processes. .	79
7.6 Data flow diagram of the Robo-Painter software at level 2, software processes from the user’s (agent’s) perspective.	80
7.7 State transition diagram of the Robo-Painter software at level 0.	81
7.8 State transition diagram of the Robo-Painter software at level 1, from the functional subsection’s perspective (measurement and post-processing).	82

7.9 State transition diagram of the Robo-Painter software at level 1, from the functional subsection's perspective (optimization and robot control).	83
8.1 3D model of the robotic painting cell with labeled elements.	88
8.2 Diagram of the pneumatic circuit used in the robotic painting cell.	91
8.3 State model. Sequential function chart of the super state mode, with identified transitions (c) and states (s).	94
8.4 Sequential function chart of the automatic mode, with identified transitions (c) and states (s).	95
8.5 Sequential function chart of the automatic mode, detail of the failure and emergency stop handling part.	96
8.6 Sequential function chart of the manual mode, with identified transitions (c) and states (s).	97
8.7 Table of transitions for the super state mode 1/2.	98
8.8 Table of transitions for the super state mode 2/2.	98
8.9 Table of states for the super state mode.	99
8.10 Table of transitions for the automatic mode 1/4.	100
8.11 Table of transitions for the automatic mode 2/4.	101
8.12 Table of transitions for the automatic mode 3/4.	102
8.13 Table of transitions for the automatic mode 4/4.	103
8.14 Table of states for the automatic mode.	104
8.15 Table of transitions for the manual mode 1/4.	105
8.16 Table of transitions for the manual mode 2/4.	106
8.17 Table of transitions for the manual mode 3/4.	107
8.18 Table of transitions for the manual mode 4/4.	108
8.19 Table of states for the manual mode.	109
A.1 Laser tracker simulation, dynamic calibration by circular movement, trajectory.	127
A.2 Laser tracker simulation, dynamic calibration by circular movement, position vector.	128
A.3 Laser tracker simulation, dynamic calibration by circular movement with maximum noise, calibration error.	128
A.4 Laser tracker simulation, dynamic calibration by circular movement with minimum noise, calibration error.	129
A.5 Laser tracker simulation, dynamic calibration by random movement, trajectory.	130
A.6 Laser tracker simulation, dynamic calibration by random movement, position vector.	131

A.7 Laser tracker simulation, dynamic calibration by random movement with maximum noise, calibration error.	131
A.8 Laser tracker simulation, dynamic calibration by random movement with minimum noise, calibration error.	132
A.9 Laser tracker measurements, dynamic calibration, trajectory.	133
A.10 Laser tracker measurements, dynamic calibration, calibration error.	134
A.11 Laser tracker measurements, circular movement with common reflector, trajectory. Data based on dynamic calibration are not coincident.	135
A.12 Laser tracker measurements, static calibration, trajectory.	136
A.13 Laser tracker measurements, static calibration, calibration error.	137
A.14 Laser tracker measurements, circular movement with common reflector, trajectory. Data based on static calibration are coincident.	138
A.15 Laser tracker measurements, circular movement with common reflector, orientation. Rotation around x-axis is not defined.	139
A.16 Laser tracker measurements, random movement, orientation. Rotation around x-axis is not defined.	139
A.17 Laser tracker measurements, random movement 1 with common reflector, trajectory.	140
A.18 Laser tracker measurements, random movement 1, velocity measured by tracker 1. The dense parts of the diagram are oscillations due to laser tracker's loss of signal.	141
A.19 Laser tracker measurements, random movement 1, velocity measured by tracker 2. The dense parts of the diagram are oscillations due to laser tracker's loss of signal.	141
A.20 Laser tracker measurements, random movement 1, acceleration measured by tracker 1. The dense parts of the diagram are oscillations due to laser tracker's loss of signal.	142
A.21 Laser tracker measurements, random movement 1, acceleration measured by tracker 2. The dense parts of the diagram are oscillations due to laser tracker's loss of signal.	142
A.22 Laser tracker measurements, random movement 2 with common reflector, trajectory.	143
A.23 Laser tracker measurements, random movement, velocity measured by tracker 1.	144
A.24 Laser tracker measurements, random movement, velocity measured by tracker 2. The dense parts of the diagram are oscillations due to laser tracker's loss of signal.	144
A.25 Laser tracker measurements, random movement, acceleration measured by tracker 1.	145

A.26 Laser tracker measurements, random movement, acceleration measured by tracker 2. The dense parts of the diagram are oscillations due to laser tracker's loss of signal.	145
A.27 MMR and ideal IMU simulations, steady state, sensor readings. Deviation of MMR data from an ideal IMU is visible.	147
A.28 MMR IMU simulation, steady state, histogram of measured acceleration with Gaussian fit. Offset from ideal zero measurements is shown, noise appears to be Gaussian.	148
A.29 MMR and ideal IMU simulations, rotation about z-axis, with added Gaussian noise, sensor readings. Deviation of MMR data from an ideal IMU is visible. ...	149
A.30 MMR IMU simulation, rotation about z-axis, with added Gaussian noise, global acceleration. Undesirable oscillation in x- and y-axis is visible, possibly filtered out.	150
A.31 MMR IMU simulation, rotation about z-axis, with added Gaussian noise, acceleration. Undesirable oscillation in x- and y-axis is visible.	151
A.32 MMR IMU simulation, rotation about z-axis, with added Gaussian noise, position. Original signal diverges, but coincides with reference when filtered. ...	152
A.33 MMR IMU simulation, rotation about z-axis, with added Gaussian noise, traveled trajectory. Without filtering, motion that was not present is generated due to double integration of errors.	153
A.34 MMR and ideal IMU simulations, harmonic motion in x- and z-axis, with added Gaussian noise, sensor readings. Deviation of MMR data from an ideal IMU is visible.	154
A.35 MMR IMU simulation, harmonic motion in x- and z-axis, with added Gaussian noise, global acceleration. Large oscillation in y-axis is visible, possibly filtered out.	155
A.36 MMR IMU simulation, harmonic motion in x- and z-axis, with added Gaussian noise, velocity. Original signal diverges, but coincides with reference when fully filtered.	156
A.37 MMR IMU simulation, harmonic motion in x- and z-axis, with added Gaussian noise, position. Original signal diverges, but coincides with reference when fully filtered.	157
A.38 MMR IMU simulation, harmonic motion in x- and z-axis, with added Gaussian noise, traveled trajectory. With filtering, the trajectory reflects real motion.	158
A.39 MMR IMU experiment, transformation of coordinate frames.	159
A.40 MMR IMU experiment, steady state, calibrated linear acceleration, histogram with Gaussian fit. Offset from ideal zero measurements is shown, noise appears to be Gaussian.	160

A.41 MMR IMU experiment, steady state, not calibrated linear acceleration, histogram with Gaussian fit. Offset from ideal zero measurements is shown, noise appears to be Gaussian.	161
A.42 MMR IMU experiment, steady state, raw acceleration, histogram with Gaussian fit. Offset from ideal zero measurements is shown, noise appears not Gaussian. .	162
A.43 MMR IMU experiment, repeated translation in z-axis, linear acceleration, global acceleration. The undesirable bias increases with time.	163
A.44 MMR IMU experiment, repeated translation in z-axis, linear acceleration, Euler angles. Visible drift and deviations are sufficiently small.	164
A.45 MMR IMU experiment, harmonic motion in z-axis (fast), linear acceleration, Euler angles. Visible drift and deviations are sufficiently small.	165
A.46 MMR IMU experiment, repeated translation in z-axis, calibrated linear acceleration, traveled trajectory. Even when filtered, the trajectory does not reflect real motion.	166
A.47 MMR IMU experiment, repeated translation in z-axis, not calibrated linear acceleration, traveled trajectory. Even when filtered, the trajectory does not reflect real motion.	167
A.48 MMR IMU experiment, repeated translation in z-axis, raw acceleration, traveled trajectory. Even when filtered, the trajectory does not reflect real motion.	168
A.49 MMR IMU experiment, harmonic motion in z-axis (slow), raw acceleration, local acceleration.	169
A.50 MMR IMU experiment, harmonic motion in z-axis (slow), raw acceleration, global acceleration.	170
A.51 MMR IMU experiment, harmonic motion in z-axis (slow), raw acceleration, global acceleration. Undesirable oscillation in x- and y-axis is visible, possibly filtered out.	171
A.52 MMR IMU experiment, harmonic motion in z-axis (slow), raw acceleration, velocity. Original signal diverges, but can be filtered out to show expected values.	172
A.53 MMR IMU experiment, harmonic motion in z-axis (slow), raw acceleration, position. Original signal diverges, but can be filtered out to show expected values.	173
A.54 MMR IMU experiment, harmonic motion in z-axis (fast), raw acceleration, local acceleration.	174
A.55 MMR IMU experiment, harmonic motion in z-axis (fast), raw acceleration, global acceleration.	175
A.56 MMR IMU experiment, harmonic motion in z-axis (fast), raw acceleration, global acceleration. Undesirable oscillation in x- and y-axis is visible, possibly filtered out.	176

A.57 MMR IMU experiment, harmonic motion in z-axis (fast), raw acceleration, velocity. Original signal diverges, but can be filtered out to show expected values.	177
A.58 MMR IMU experiment, harmonic motion in z-axis (fast), raw acceleration, position. Original signal diverges, but can be filtered out to show expected values.	178
A.59 RealSense T265 experiment, transformation of coordinate frames.	180
A.60 RealSense T265 experiment, steady state, histogram with Gaussian fit. Offset is visible but sufficiently small, no noise distribution is shown.	181
A.61 RealSense T265 experiment, steady state, position. The measured data are stable in sufficient range.	182
A.62 RealSense T265 experiment, repeated translation in y-axis, position. With coordinate frames initialization part.	183
A.63 RealSense T265 experiment, harmonic motion in y-axis (fast), position. With coordinate frames initialization part. Data resemble expected values.	184
A.64 RealSense T265 experiment, harmonic motion in y-axis (slow), position. With coordinate frames initialization part. Data resemble expected values.	185
A.65 RealSense T265 experiment, harmonic motion in y-axis (fast), position. With coordinate frames initialization part. Data resemble expected values, but with an observable drift.	186
A.66 MMR IMU and RealSense T265 experiment, repeated translation in y-axis, position. T265 shows expected data, MMR follows them only inaccurately.	188
A.67 MMR IMU and RealSense T265 experiment, translation in z-axis, position. T265 shows expected data, MMR follows them only inaccurately.	189
A.68 MMR IMU and RealSense T265 experiment, harmonic motion in y-axis (fast), position. T265 shows expected data, MMR follows them only inaccurately, with offset.	190
A.69 MMR IMU and RealSense T265 experiment, harmonic motion in y-axis (slow), position. T265 shows expected data, MMR follows them only inaccurately, with offset.	191
A.70 MMR IMU and RealSense T265 experiment, harmonic motion in z-axis (fast), position. T265 shows expected data, MMR follows them only inaccurately, with offset.	192
A.71 MMR IMU and RealSense T265 experiment, harmonic motion in z-axis (slow), position. T265 shows expected data, MMR follows them only inaccurately, with offset.	193

B.1 Trajectory 1 sampled by tracking camera T265 held in hand, random motion imitating painting, 1D.	197
B.2 Trajectory 1 sampled by tracking camera T265 held in hand, random motion imitating painting, 3D.	198
B.3 Trajectory 2 sampled by tracking camera T265 held in hand, random motion imitating painting, 1D.	199
B.4 Trajectory 2 sampled by tracking camera T265 held in hand, random motion imitating painting, 3D.	200
B.5 Error of solutions of inverse kinematics while performing trajectory 1 by KUKA in PyBullet simulation.	202
B.6 Error of solutions of inverse kinematics while performing trajectory 2 by KUKA in PyBullet simulation.	202

List-of-Tables

5.1 Mbientlab MMR motion tracking sensors characteristics [43],[44].	36
5.2 Structure of the csv file with downloaded logged data from the MMR IMU sensor.	39
5.3 Intel RealSense tracking camera T265 sensors characteristics [60].	51
5.4 Structure of the csv file with downloaded logged data from the Intel RealSense tracking camera T265.	53
5.5 Comparison of presented motion tracking techniques with respect to the use case of tracking of a hand-held spray gun.	58
7.1 Data glossary of the Robo-Painter software.	74
8.1 Table of identified errors that can occur during a run of the PLC program.	90
8.2 Details of the emergency stop mode that occurs when the emergency stop switcher is activated during a run of the PLC program.	90
8.3 Structure of the table of transitions.	92
8.4 Structure of the table of states.	92



Chapter 1

Introduction

According to [1], spray painting was the first industrial application employing robots. It happened in a Norwegian company Trallfa producing wheelbarrows and the motivation was unsurprisingly the same as today. Unpleasant working environment in paint workshops drew people away and there was lack of workforce.

Trallfa managed to develop a commercial successful painting robot that was in 1985 bought by ASEA (later ABB). Since then, painting robots take part in manufacturing ordinarily, especially a lot they are used in automotive industry [1].

At the early stages, robots were installed in painting operations as these were considered being easy to automate [1]. Nowadays, a lot of effort is put into research in area of industrial painting automation, because it may be easy to automate in the first place, but it eventually shows to be difficult to do it optimally, efficiently and in top quality.

This work aims to contribute to the topic of robotic painting and industrial painting automation overall. It presents a novel approach to robotic trajectory planning based on human painter's know how and skills. It focuses on automation of batch productions suggesting time, cost and effort efficient solution that has moreover the potential to be optimal and accurate.

The contribution is not only on a theoretical level, neither is its scope reduced to a specific problem without a context, although some parts are elaborated very thoroughly. On the contrary, this work approaches the automation of industrial painting as a complex task and designs a complete solution to it.

Chapter 2

Background and literature review

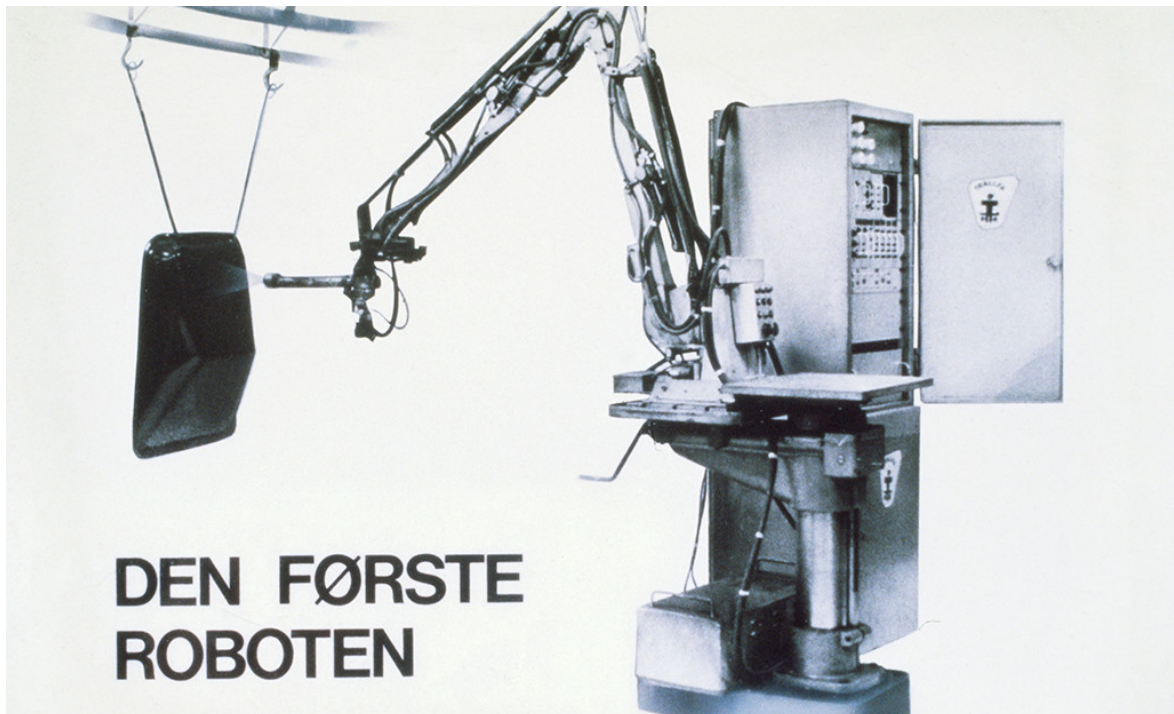


Figure 2.1: The world's first industrial robot by Norwegian company Trallfa. The robot was developed for industrial painting and later bought by ASEA (ABB) [2].

Industrial painting or more generally coating is a technological operation often included in manufacturing processes. Its purpose is to cover a product's surface with a layer of material to provide it with mechanical and chemical protection and/or desired visual characteristic. The coated product can be made of various materials, mostly metal or wood. Industries employing this manufacturing process are mainly automotive, metal and furniture industries. The material of coating layer can be various e.g. metallic alloy, ceramics, polymer and is applied as a solid in form of powder, rods, plates, wires or other arrangements, or as a liquid

or gas [3].

There exist different mechanisms of coating material deposition that can be sorted into bigger groups. Vapor deposition is done in a chamber with high vacuum where the material is transfer to a vapor state and afterwards condensates on the surface of a coated product. Electrodeposition coating takes place in a cell filled with electrolyte and is powered by electric current. The material is transferred towards surface of coated product where it creates a desired layer. Sol-gel coating is done by dipping the coated product into a tub with adhesive sol-gel substance that sticks to it and forms a layer on its surface [3].

Spraying is another group of deposition techniques. It includes thermal spray, plasma spray, cold spray, warm spray and arc wire spray coating. These methods differ in the way how material is prepared before it is sprayed by a high-speed jet through air onto the coated product. Some use heat produced by diverse sources, some rely on impact and solid mechanics of material particles [3].

The other than spraying techniques are performed in chambers or tubs and the role of human worker in these processes is a role of an operator who controls the process and manipulates with the material, but is not directly involved in the coating itself. The spraying techniques on the contrary require a presence of a worker who holds a spraying tool (a spray gun with jet) against the coated product and with his or her movements defines and performs the coating operation. These techniques are due to their nature possible to automate by replacing the human worker with an industrial robot.

This work focuses on spraying techniques, particularly on in industry very often used cold spray or spray painting, where the material is deposited in form of aerosol. There are multiple factors influencing quality of painted layer such as its thickness, uniform coverage, speed an angle of spraying gun and state of the aerosol. It can be concluded that most of the factors depend on the trajectory of the painting tool and therefore the trajectory planning plays a crucial role in the success of a painting operation.

There has been a lot of research done in the topic of painting robot trajectory planning. Many of known methods utilize CAD models (3D data about product's geometry) and mathematical and physical models of painting mechanics. The first representative of these was an Automatic Trajectory Planning System (ATPS) developed by a team of professor Suk-Hwan Suh in 1991 [4]. It did surface approximation using B-splines and path decomposition on short intervals. Other recent examples are [5] or [6].

The trajectory can be also generated as a sequence of predefined trajectories valid for geometric primitives on which the product's surface is decomposed [7]. The trajectory planning can also be solved by optimization techniques, defining the task as a constrained variational problem. This approach minimizes variation in accumulated layer thickness by

evaluating it for each surface point and painting tool position and orientation and applying nonlinear optimization to find desired solution [8].

Alternatively, instead of a CAD model, a set of pictures of the product can be used to extract its geometric features [7]. To allow even more flexible trajectory planning a system utilizing 3D scanner can be used, that scans the part when moving into the painting cell on a conveyor and designs the trajectory right before the painting operation [9].

There are also solutions that employ vision systems into the trajectory planning and painting operation control. The painting process can be accelerated when the products that come into a painting cell on a conveyor do not stop but move continuously. To be able to paint these moving parts, robot has to be flexible and correct its trajectory to correspond with the actual position of painted parts. Such system is implemented for example in [10]. Robots can achieve even higher level of autonomy by performing self-localization techniques using vision systems mounted on the robot's end effector. Such solution exists for mobile robot performing industrial painting of walls in construction sites [11].

An interesting approach is to use virtual reality (VR) system, simulate painting operation and use collected data for robot programming [12].

The research and literature concerning topic of painting robot trajectory planning and optimization is extensive. Each method has specific aspects making it more ideal in some cases and less ideal in others. The topic is definitely not closed yet, there has not been an optimal solution discovered. A fact to keep in mind is, that the industrial practice is still very often hard automation and trial-and-error methods. This thesis presents a novel method of painting robot trajectory planning and aspires to contribute to the topic with own innovative and competent solution.

Chapter 3

Concept of the solution

Chapters 1 and 2 describe the selected problem of industrial painting automation, its background and motivation for solving it. This chapter aims to present a concept of the solution designed and elaborated further in this work. The solution is meant to be innovative and to provide an alternative, efficient way of automation of a painting operation, that can in general be extended to other manufacturing processes.

The main idea of presented solution is to make use of explicit and implicit knowledge and experience of a human worker, who performs the industrial painting professionally. The solution employs a robot and demands it to resemble the human. The goal is to generate a trajectory for the robot and a control program for the robot in such way, that the robotic painting operation imitates the worker's painting operation. To achieve so, motion tracking technique is used to precisely capture the real painting performed by a worker. The trajectory of worker's painting tool is sampled during painting and transformed to the robot trajectory. The control program for the robot is then generated to follow this trajectory, which results in the robot repeating the painting operation according to worker's example.

The intention is that the human paints first several products of a series, finds the optimal way of painting it and then records the ideal painting operation using a motion tracking system. From this record the trajectory of painting tool is extracted and transferred to the robotic control system. The system then generates a control program so, that the painting tool held by the robot as end-effector reproduces the trajectory of the tool held by the worker. The robot then starts painting the rest of the products in the series instead of the human but in the same way the human would do. This approach is motivated by several thoughts:

- A professional industrial painter is highly skilled in the painting activity. It is a person, who understands the problem very well and who has experienced many of its forms.
- Trajectory planning systems try to design an optimal trajectory utilizing knowledge base and mathematical and physical models. A human painter has an extensive knowledge

base and a natural sense for physics, supported by experience.

- Even if the planning system had a very good knowledge base and model, it would be by a big part constructed with the help of human painter. Therefore, it would hardly outperform the human.
- The trajectory planning systems need a reasoning engine to infer an optimal trajectory from presented data. Humans are by default equipped by an excellent reasoning system, usually outperforming the artificial one.
- Data needed to help the system to plan a trajectory, as for example 3D models of painted product's geometry, are often not available. Their collection would be an extra operation.
- It is not desired to automate industrial painting operations because humans would not do it well enough. The reason is, there is a lack of workforce. Therefore, the robot does not have to do it better than humans, doing it just the same is sufficient.
- Using motion tracking of a painter for generation of robot control program does not require any other specialist or robotic expert. It reduces cost and maintains exclusive know-how.

Implementing this approach has several further advantages:

- The painter can review several attempts to paint a product optimally and merge them to create a truly optimal trajectory he or she would otherwise not be able to achieve at once.
- The company that performs industrial painting can keep a few highly skilled painters to record the model trajectories. The profession will not be forgotten, the workers will not lose their jobs, but their working conditions will improve significantly, since they will spend less time with the painting itself.
- The automation of painting of a new product does not require a specialist for robotics and the company with such system is therefore more flexible.

The suggested solution is meant mostly for companies engaged in middle and small series production. The solution would be overly complicated for a job production company. In mass production it would be probably easier for a robotic expert to fine tune the robot control program using available conventional methods, when there is time and space for doing so.

From the technical point of view the designed solution should be implemented in form of a sensory system with a corresponding control software. The sensors include a motion

tracking sensor fixed to a spray gun by a custom-made jig. The role of the motion tracking sensor is to track trajectory of the spray gun. To reconstruct the painting operation not only position but also orientation need to be captured in form of 6DoF data. In fact, information about rotation of the spray gun around its own longitudinal axis is unnecessary, since the beam of paint is parallel to this axis and has usually a circular trace. The way how the beam falls on a painted product is independent on the rotation around longitudinal axis and therefore, even 5DoF measurements are sufficient.

In terms of accuracy, we assume that units of millimeters are satisfactory. The painting tool is not in contact with the product, the optimal distance to it has tolerances. Also, it is a common practise that the product goes through a quality control after it is painted and any imperfections are fixed afterwards. Often there are also spots needed to be painted with a brush not a spray gun, which is also done outside the main painting operation. The presence of human inspection and correction after the robotic painting lowers the requirements on robotic painting accuracy.

Another sensor mounted on the spray gun is there to record when the spray gun was triggered. The corresponding software should also allow for data management and automatically generate control program for a robot based on recorded trajectory.

The solution presented as a concept in this chapter is elaborated further in this work in chapters 5 - 8.



Chapter 4

Thesis targets

The goal of this work is to automate an industrial painting operation by designing a robotic painting cell. The automation process is based on novel approach using worker's motion tracking to imitate the painting performed by human. The ambition of this work is to provide a complex solution for the painting operation automation problem. It approaches it as an engineering task to be solved rather than a theoretical study. It means it aims to be concerned with all main parts and stages of automation and eventually provide a complete and real solution for it.

At the same time, the aspiration is to design a novel approach that would in some aspects outperform existing solutions. This work will also contribute to the indoor motion tracking problem, that is a real engineering challenge in general and therefore deserves attention of academic community.

This chapter defines several thesis targets to meet:

- Define a concept of a novel solution of the industrial painting automation task (Chapter 3).
- Build and test a system for trajectory recording using motion tracking sensors (Chapter 5).
- Provide a transfer of the trajectory to a robotic control program and verify the concept of the solution by simulations (Chapter 6).
- Elaborate a study of a software supporting the defined solution (Chapter 7).
- Design a whole robotic painting cell including control of the manufacturing process to complete the task of automating a painting operation (Chapter 8).



Chapter 5

Motion tracking task

This chapter addresses a key part of the presented solution of industrial painting automation task. Using motion tracking of a human for robot trajectory synthesis is a novelty that has the potential to reduce cost, time and effort, usually needed to automate a manufacturing process. This work builds on this premise and hence a lot of attention was dedicated to mastering motion tracking task. Also, the motion tracking part introduces most of the engineering challenges associated with this work and is therefore the most elaborate of all chapters.

The aim of motion tracking in general is to determine position and orientation in time, or possibly other motion characteristics, of a tracked object using sensory measurements. The tracked object may be various - an aircraft, a tool or a part of human body, the sensors may be of different kinds and combinations too. The output depends on the goal of tracking and the capability of used measurement technique. The level of detail, extensiveness and accuracy can greatly vary [13].

In outdoor environments Global Navigation Satellite System (GNSS) dominates. It is used in almost every navigation application, sometimes improved by fusing with additional measurement techniques [13], [14].

Indoor positioning, navigation or motion tracking is compared to outdoor techniques more varied, still evolving and still not perfectly mastered. This confirms also holding of International Conference on Indoor Positioning and Indoor Navigation [15] organized yearly since 2010. The issues with indoor positioning come from several sources. First, many tracked operations indoor may require very high accuracy (e.g. manufacturing tasks, surgical operations). Next, the indoor environment may be very dynamic (e.g. manufacturing line, a warehouse). It can be as well cramped, blocking the view or reach of sensors. Also, the conditions regarding air and light, magnetic or electric disturbances may be harsh.

This chapter describes a detailed analysis of indoor motion tracking task in context of industrial painting automation and provides a design of its solution, supported by performed

experiments and studied literature. Sections 5.1 - 5.3 provide overview, needed background and equations. Sections 5.4 - 5.7 focus on particular tracking techniques, experiments and conclusions.

■ 5.1 Overview of motion tracking techniques

Motion tracking is a general task and there are many different techniques of solving it. They differ in the physical principal they are based on and can be sorted in following categories [16], [13]:

- Optical techniques
 - Light pulses
 - Camera
- Inertial techniques
- Magnetic techniques
- Acoustic techniques
- Radio techniques
 - Bluetooth
 - Ultra-wideband
 - Global Navigation Satellite System (GNSS)
- Hybrid techniques

In this section, I provide a general overview of available motion tracking technologies. In the rest of this chapter, I focus on three of them that seem relevant to the application of motion tracking of a hand-held spray gun. For each of them, I provide detailed description of working principle and hardware and software characteristics. I chose real products representing the given tracking techniques, got them to work, prepared experiments, tested them, evaluated their performance and compared them. Section 5.4 is concerned with laser trackers, an example of optical techniques utilizing light pulses. Section 5.5 is dedicated to inertial sensors, section 5.6 to a combination of inertial sensor and a camera.

■ Optical techniques

Optical techniques employing light pulses utilize transmitters and receivers, optionally also reflectors. They are based on measurement of time of travel by a light beam between transmitter and receiver, followed by trilateration calculations. Alternatively, they are combined with other techniques such as inertial sensing or angle encoders. The achieved accuracy can be very high [17].

The set-up of such system is relatively space demanding. There are usually multiple receivers distributed over the area of motion. Also, the hardware for pulse generation can not be very small. These systems require a free line of sight between the receivers and transmitters, that further limits the type of environment and motion it is suitable for. Next, it is necessary to calibrate the mutual position of the distributed receivers [14].

An example of the optical techniques utilizing light pulses are laser trackers [17], visible light communication systems [18] or devices for virtual reality [19].

Optical techniques based on camera images use image correlation, visual features detection and tracking and other computer vision algorithms. Also, depth cameras or sensing visual landmarks can be applied. Similar to the light pulses techniques they are space demanding. They usually require multiple cameras distributed around the measurement area and may stop working when the view is blocked. Also, they need ideal light operating conditions [20], [21].

An example of motion tracking systems utilizing cameras are gaming systems such as Microsoft Kinect [22] or systems used by self-guided vehicles [14].

■ Inertial techniques

Implemented in form of inertial measurement units, inertial techniques make use of accelerometer, gyroscope and optionally magnetometer data. Accelerometer measures vector of acceleration, gyroscope measures vector of angular velocities and magnetometer vector of magnetic field intensity. An integration and double integration procedure are necessary to derive orientation and position from measured data. The magnetometer data can be fused together to reference the calculated orientation [23].

The achieved accuracy is low and progressively decrease with length of measurement. The inertial technique, when used alone, serves only for rough estimates of tracked motion. However, it is realized by an inexpensive, compact, small in size and low energy consumption hardware. For that reason, it is used widely in industry, being part of smartphones, wearables, and often complements other motion tracking techniques [23], [14], [24], [25].

■ Magnetic techniques

This technique is built on magnetic field intensity measurements. There exist local anomalies of Earth's magnetic field that can be detected by a magnetometer. Mapping these anomalies to space is used for self-localization in that space. When used indoors, it utilizes the fact that an anomaly of the ambient magnetic field in a building is constant and unique, being caused by static metal parts in the construction and the environment [26].

The sensor can be very small and wireless, which is for an advantage. On the other hand, the magnetic technique requires prior generation of a magnetic map of the space. It is also affected by any moving magnetic parts, changing the otherwise static ambient magnetic field, which makes it difficult to use in an industrial environment. The precision depends on the resolution of magnetic map and is around units of centimeters [26].

The magnetometer can as well be used only as a compass to determine tracked object's heading [14].

■ Acoustic techniques

These techniques work in a similar way as the optical ones using light pulses. Instead of light, ultrasound pulses are used in this case. The system is built up from transmitters and receivers and based on time of flight measurements and trilateration [27]. Alternatively, the system can utilize echoes from nearby objects instead of having receivers [28].

The space set-up is demanding, but the accuracy can be very high, even bellow one millimeter. These systems are less sensitive to environmental conditions than the optical ones, however, suffer from the line of sight limitations equally [27], [28].

■ Radio techniques

Radio techniques rank among active beacons techniques together with some optical and acoustic techniques. The systems of active beacons are composed of transmitters and receivers. One group is mounted around a space at known locations in large number representing the beacons, the other group is in low number mounted on the tracked object. The system measures either time of arrival and performs trilateration, or direction in form of heading angle and performs triangulation [14].

Alternatively, the technique utilizes strength of signal measurements instead of time of flight or angle of arrival. Similar to magnetic techniques, a map of space based on values of signal strength can be built and use for self-localization. The Bluetooth systems are built on ability to measure the strength of signal. An advantage of Bluetooth technology is, that it is commonly available in smartphones and wearables. Applications aiming to provide indoor

navigation for people in shopping malls, museums, offices or other complex buildings are implemented based on this premise [29].

The ultra-wideband technique is based on time of arrival measurement. The signals can travel and be measured on longer distances than by Bluetooth and they can also get through obstacles smoothly. These characteristics make the ultra-wideband technique generally more accurate than the Bluetooth, below one centimeter. On the other hand, it requires building an infrastructure of beacons and receivers. The ultra-wideband systems are implemented for navigation more in warehouses or factories [16].

Global Navigation Satellite Systems (GNSS) are well established for outdoor positioning. The well-known example is the Global Positioning System (GPS). Unfortunately, they can not be used indoors, which is very limiting for industrial applications. The technique is based on measurements of time of arrival of radio signals from satellites to receivers on the ground and performing trilateration. Even the accuracy in tens of centimeters is not enough for industrial usage [14].

The radio techniques are in general not capable of orientation estimates.

■ Hybrid techniques

A wide selection of motion tracking technologies with different characteristics allows for their combination in order to achieve better performance. A typical sensor that is being combined with others is the inertial sensor. There are many applications where sensor fusion is applied on inertial data and ultrasound measurements [30], [31], or inertial and optical measurements [19], [21]. The inertial sensor can also complement systems based on radio techniques to extend their position tracking by orientation data [23].

■ 5.2 Coordinate frames and transformations

A crucial issue of any motion tracking task is the correct understanding of used coordinate systems and reference frames and knowledge of their relationship to the measured data. A reference or coordinate frame is understood as an ordered set of three mutually orthogonal unit-length direction vectors. A coordinate system specifies a mechanism for locating points within a reference frame [32].

Within this work, I refer to several types of coordinate frames [32], [23]:

- **Geographical reference frame - NED** - it is a coordinate frame having the x-axis aligned with geographic north, y-axis aligned with geographic east and z-axis pointing down (towards the Earth's surface, antiparallel to its outward normal). The absolute

orientation is measured w.r.t. this frame. It is sometimes called world frame or local navigational frame, when its origin lies in the center of gravity of navigated body.

- **Geographical reference frame - ENU** - another variant of the inertial reference frame having the x-axis aligned with geographic east, y-axis with geographic north and z-axis pointing up (towards the sky, parallel to Earth's outward normal). The absolute orientation can be measured w.r.t. this frame, but this convention is not used within this work.
- **Body coordinate frame** - it is rigidly attached to a body, moves and rotates with the body. The raw inertia sensor measurements are expressed in this frame. Within this work, it is also called local frame.
- **Initial coordinate frame** - a static coordinate frame. A frame corresponding to the body coordinate frame of the sensor, as it was positioned at the beginning of the measurement. Within this work, it can be also called global frame. The relative orientation is measured w.r.t. this frame.

Some sensors enable the user to express the measured data in the NED (ENU) frame, initial frame, or both.

In our application, a trajectory of a spray gun is tracked during painting to be later repeated by a robot leading ideally to the same result. It is necessary to set a reference coordinate frame in the space and a reference coordinate frame on the product that is being painted, with a constant transformation between these two frames. Ideally, the space reference frame could be attached to a part of the holder of the product, so that the position and orientation of the product w.r.t. to this frame remains constant. The trajectory tracking should be further done so, that the 6DoF data can be expressed w.r.t. this space reference coordinate frame.

The most straight forward solution would be to align the space reference coordinate frame with the coordinate frame in which the measured data are expressed. Either the space reference coordinate frame can be aligned with NED, or the body frame of the sensor can be at the beginning of the measurement aligned with the chosen space coordinate frame. All the measurement can be later transformed into the NED or initial body frame respectively.

I chose the second option, being easier to achieve in practice and possible to achieve with all types of used sensors. It would be also possible to choose the reference frame for measurement so, that it is not aligned with the reference space frame, but that a known constant transformation between them exists. I did not choose to do so, finding it unnecessarily complicated to transform all the data once more, but I admit that it can be the

most comfortable case in the real measurement set-up in the painting workplace. Knowing the orientation of the initial coordinate frame of the sensor, we can align it with the space reference coordinate frame at the beginning of the measurement easily. In practice, it would be convenient to manufacture a jig so that placing the spray gun with the attached sensor into that jig would ensure the frames alignment.

As a result of accurate measurement preparation, the 6DoF trajectory data are expressed w.r.t. to the known reference space coordinate frame. The position and orientation of the robot coordinate frame is also known and the constant transformation from this robot frame to the reference space frame can be determined. Finally, the 6DoF data are transformed to the robot coordinate frame by multiplying them by the robot-space frame transformation matrix. Based on that transformed data, the robot control program can be generated.

In the rest of this section, I present a set of useful formulas needed for coordinate frames transformation calculations.

A matrix of directional cosines \mathbf{S} [33] is expressed using quaternions as follows:

$$\mathbf{S} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_0q_2 + q_1q_3) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_0q_1 + q_2q_3) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}, \quad (5.1)$$

where q_0, q_1, q_2, q_3 are components of a unit quaternion q such that $q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$ [34].

The matrix of directional cosines \mathbf{S} expressed using Euler angles is of a form:

$$\mathbf{S} = \begin{bmatrix} \cos \psi \cos \theta & -\cos \psi \sin \theta & \sin \psi \\ \cos \phi \sin \theta + \sin \phi \sin \psi \cos \theta & \cos \phi \cos \theta - \sin \phi \sin \psi \sin \theta & -\sin \phi \cos \psi \\ \sin \phi \sin \theta - \cos \phi \sin \psi \cos \theta & \sin \phi \cos \theta + \cos \phi \sin \psi \sin \theta & \cos \phi \cos \psi \end{bmatrix}, \quad (5.2)$$

assuming the rotation in XYZ sequence about ϕ, ψ, θ angles [34].

From (5.1) and (5.2) it can be derived that:

$$\begin{aligned} \phi &= \arctan \frac{-s_{23}}{s_{33}}, \\ \psi &= \arctan \frac{s_{13}}{\sqrt{1 - s_{13}^2}}, \\ \theta &= \arctan \frac{-s_{12}}{s_{11}}, \end{aligned} \quad (5.3)$$

and

$$\begin{aligned}
q0 &= -\sin \frac{\phi}{2} \sin \frac{\psi}{2} \sin \frac{\theta}{2} + \cos \frac{\phi}{2} \cos \frac{\psi}{2} \cos \frac{\theta}{2}, \\
q1 &= \sin \frac{\phi}{2} \cos \frac{\psi}{2} \cos \frac{\theta}{2} + \cos \frac{\phi}{2} \sin \frac{\psi}{2} \sin \frac{\theta}{2}, \\
q2 &= -\sin \frac{\phi}{2} \cos \frac{\psi}{2} \sin \frac{\theta}{2} + \cos \frac{\phi}{2} \sin \frac{\psi}{2} \cos \frac{\theta}{2}, \\
q3 &= \sin \frac{\phi}{2} \sin \frac{\psi}{2} \cos \frac{\theta}{2} + \cos \frac{\phi}{2} \cos \frac{\psi}{2} \sin \frac{\theta}{2},
\end{aligned} \tag{5.4}$$

where s_{ij} stands for an i -th row, j -th column element of matrix \mathbf{S} [34].

Equations (5.1), (5.2), (5.3) and (5.4) are implemented in Matlab in the Robotics System Toolbox as *quat2rotm*, *eul2rotm*, *rotm2eul* and *rotm2quat* respectively.

5.3 Dynamics

To derive equations later needed to accomplish the motion tracking task, I followed the description of particle dynamics presented for example in [35]. First, the position vector is expressed:

$$\mathbf{r}_g(t) = x(t)\mathbf{i} + y(t)\mathbf{j} + z(t)\mathbf{k}, \tag{5.5}$$

where the \mathbf{i} , \mathbf{j} , \mathbf{k} are unit directional vectors defining the global coordinate frame. Since the global frame is fixed, these components are not functions of time. Further, the velocity vector can be expressed as:

$$\begin{aligned}
\mathbf{v}_g(t) &= \frac{d\mathbf{r}}{dt} = v_x(t)\mathbf{i} + v_y(t)\mathbf{j} + v_z(t)\mathbf{k}, \\
v_x(t) &= \frac{dx(t)}{dt}, \\
v_y(t) &= \frac{dy(t)}{dt}, \\
v_z(t) &= \frac{dz(t)}{dt},
\end{aligned} \tag{5.6}$$

followed by the acceleration vector:

$$\begin{aligned}
\mathbf{a}_g(t) &= \frac{d\mathbf{v}}{dt} = a_x(t)\mathbf{i} + a_y(t)\mathbf{j} + a_z(t)\mathbf{k}, \\
a_x(t) &= \frac{dv_x(t)}{dt} = \frac{d^2x(t)}{dt^2}, \\
a_y(t) &= \frac{dv_y(t)}{dt} = \frac{d^2y(t)}{dt^2}, \\
a_z(t) &= \frac{dv_z(t)}{dt} = \frac{d^2z(t)}{dt^2}.
\end{aligned} \tag{5.7}$$

The derivatives in relationships (5.6) and (5.7) can be approximated by finite differences resulting in:

$$\begin{aligned}v_x^{k+1} &= a_x^k(t^{k+1} - t^k) + v_x^k, \\v_y^{k+1} &= a_y^k(t^{k+1} - t^k) + v_y^k, \\v_z^{k+1} &= a_z^k(t^{k+1} - t^k) + v_z^k\end{aligned}\tag{5.8}$$

and finally leading to equations for position calculation:

$$\begin{aligned}x^{k+1} &= v_x^k(t^{k+1} - t^k) + x^k, \\y^{k+1} &= v_y^k(t^{k+1} - t^k) + y^k, \\z^{k+1} &= v_z^k(t^{k+1} - t^k) + z^k.\end{aligned}\tag{5.9}$$

The equations (5.8) and (5.9) can be reformulated for velocity calculation from position vector:

$$\begin{aligned}v_x^k &= \frac{x^{k+1} - x^k}{(t^{k+1} - t^k)}, \\v_y^k &= \frac{y^{k+1} - y^k}{(t^{k+1} - t^k)}, \\v_z^k &= \frac{z^{k+1} - z^k}{(t^{k+1} - t^k)}.\end{aligned}\tag{5.10}$$

and acceleration calculation from velocity vector:

$$\begin{aligned}a_x^k &= \frac{v_x^{k+1} - v_x^k}{(t^{k+1} - t^k)}, \\a_y^k &= \frac{v_y^{k+1} - v_y^k}{(t^{k+1} - t^k)}, \\a_z^k &= \frac{v_z^{k+1} - v_z^k}{(t^{k+1} - t^k)}.\end{aligned}\tag{5.11}$$

The superscript k indicates the time step.

5.4 Laser Trackers

In order to design an appropriate method of tracking the trajectory of a painting tool, we conducted several benchmark measurements. For these we chose a device that is capable of highly accurate measurements without a need to consider if it is suitable for the application in a painting workplace or not. The measurements were conducted in laboratory and should simulate measurements of real painting operation in a painting workplace. The results are meant to give us a notion of the character of measured movements, data and measurement conditions.

We decided to use absolute laser trackers. Laser tracker is a 3D position tracking device widely used in industrial applications thanks to its high accuracy. The output of the laser tracker is the position of a tracked point, expressed relatively to the tracker's local coordinate system. The declared accuracy of the position data is within 15 to less than 500 micrometers, depending on the distance of the tracked point and used measurement accessories [36], which is beyond the needs of our application and therefore considered as optimal.

■ 5.4.1 Method

A laser tracker is composed of a stand with a laser source and a head that has two degrees of freedom. The head points the laser beam towards a reflector, optical device, that reflects the beam back to the head. The head can rotate around the vertical and the horizontal axis so that it is able to follow the reflector in space. The position of the reflector at each time is described by horizontal and vertical angles of the head and the distance traveled by the laser from the head to the reflector and back. The distance is determined from the interference of the reflected laser beam when it reaches the interferometer in the laser head [17].

The described principal works under an assumption that the line of sight between the laser head and the reflector is clear and the beam can reach the reflector. This can be limiting when measuring movements include wide range rotations or are carried out in a dynamic environment with moving objects or people, potentially blocking the line of sight.

We focus on motion tracking of a painting tool, which is represented as a body in space and has six degrees of freedom. Since that, the three coordinates of a position vector obtained from one laser tracker are not enough to locate the painting tool in space. To handle this limitation, we used two laser trackers and we were tracking two different points. Measuring position of two points on a body allows for computation of the orientation of the body. The two measured points were fixed on a solid bar representing the painting tool. Although we measured six coordinates, only five of them are independent. This is a conclusion of a constrain put on the distance between the two points, which has to remain constant. Using this set-up, it becomes possible to reconstruct the position and orientation of the tool except for the rotation along its own axis. We assume this information to be unnecessary as explained earlier in chapter 3.

■ 5.4.2 Hardware specification

We used two absolute laser trackers Leica AT960 [37] and Leica AT901 [38] (Fig. 5.1) and two RRR 1.5" corner-cube reflectors with acceptance angle $\pm 30^\circ$ [36]. First, we intended

to use the Super CatEye reflectors (Fig. 5.2). They allow to track a target even when it is moving in wide physical area and rotates a lot, because it has an ultra-wide laser acceptance angle of $\pm 75^\circ$ [36]. After several tests, we concluded that the accuracy drops compared to the classical corner-cube reflectors and there still occur situations when the line of sight is interrupted, hence it is not beneficial. Both laser trackers can cover a measurement volume of up to 160 meters in diameter.



Figure 5.1: The absolute laser trackers AT901 (front) and AT960 (back) by Leica. Dimensions ca. 250x250x500 millimeters, weight 15 kilograms. [37].

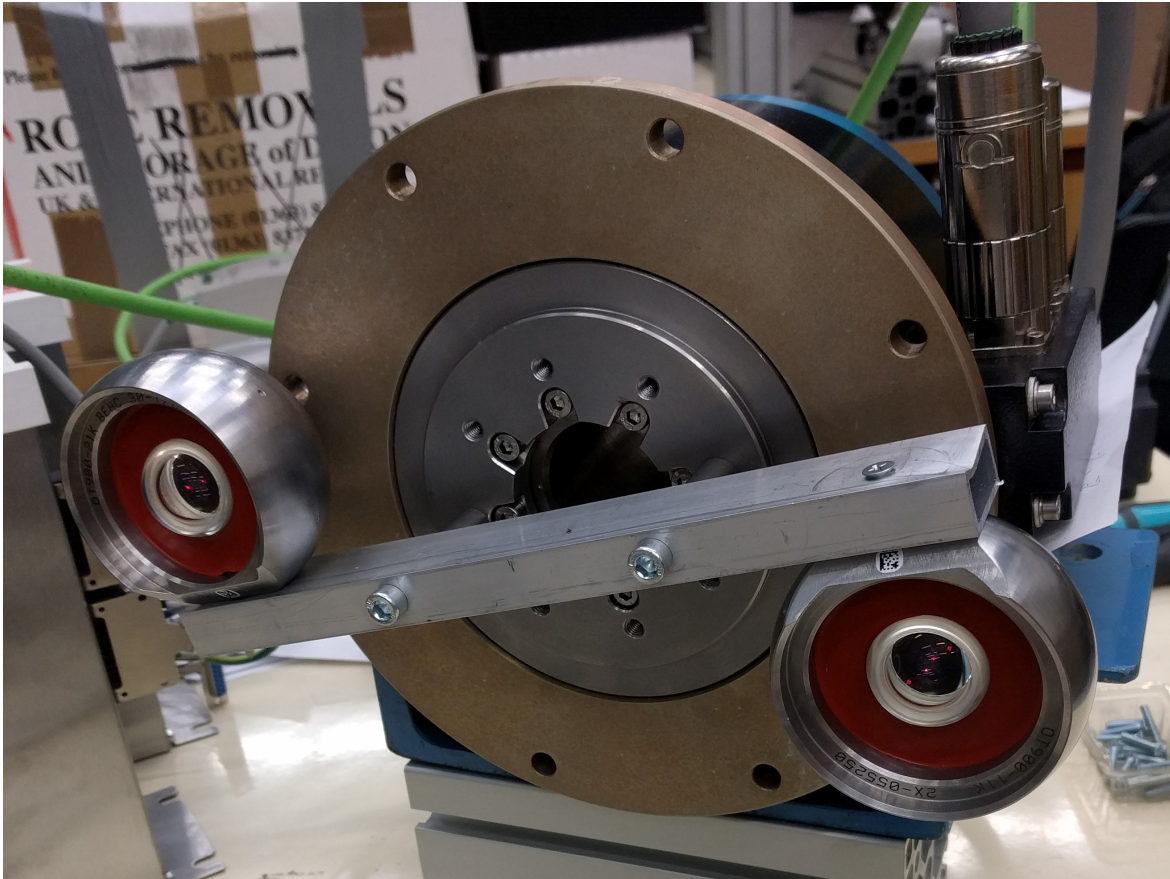


Figure 5.2: Two Super CatEye reflectors mounted on a solid bar and fixed on a rotary drive during experiment.

■ 5.4.3 Software specification

The AT960 controller runs the system software and provides interface to the tracker. It offers a Real-Time Feature Pack that provides measurement data in real-time over EtherCAT at frequency up to 1000 Hz and handles the synchronization of the two laser trackers when used together. The Tracker Pilot software by Leica enables user to quickly check the AT system, initialize it and set some necessary parameters connected to the AT hardware and software [37].

For data acquisition, analysis and management, including logging and export, a third-party metrology software has to be used [37]. In our laboratory, we use TwinCAT 3 software by Beckhoff. The software allows to configure the measurement and to export the measured data into csv files.

■ 5.4.4 Post-processing

To get the desired trajectory data some post-processing must be applied on the logged measurements. Firstly, the data in form of vertical angle [rad], horizontal angle [rad] and distance [m] w.r.t. the tracker coordinate frame (world) need to be used to calculate position vector $\mathbf{r}_w = [x_w \ y_w \ z_w]^T$ defining the position of tracked target in the world coordinate frame. After this stage, each laser tracker has its own measured data expressed w.r.t. its own world frame.

Next, it is necessary to transform the position vectors \mathbf{r}_w to one common coordinate frame in order to get a uniform description of the motion of tracked body. Also, this enables further computation with the position vectors such as the orientation determination. In order to transform both vectors to one common frame, it is necessary to know the mutual position and orientation of the two world frames of individual trackers. It is achieved by a calibration procedure that leads to determination of a transformation matrix \mathbf{T} [33] describing the transformation from one to another.

When the position vectors \mathbf{r}_w are transformed to a common frame, their difference is used to calculate orientation of the tracked body. The velocity and acceleration along the trajectory can be calculated as well.

All data post-processing was implemented in Matlab R2019b and can be found in script *laser_data_post.m*, that also includes visualizations and animations. A script called *laser_velocity_acceleration.m* contains code for velocity and acceleration calculations and visualizations. The source code is to be found in the appendix (C.1). It also includes a template script *cutting_the_data_2_march.m* for basic raw data management. That contains functions for handling outliers, removing insignificant data parts and basic visualization. There is no need for data synchronization. Data signals from both laser trackers are synchronized by the AT960 controller running the Real-Time Feature Pack.

■ Calculation of position vectors

The laser trackers determine the position of a tracked point by providing the vertical and the horizontal angle of the tracker's head orientation and the distance between the head and the tracked point. It is necessary to determine how the angles are defined and in what range their values lie. The vertical angle, labeled θ , is an elevation angle of the tracker's head, measured from the positive z-axis of the tracker coordinate frame (pointing towards the sky) towards the xy-plane. The $\theta \in \langle -\pi; \pi \rangle$ is equal zero when the head points upwards.

The horizontal angle φ is an azimuth angle, $\varphi \in \langle -\pi; \pi \rangle$. There was a discrepancy between the two laser trackers. AT960 measures φ from the positive x-axis, where $\varphi = 0$,

increasing towards the positive y-axis and decreasing towards the negative y-axis. When aligned with negative x-axis, $\varphi = -\pi$ or π . AT901 measures φ from the positive y-axis, where $\varphi = 0$, increasing towards the positive x-axis and decreasing towards the negative x-axis. $\varphi = -\pi$ or π when aligned with negative y-axis. To unify the angles declaration, I decided to use the AT960 notation and recalculate the AT901 horizontal angle as $\varphi = \frac{\pi}{2} - \tilde{\varphi}$, where the $\tilde{\varphi}$ is the originally measured one.

Having angles θ, φ and distance d defined as described, the position of tracked point is determined by an equivalent of spherical coordinates. To transform it to the Cartesian coordinates equation (5.12) is utilized. To distinguish between the two laser trackers, I use subscripts, 1 stands for the AT901 and 2 for the AT960. \mathbf{r}_1 describes the position of the reflector tracked by AT901 expressed in the AT901 coordinate frame. \mathbf{r}_2 describes the position of the other reflector tracked by AT960 expressed in the AT960 coordinate frame.

$$\begin{aligned}\mathbf{r} &= [x \ y \ z]^T, \\ x &= d \sin \theta \cos \varphi, \\ y &= d \sin \theta \sin \varphi, \\ z &= d \cos \theta.\end{aligned}\tag{5.12}$$

■ Calibration

To be able to relate the measurements of both laser trackers and make further computations, it is necessary to know the mutual position and orientation of the two tracker's world frames. An ideal way of doing that is to find a transformation matrix \mathbf{T}_{12} from the world frame of AT901 to the world frame of AT960 that satisfies following equation:

$${}^1\mathbf{r}_2 = \mathbf{T}_{12} {}^2\mathbf{r}_2.\tag{5.13}$$

The superscript denotes in which global coordinate frame is the vector \mathbf{r} expressed.

The matrix \mathbf{T}_{12} is determined using a calibration procedure. During the calibration, only one reflector is used and both laser trackers track it together. If the matrix \mathbf{T}_{12} was known and the vector ${}^2\mathbf{r}_2$ was transformed to the coordinate frame 1 using (5.13), the identity ${}^1\mathbf{r}_1 = {}^1\mathbf{r}_2$ would hold. Based on this conclusion a regression formula can be derived as follows:

$${}^1\tilde{\mathbf{r}}_1 = \mathbf{T}_{12} {}^2\tilde{\mathbf{r}}_2,\tag{5.14}$$

where ${}^1\tilde{\mathbf{r}}_1$ and ${}^2\tilde{\mathbf{r}}_2$ are matrices of measured position vectors \mathbf{r} of dimensions $4 \times N$. The fourth row of ones is added for consistency with the matrix \mathbf{T} dimensions 4×4 . N stands for the number of measurements.

From (5.14) applying pseudoinversion final expression for \mathbf{T}_{12} is derived:

$$\mathbf{T}_{12} = (\mathbf{1}\tilde{\mathbf{r}}_1 \ \mathbf{2}\tilde{\mathbf{r}}_2^T)(\mathbf{2}\tilde{\mathbf{r}}_2 \ \mathbf{2}\tilde{\mathbf{r}}_2^T)^{-1}. \quad (5.15)$$

Alternatively, the elements of matrix \mathbf{T}_{12} can be determined using an optimization method. The objective vector function \mathbf{f} can be defined as:

$$\mathbf{f} = \mathbf{T}_{12}\mathbf{2}\tilde{\mathbf{r}}_2 - \mathbf{1}\tilde{\mathbf{r}}_1, \quad (5.16)$$

where the elements of matrix \mathbf{T}_{12} are the optimized parameters.

Minimizing \mathbf{f} using least square method leads eventually to the same solution as the regression in (5.15). This conclusion was confirmed by the experiments where both approaches were tested. The optimization was implemented using a Matlab function *lsqnonlin*.

It would be possible to further adjust the form of equations (5.15) and (5.16) to utilize the general knowledge of matrix \mathbf{T} characteristics. The matrix dimensions are 4x4 but the elements in the last row are known constants [0 0 0 1]. In fact, the matrix has only six independent variables. The matrix of directional cosines \mathbf{S} that is a submatrix of the matrix \mathbf{T} (5.17) is orthonormal [33] and therefore represents only three independent variables, the three angles of rotations. The other three are elements of a displacement vector \mathbf{r} .

$$\begin{aligned} \mathbf{T} &= \begin{bmatrix} \mathbf{S} & \mathbf{r} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{T}_{\varphi_x}(\varphi_x)\mathbf{T}_{\varphi_y}(\varphi_y)\mathbf{T}_{\varphi_z}(\varphi_z)\mathbf{T}_d(x_d, y_d, z_d) \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \varphi_x & -\sin \varphi_x & 0 \\ 0 & \sin \varphi_x & \cos \varphi_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \varphi_y & 0 & \sin \varphi_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \varphi_y & 0 & \cos \varphi_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \varphi_z & -\sin \varphi_z & 0 & 0 \\ \sin \varphi_z & \cos \varphi_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & x_d \\ 0 & 1 & 0 & y_d \\ 0 & 0 & 1 & z_d \\ 0 & 0 & 0 & 1 \end{bmatrix}. \end{aligned} \quad (5.17)$$

Choosing the elements of matrix \mathbf{T}_{12} to be the unknown parameters directly is however computationally easier than choosing the rotation angles and displacements. Proceeding with the second option would require to differentiate the expressions (5.15) and (5.16) with respect to the angles φ_x , φ_y , φ_z and displacements x_d , y_d , z_d and the resultant relationships would be nonlinear.

The number of unknown parameters equal 12 or even 16, when all elements of \mathbf{T}_{12} are being identified, is still very low compared to the amount of easily collected measurement data. Therefore, the calibration will converge and there is no need of using the alternative demanding approach with six parameters. This conclusion was confirmed even experimentally.

■ Transformation and calculation of orientation

The tracking device is static and its coordinate frame can be assumed as the world coordinate frame. The reflector is the moving part, attached to the tracked object. All measured data (position of the reflector) are expressed w.r.t. the world frame, which is convenient, since the world frame is determined from the position and orientation of the laser tracker and its location can be therefore easily reconstructed. For that reason, I set the world frame to be the global coordinate frame and the coordinate frame of a reflector to be the local frame.

It is important to notice, that measurement done by each laser tracker has its own global and local reference frame. Also, the measurement determines only the position of the center of the respective local frame, not its orientation. Aiming to determine 6DoF trajectory data of a body represented by the solid bar on which the reflectors are mounted, it is necessary to choose one global and one local frame and relate the 6DoF data to them. I chose the resultant global frame to be the global frame of laser tracker 1 (AT901) and the local frame to have its center in the local frame of reflector 1.

The position of the local frame is equal to the position vector \mathbf{r}_1 and does not need to be transformed anyhow. To determine the orientation of the local frame, the position vector \mathbf{r}_2 first needs to be transformed to the global frame using the equation (5.13).

As already mentioned in section 5.4.1, it is not possible to determine all three orientation angles. Therefore, I set the x-axis of the body local frame to be parallel to the connecting line between reflector 1 and reflector 2. This x-axis represents the longitudinal axis of the spray gun and rotation about this axis is thus not interesting for our objective.

The vector of connecting line between reflectors can be noted as \mathbf{R} and calculated as follows:

$$\begin{aligned}\mathbf{R} &= [R_x \ R_y \ R_z]^T \\ \mathbf{R} &= {}^1\mathbf{r}_1 - {}^1\mathbf{r}_2.\end{aligned}\tag{5.18}$$

Then we can project the vector \mathbf{R} into the xy-plane of the global frame and calculate the angle φ_R between the x-axis and the projected vector \mathbf{R}^{xy} :

$$\varphi_R = \arctan \frac{R_y}{R_x}.\tag{5.19}$$

Next, we determine the angle θ_R between the projected vector \mathbf{R}^{xy} in the xy-plane and the original vector \mathbf{R} :

$$\theta_R = -\pi/2 + \arctan \frac{\|\mathbf{R}^{\text{xy}}\|}{R_z}.\tag{5.20}$$

The two angles φ_R and θ_R determine the orientation of the local frame. To express it in form of Euler angles the most straight forward way is to use the ZYX sequence. The first

rotation is about z-axis by angle φ_R , the second rotation is about y-axis by angle θ_R , the third rotation cannot be determined and can thus be omitted.

Another option is to compose the matrix of directional cosines \mathbf{S} :

$$\begin{aligned} \mathbf{S} &= \mathbf{S}_z(\varphi_R)\mathbf{S}_y(\theta_R) \\ \mathbf{S} &= \begin{bmatrix} \cos \varphi_R & -\sin \varphi_R & 0 \\ \sin \varphi_R & \cos \varphi_R & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta_R & 0 & \sin \theta_R \\ 0 & 1 & 0 \\ -\sin \theta_R & 0 & \cos \theta_R \end{bmatrix} = \\ &= \begin{bmatrix} \cos \theta_R \cos \varphi_R & -\sin \varphi_R & \cos \varphi_R \sin \theta_R \\ \cos \theta_R \sin \varphi_R & \cos \varphi_R & \sin \theta_R \sin \varphi_R \\ -\sin \theta_R & 0 & \cos \theta_R \end{bmatrix}. \end{aligned} \quad (5.21)$$

From matrix \mathbf{S} (5.21) Euler angles and quaternions can be determined using (5.3) and (5.4).

■ Calculation of velocity and acceleration

The calculation of velocity and acceleration is done simply by differentiating values of the position vector using (5.10) and (5.11) from section 5.3. The used superscript indicates the time step, or rather the measurement index, going from $k = 1$ to $k = K$, where K is the number of measurements. The expression $(t^{k+1} - t^k)$ stands for the length of the k -th time step, that is calculated from the vector of time stamps corresponding to the \mathbf{r} data.

Velocity and acceleration were calculated from ${}^1\mathbf{r}_1$, ${}^2\mathbf{r}_2$ and ${}^1\mathbf{r}_2$ data, resulting in ${}^1\mathbf{v}_1$, ${}^2\mathbf{v}_2$, ${}^1\mathbf{v}_2$ and ${}^1\mathbf{a}_1$, ${}^2\mathbf{a}_2$, ${}^1\mathbf{a}_2$.

■ 5.4.5 Simulation

Before proceeding with the experiments, I prepared a simulation to verify how the calibration procedure performs. The simulation resembles dynamic calibration. The dynamic calibration is easier to perform and it is desired to find out whether and under which conditions it can be used.

I chose a position and orientation of the world frame of AT960 expressed in the world frame of AT901 and calculated the true transformation matrix \mathbf{T}_{12} . Then, I generated trajectories ${}^1\mathbf{r}_1$ of length 100 points. Position vector ${}^2\mathbf{r}_2$ was calculated using the inverse of true \mathbf{T}_{12} matrix following (5.13).

Two different levels of Gaussian white noise were added. First, the maximum value of the random component was set to 15 micrometers, next it was set to 500 micrometers. These values correspond to the range of accuracy of the laser tracker measurements declared by the

manufacturer [36]. Therefore, the two designed variants represent the best and the worst case of the real measured data regarding their accuracy.

■ 5.4.6 Measurement scenarios

■ Simulation

The generated simulation data were of two types, random and circular. The random data were generated as a random walk, the circular using harmonic functions.

■ Experiment

First set of measurements was done to collect calibration data. The aim of calibration procedure is to gather data from both laser trackers following one common target. First, we did dynamic measurement, performing random movement of one reflector held by hand. Next, we did static measurements. The reflector was put on 12 different places, where it was left static for several seconds. The 12 short sets of static measurements were put together and used for calibration.

For the next measurement scenario, we made use of the laboratory equipment and mounted the solid bar with two fixed reflectors on a rotary drive (Fig. 5.2). The laser trackers measured circular trajectory of reflectors, either each following one reflector or both following the same one. This scenario, when the true trajectory is known, serves well for analysis of the goodness of calibration.

Lastly, we measured with two reflectors fixed on the solid bar held in hand. We performed random movements possibly similar to those done during painting.

The sampling frequency was set to 1000 *Hz*.

■ 5.4.7 Results

We conducted two series of laboratory measurements with two laser trackers following scenarios in section 5.4.6. All data were post-processed and visualized using *laser_data_post.m* and *laser_velocity_acceleration.m*. Based on an analysis of results, I drew several conclusions, which I present in this section. The conclusions are supported by visualizations in Fig. A.1 - A.26 in the appendix A.1.

In the presented figures, I use RGB-xyz convention to indicate coordinate frames. The trajectory plots are final frames of animation done by the visualization part of the script, the large coordinate frame represents the global frame, the small one is the local frame and moves

during the animation based on the measured orientation. There are always both tracked reflectors shown in the trajectory plot, one in black and one in magenta color.

■ Conclusions form simulations

- The calibration error, calculated as the absolute difference between ${}^1\mathbf{r}_1$ and ${}^1\mathbf{r}_2$, depends on the level of added Gaussian noise rather than on the type of motion (Fig. A.3, A.4, A.7, A.8).
- When data contain maximum level of noise, the calibration is not accurate enough. The calibration error is in units of millimeters ($10^{-3} m$) (Fig. A.3, A.7).
- The calibration is sufficiently accurate when the level of noise is minimal. The calibration error is in units of hundredths of millimeters ($10^{-5} m$) (Fig. A.4, A.8).

■ Conclusions from experiments

- The dynamic calibration procedure is not accurate enough. The mean value of error, calculated as the absolute difference between ${}^1\mathbf{r}_1$ and ${}^1\mathbf{r}_2$, is in units of millimeters ($10^{-3} m$). That is, compared to the accuracy of individual laser tracker measurement, very large error (Fig. A.10).
- Inaccurate calibration leads to inaccurate trajectory data. There is a visible offset in trajectories after transformation and the orientation calculation is not reliable (Fig. A.11).
- The static calibration procedure is very accurate. The mean values of error, calculated as the absolute difference between ${}^1\mathbf{r}_1$ and ${}^1\mathbf{r}_2$, is in units of hundredths of millimeters ($10^{-5} m$) (Fig. A.13).
- The static calibration is the optimal approach. It was also recommended by experts from the laser tracker manufacturer. However, they did not state any reason why the dynamic procedure should not be used (Fig. A.14).
- The calibration error of dynamic calibration procedure corresponds to the simulated calibration with maximum added noise (Fig. A.3, A.7, A.10).
- The calibration error of static calibration procedure corresponds to the simulated calibration with minimum added noise (Fig. A.4, A.8, A.13).
- It is possible to determine the body local frame orientation from the measurements of two tracked reflectors as described in section 5.4.4. The Euler angles can be directly determined (Fig. A.15, A.16).

- Values of velocity during a random movements resembling movements of spray gun during painting are in units of tenths of meters per second ($10^{-1} \frac{m}{s}$) (Fig. A.18, A.19, A.23, A.24).
- Values of acceleration during a random movements resembling movements of spray gun during painting lie between 0.5 and 6 g (5 to 60 $\frac{m}{s^2}$) (Fig. A.20, A.21, A.25, A.26).
- The laser trackers measuring in real time mode with sampling frequency 1000 Hz suffer from data loss. Sometimes measurement entry is not returned on time and is therefore replaced with the previous one. The smoothness of motion is corrupted and outliers are brought to the data.
- The lost data lead to sharp changes in calculated velocity values. The velocity values then oscillate between zero and a large number (Fig. A.18, A.19, A.24).
- The outliers in velocity created as a result of position data loss propagate further to the acceleration (Fig. A.20, A.21, A.26).
- The AT960 suffers from data loss more than the AT901 (Fig. A.18 - A.26).
- Major problem of this measurement method is the reliance on a clear line of sight between the laser tracker and the reflector. During the measurements of random movements trackers often lost the reflector when it moved too fast or rotated too much.
- When there are multiple reflectors involved in a measurement and a laser tracker loses its reflector, it is very likely that it will find and continue tracking a different one. This strongly complicates application of method with two trackers and two reflectors described in section 5.4.1.

5.5 Inertial Sensor

An inertial sensor or an inertial measurement unit (IMU) refers to a measurement device that consists of an accelerometer and a gyroscope. Such device usually provides measurement in 6 axis using three-axis accelerometer and three-axis gyroscope. A three-axis magnetometer can be added to form a 9-axis sensor.

The IMU sensor outputs measurements in the body coordinate frame (the frame rigidly attached to the sensor) with respect to the inertial reference frame. The gyroscope measures angular velocity ω [rad/s], the rate of change of the sensor's orientation. Accelerometer measures specific force f [m/s^2] consisting of the sensor's linear acceleration a [m/s^2] and

the earth's gravity acceleration g [m/s^2] [23]. To determine the sensor's linear acceleration, the gravity component must be subtracted during post-processing.

Since the IMUs are nowadays often implemented as microelectromechanical systems (MEMS), they can be very small, light and inexpensive. They are widely used in electronic devices such as smart phones, wearables or virtual reality controllers [23].

On the other hand, the nature of MEMS brings several aspects, that negatively influence the sensor performance and accuracy. There have been attempts to analyze and model these issues, while using the models can help to compensate their impact [25]. Among the main issues belong bias and scale factor. Bias (zero offset) is the average sensor output over a specific time measured at specified operating conditions that has no correlation with the input. The scale factor (sensitivity) represents the change in sensor output against the change of sensor input - the measured quantity [24]. Another important issue is that the MEMS IMUs when used in low-g applications such as motion tracking of handheld tools show low signal-to-noise ratio [24].

Both scale factor and bias are influenced by material and construction of the sensor (nonlinearity, hysteresis, axis misalignment etc.) and by the conditions of operation (temperature, pressure) [25]. Their effect can be partially removed by calibration, however it is not a perfect solution. The calibration of an IMU can be performed by leaving the sensor stationary for a short period of time, sensing its data and calculating the mean. This method is sufficient in case of the gyroscope, since it is expected to measure only the earth's angular velocity. The earth's angular velocity is of order 10^{-5} , whereas the bias is usually of order 10^{-2} and hence the earth's angular velocity can be neglected [23].

The accelerometer when stationary should measure the gravity, the centrifugal acceleration and the Coriolis acceleration. The gravity component is significantly higher in magnitude than the other two and should be observed in the measurement of acceleration in z-axis (assuming the sensor coordinate frame to be aligned with the earth coordinate frame). However, the board (PCB) on which the sensor lies and the sensor case in which the accelerometer is mounted are not completely flat and also the accelerometer is not mounted perfectly aligned with the case. As a result, the gravity components in x- and y-axis are nonzero and therefore, it is not possible to distinguish between measured gravity and bias [23], [14].

Another thing to consider is that the sensor bias can vary over time due to a flicker noise [25], [23], which can be hardly modeled.

■ 5.5.1 Method

The option to use an IMU sensor for motion tracking of the spraying gun seems to be very promising. The IMU sensor is light, small, has low power consumption and can be

used wireless. On top of that, the sensor is resistant to the harsh environment of a painting workplace, because the electronics can be closed in an impermeable case. For these reasons, I decided to design and test a solution of the motion tracking task with an IMU sensor. The sensor is meant to be mounted directly on the spraying gun and to measure the operator's movements during the painting operation.

The algorithm of obtaining orientation and pose estimates from data from an IMU sensor is called dead-reckoning. The principal of dead-reckoning is shown in Fig. 5.3. The biggest problem of this algorithm lies in the integration and double integration operations. The noise and bias in the IMU measurements introduce an error that, after integration of such noisy signal, results in an integration drift. The error accumulates with time and increases without bound. After a while, the position and orientation estimates are useless. The literature describing the issue of the integration drift is extensive.

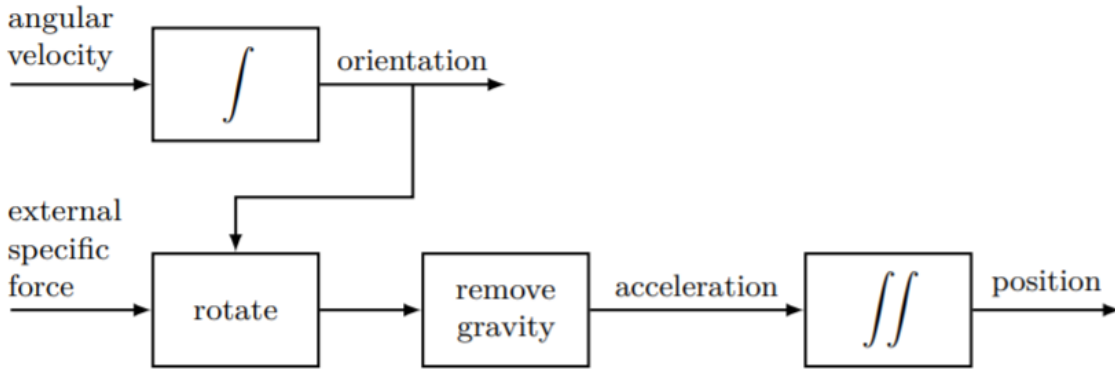


Figure 5.3: The algorithm of dead-reckoning. Orientation and position are determined based on measured angular velocity and external specific force [23].

There are two options how to handle the inaccuracy of position and orientation estimates from IMU sensors. The first is to properly filter the measured data before they are post-processed. This option was implemented for instance in [39], [40] or [41]. The model of sensor's characteristics (bias, nonlinearity etc.) is utilized at this stage to design such pre-processing that together with filtering corrects the most of sensor's error. I focus on this approach in this section.

The second option is to fuse the IMU data with another sensor measuring position and orientation with lower frequency and thus repeatedly correct the IMU estimates. This way, the integration drift is suppressed (bounded) and good position and orientation estimates of a high frequency are obtained. Many such solutions are described in literature, among the additional sensors belong GPS (global positioning system), optical sensors (cameras) or UWB

(ultra-wideband) systems [23]. I will cover this option in section 5.6.

5.5.2 Hardware specification

For the motion tracking of hand-held spray gun by an IMU sensor I chose a product MetaMotionR made by Mbientlab [42]. The MMR is a wireless motion and environmental sensor designed mainly for wearable applications. It includes PCB with six on-board sensors, Bluetooth Low Energy module, NOR flash memory and rechargeable Lipo battery (via micro USB) enclosed in a splash resistant (IP40) plastic case (see Fig. 5.4). The on-board sensors are gyroscope, accelerometer, magnetometer, barometer, luminosity sensor and temperature sensor. I was originally working with a version MMR+ that includes also a vibrating coin motor soldered to the PCB, which I later had to remove, since it was made of metal and was heavily disrupting the magnetometer measurements.

All named sensors are made by Bosch (except the luminosity one by Lite-On). The sensors used for the motion tracking application are BMI160 (3-axis accelerometer and 3-axis gyroscope) [43] and BMM150 (3-axis magnetometer) [44]. An overview of sensor characteristics is displayed in Tab. 5.1.



Figure 5.4: The MetaMotionR+ sensor by Mbientlab. The PCB and battery are enclosed in a plastic case (now open). Dimensions 27x27x4 millimeters, weight 6 grams. The orientation of the body coordinate frame is shown (following the RGB-xyz convention).

	Accelerometer	Gyroscope	Magnetometer
Temperature range	$-40\dots + 85^{\circ}C$	$-40\dots + 85^{\circ}C$	$-40\dots + 85^{\circ}C$
Range	$\pm 2g, \pm 4g, \pm 8g, \pm 16g$	$\pm 125^{\circ}/s, \pm 250^{\circ}/s, \pm 500^{\circ}/s,$ $\pm 1000^{\circ}/s, \pm 2000^{\circ}/s$	$\pm 1300\mu T$ (x, y - axis), $\pm 2500\mu T$ (z - axis)
Sensitivity	$\pm 2g : 16384$ LSB/g $\pm 4g : 8192$ LSB/g $\pm 8g : 4096$ LSB/g $\pm 16g : 2048$ LSB/g	$\pm 125^{\circ}/s : 262.4$ LSB/ $^{\circ}/s$ $\pm 250^{\circ}/s : 131.2$ LSB/ $^{\circ}/s$ $\pm 500^{\circ}/s : 65.6$ LSB/ $^{\circ}/s$ $\pm 1000^{\circ}/s : 32.8$ LSB/ $^{\circ}/s$ $\pm 2000^{\circ}/s : 16.4$ LSB/ $^{\circ}/s$	-
Offset	$\pm 40mg$	$\pm 3^{\circ}/s$	$\pm 40\mu T$
TCO	$\pm 1mg/K$	$0.05^{\circ}/s/K$	-
Noise density	$180\mu g/\sqrt{Hz}$	$0.007^{\circ}/s/\sqrt{Hz}$	-
Resolution	16 bit	16 bit	$0.3\mu T$

Table 5.1: Mbientlab MMR motion tracking sensors characteristics [43],[44].

5.5.3 Software specification

Mbientlab provides an SDK for sensor control MetaWear C++ API [45] and wrappers of the shared C++ library for several other programming languages. There exists a basic app for Android, Windows and iOS for data acquisition but the developers can use the APIs available in Swift, C++, C#, Javascript and Python to create their own applications.

Since the MetaWear App provides only basic access to sensor configuration, measurement settings and visualization, I designed my own application. It was developed on Linux and written in Python 3 using the shared C++ MetaWear library with Python API wrapper and bindings [45]. At this stage, the application serves the research and development purposes of this thesis and it is accessible through the command line. The configuration is done either by parameters part in the head of the script or passed as command line arguments. Later, the procedure can be easily adjusted to fit the hardware and software of the user. To run the application described in this section, the user needs to follow the initial steps of downloading and installing the MetaWear SDK, Python API etc. described in the Mbientlab tutorial page, in section Linux/Python APIs [46]. The source code of the application is available in the appendix C.2.

Sensor fusion software

An important feature of the Mbientlab sensor is that it comes equipped with the 9-axis Bosch Sensortec Sensor fusion software [47]. The Kalman filter based fusion software fuses raw measured data from accelerometer, gyroscope and (optionally) magnetometer to calculate absolute or relative orientation of the sensor. There are four different operation modes - NDoF (absolute orientation), IMUPlus, Compass and M4G (relative orientation) [48]. The

orientation is expressed in form of Euler angles or quaternions w.r.t. NED coordinate frame (in case of absolute orientation) or initial coordinate frame (in case of relative orientation). The initial frame coincides with the sensor coordinate frame at time zero of the measurement. The sensor coordinate frame is depicted in Fig. 5.4.

The Sensor fusion software also utilizes the sensor error models and offers corrected values of measured angular velocity and corrected acceleration together with measured gravity vector and linear acceleration, which is the acceleration vector free of gravity components. The fusion algorithm makes use of factory calibration data, while the results shall be further improved, if the user performs calibration procedure before the respective measurement.

There are several limitations connected with the usage of the Bosch sensor fusion. The update frequency is 100 Hz , which is significantly lower compared to the maximal raw data output frequency (3200 Hz for accelerometer and 1600 Hz for gyroscope). Further, the software does not allow more than two fused data outputs at a time and also lowers the number of additional raw data outputs that can be logged. In practice it means that the user is unable to measure for example quaternions, raw acceleration and gravity together. Another thing to keep in mind is, that even the calibrated sensor fusion can hardly correct the varying bias or unexpected magnetic disturbances (i.e. presence of metal object near the sensor).

■ Streaming and logging data

The MMR sensor runs in two different modes - streaming and logging. In general, the data signals integrated in the developed application are: *Euler angles [deg]*, *quaternions [1]*, *linear acceleration [g]*, *gravity vector [g]*, *raw acceleration [g]*, *raw angular velocity [deg/s]* and *magnetic field intensity [T]*. The measured data are always transferred by the Bluetooth Low Energy technology, which limits the amount of data transferred by the unit of time. During the streaming, the data are sent to controlling PC in the real time, allowing for real-time visualization. During the logging, data are stored in the sensor internal memory and downloaded at a later stage, when the measurement finishes. As a result, the streaming mode allows only one sensor fusion signal at a time or one (or a combination of more) raw data signals depending on the chosen sampling frequency.

The streaming helps to quickly check if the sensor is working, what is the range of measured values, identify the orientation of NED coordinate frame, discover sources of magnetic distortions etc. To realize such test measurement, the user runs the *stream_and_plot.py* script, passing the command line arguments in form of: *data signal* (eul/ quaternion/ linacc/ gravity/ acc_gyro_mag) *measurement duration* (float, in seconds) *calibrate* (True/ False) *print_measurement* (True/ False). The script *stream_and_plot.py* further runs *stream_multi_sensor.py* where can be found and manually set low-level configuration of

the sensors if needed and *animate_multi_sensor.py* that handles the visualization of the measured data. The example command for streaming Euler angles for 10 seconds with prior calibration and without printed results in the terminal window would look as follows:

```
python3 eul 10 True False
```

For the purpose of spraying tool tracking and trajectory generation the sensor should be run in logging mode. The logging mode stores the measured data signals into the internal flash memory first, which makes it possible to measure more data outputs at a time with higher frequency than by streaming, because it does not involve the Bluetooth transfer. The data are downloaded from the memory via Bluetooth later, once the measurement finishes.

There are however still some limitations of the number and frequency of outputs, caused by the Sensor fusion algorithm described in section 5.5.3.

The script for data logging with MbiEntlab MMR sensor is called *log_multi_sensor.py*. The script controls the sensor, data logging and downloading, storing in a form of csv file and basic data visualization. The high-level configuration can be accessed in the head of the script, allowing to choose which data signals should be logged (Euler angles, quaternions, linear acceleration, gravity, acceleration, gyroscope, magnetometer). One or more than one can be set True, but not all combinations are feasible by the sensor. Working combinations of signals are for example eul+linacc, linacc+gravity+acc, gravity+acc, quaternion+acc or acc+gyro+mag. Those that are not working are for instance eul+gravity+acc, quaternion+gravity+acc, eul+linacc+acc. Further, the user chooses if to perform calibration prior to the measurement or not, how long the measurement should last (in seconds), if to print the results in the terminal and the output folder, where the csv files with measured data will be stored.

The structure of the *log_multi_sensor.py* is following. The PC first connects to the MMR sensor via Bluetooth using the sensor mac address. At the end of the sensor operation the sensor must be again properly disconnected. In case the sensor wasn't disconnected properly after the last operation due to an unexpected error or similar reasons, the following logging procedure fails. Because of that, the script first connects and disconnects the sensor and then connects again.

The interaction with the sensor is implemented through a class *State* that includes three main methods: *setup*, *start* and *stop*. Further, each data signal has its own data handler method that manages data download and write to file operations.

In the *setup* method, the sensor signals are first configured. The user can access the low-level configuration parameters here. Namely they are: accelerometer data range and sampling frequency, gyroscope data range and sampling frequency, magnetometer mode

Euler angles	<i>eul_data_timetag.csv</i>	epoch [ms]	pitch [deg]	roll [deg]	yaw [deg]	heading [deg]
Quaternions	<i>quaternion_data_timetag.csv</i>	epoch [ms]	w [1]	x [i]	y [j]	z [k]
Linear acceleration	<i>linacc_data_timetag.csv</i>	epoch [ms]	x-axis [g]	y-axis [g]	z-axis [g]	
Gravity vector	<i>gravity_data_timetag.csv</i>	epoch [ms]	x-axis [g]	y-axis [g]	z-axis [g]	
Acceleration	<i>acc_data_timetag.csv</i>	epoch [ms]	x-axis [g]	y-axis [g]	z-axis [g]	
Angular velocity	<i>gyro_data_timetag.csv</i>	epoch [ms]	x-axis [deg/s]	y-axis [deg/s]	z-axis [deg/s]	
Magnetic field intensity	<i>mag_data_timetag.csv</i>	epoch [ms]	x-axis [T]	y-axis [T]	z-axis [T]	

Table 5.2: Structure of the csv file with downloaded logged data from the MMR IMU sensor.

and sensor fusion mode. Note that if the sensor fusion output is chosen, the sensor fusion configuration parameters override the accelerometer and gyroscope parameters (e.g. it is not possible to change the sampling frequency of the accelerometer, while it is set to sensor fusion default 100 *Hz*). All the available configuration constants can be found in the python script *cbindings.py* on the GitHub page accessible through the MetaWear SDK [45]. Next, a logger object is created for each chosen data signal.

Next, the *start* method triggers logging operation, enables respective data signals and starts logging. When the *stop* method is called, it stops the logging, disables the data signals and subscribes to all active loggers to download the logged data. The data are downloaded with the help of respective data handlers and saved into csv files in a designed format. There is always exactly one csv file for each chosen data signal (quaternion, acceleration etc.) and it is saved in the output folder labeled with an automatically generated date and time tag to ensure name uniqueness and clarity. The structure of the individual csv files is described in the Tab. 5.2.

After the logged data are download, the script resets the device configuration and disconnects it properly. Eventually, a plot of measured data is displayed to provide a brief visual check that the measurement was done correctly.

■ Calibration

Thanks to the fact that I design my own application for data acquisition it was possible to include automatic calibration procedure to the process of measurement. The Bosch sensor comes factory calibrated and further calibration should not be necessary. Nevertheless, the Mbientlab tutorials [46] recommend to always perform calibration before measurement if it includes the outputs from Sensor fusion software. The MetaWear basic App does not provide the option to calibrate the sensor, instead a calibration script must be run and for that the user already needs to get involved in the developer style interaction with the device.

Both main streaming and logging scripts mentioned in the section 5.5.3 allow the user to set calibration parameter `True` and provide methods to handle the calibration procedure. The calibration itself consists of three activities and the user is asked to perform it before the script continues to the setup stage.

Firstly, the sensor should be left lying on a flat surface to calibrate for the gyroscope bias. Secondly, the sensor should be rotated around one of its axes (chosen arbitrarily). The rotation should be stopped for a while after each 45° . This should calibrate the accelerometer bias. Thirdly, the user should move the sensor randomly in space, calibrating the magnetometer. During the calibration user can watch the calibration status of each of the three sensors, starting at 0 (not calibrated) and increasing up to 3 (calibrated) during the procedure. When all the sensors are calibrated (value 3), the script continues with the measurement task itself.

It is important to note, that the sensor has to be reset after each measurement operation and it therefore loses its calibrated state. The calibration if desired has to be performed repeatedly at the beginning of each new measurement.

■ 5.5.4 Post-processing

The section 5.5.3 describes the available data outputs of the MMR IMU sensor. The Bosch Sensortech Sensor fusion software fuses raw data to obtain orientation, but there is not further post-processing done in order to calculate velocity and position from the raw data. Since the motion tracking task and painting trajectory generation requires 6DoF data (orientation and position) I attempted to design a post-processing procedure based on the dead-reckoning algorithm (see Fig. 5.3) to calculate the position from accelerometer data. To first get to know the limits of the designed solution (either hardware, based on the sensors characteristics or software, based on the used calculations and functions) I decided to create a simulation of the MMR IMU sensor. The simulation serves as a testing tool and as a proof of concept of the designed post-processing solution. The details follow in the section 5.5.5.

All the post-processing was done in Matlab R2019b. The main script for IMU data post-processing is called `mbientlab_data_post.m` and it utilizes a function `double_int_filter.m`, which takes care of the double integration and the filtering part. It also performs extensive data visualization, including trajectory animation. All source code is available in the appendix (C.2).

The code is configurable through several parameters available at the beginning of the script, of which the most important are the `import` and `name`. By the `import` parameter the user chooses which types of data will be loaded to `data_all` structure for post-processing (quaternions, acceleration etc.). By the `name` the unique csv file is identified (`name` equals the unique time tag created during measurement and included in the csv file name).

■ Synchronization

When the data are loaded into a *data_all* structure, they first need to be synchronized. Each data output in the csv file contains an epoch entry, a time stamp, stating when the data were sampled in form of the unix time stamp. If more than one data signal were enabled and logged, the data entries of each data signal are slightly shifted in time, because the data signals could not be enabled in the very same moment. It is therefore necessary to synchronize the data entries based on the epoch entry. Unfortunately, the resolution of the epoch (the unix time stamp) provided by Mbientlab is only 1 millisecond. In practice, it means that if the used sampling frequency is higher than 1000 *Hz*, there will be more than one data entry with the identical time stamp. Also, the real sampling frequency does not exactly correspond with the one demanded during configuration, due to the hardware limitations of the sensor. To know the exact time elapsed between two consecutive acceleration measurements is very important for the double integration operation, therefore the data synchronization deserves attention.

Each field in the *data_all* structure contains measured data from one data signal (e.g. quaternions, acceleration, gyroscope measurement etc.). Firstly, the start time stamp is identified as the latest one of the initial time stamps of all present data signals. Secondly, the end time stamp is identified as the earliest one of the last time stamps of all present data signals. All the data signals are afterwards cropped to contain only data entries between the start and end time stamp.

Further, the data signal with the highest sampling frequency (the one with the most data entries between the start and the end time stamp) is found and the vector of its time stamps is set to be the synchronization time vector for all the other data signals. The other data signals are upsampled using *interp1* function in a "sample and hold" manner (using the parameter *previous*).

If the average sampling frequency of the synchronization data signal was higher than 1000[*Hz* there are repeated entries in the synchronization time vector. Such time vector could not be used in the double integration operation, because the time difference between two data entries would be calculated as equal to zero and would result in an undefined mathematical operation of zero division. In such case, the time vector is replaced by a vector of linearly spaced values between start and end time stamp of a length identical with the length of the synchronization data signal.

The described manipulations with the data time stamps can possibly affect the results of the post-processing operation. However, it cannot be avoided, since it originates from the hardware and software character of the used sensor.

■ Transformation

A crucial issue of any motion tracking task is the correct understanding of used coordinate frames and knowledge of their relationship to measured data. The MMR IMU sensor produces raw measurement expressed in the body coordinate frame (local). The local frame is rigidly attached to the sensor and its orientation is obvious from the Fig. 5.4. The local coordinate frame moves together with the sensor. The Sensor fusion algorithm produces all data except for the orientation measurements also w.r.t. the local frame. The orientation data (Euler angles and quaternions) are expressed either w.r.t. the NED (north-east-down) coordinate frame or the initial coordinate frame (global). The first case is valid when NDoF fusion mode is used, which is the case for all the measurements presented in this work.

I chose to express all measured data in the initial coordinate frame (global). However, in case of the MMR IMU sensor, I make use of the orientation expressed w.r.t. the NED frame, because I needed to subtract the gravitational component from the vector of specific force measured by the accelerometer. For the transformation calculations I was following common rules and relationships described for instance in [33].

Firstly, I calculate the matrix of directional cosines $\mathbf{S}_{\mathbf{wg}}$ that describes the transformation from the NED coordinate frame (world) to the initial coordinate frame (global) using the initial orientation data entry expressed in the world frame. The $\mathbf{S}_{\mathbf{wg}}$ matrix is used to transform vectors in the global frame into vectors in the world frame. Secondly, I calculate the matrices of directional cosines $\mathbf{S}_{\mathbf{wl}}^m$ that describe the transformation from the world frame to the local frame from each orientation data entry. The matrix of directional cosines can be formed either using Euler angles following (5.2) or quaternions following (5.1).

Using the $\mathbf{S}_{\mathbf{wl}}^m$, the local acceleration measurements can be transformed to the world frame following this equation:

$$\mathbf{a}_{\mathbf{w}}^m = \mathbf{S}_{\mathbf{wl}}^m \mathbf{a}_{\mathbf{l}}^m, \quad (5.22)$$

where the $\mathbf{a}_{\mathbf{w}}$ stands for the acceleration vector expressed in the world frame and $\mathbf{a}_{\mathbf{l}}$ stands for the acceleration vector expressed in the local frame. The superscript m denotes the measurement (data entry) index.

Note that we do not need to use the full transformation matrix \mathbf{T} [33] that includes rotation and translation part, since we are interested only in the relative orientation of the considered frames and do not even have the information about the displacement.

When the raw acceleration is used, the gravity vector must be subtracted to obtain gravity free acceleration. In case the linear acceleration is used, this step is skipped, because the linear acceleration is already gravity free. The gravity vector $\mathbf{g}_{\mathbf{w}}$ expressed in the NED coordinate system is $\mathbf{g}_{\mathbf{w}} = [0 \ 0 \ g_0]^T$, where $g_0 = 9.80665 \text{ m/s}^2$ stands for the standard

acceleration due to Earth's gravity according to the International Bureau of Weights and Measures [49]. The gravity free acceleration $\mathbf{a}_{\mathbf{w},\text{free}}^m$ at each time step m is therefore calculated as follows:

$$\mathbf{a}_{\mathbf{w},\text{free}}^m = \mathbf{a}_{\mathbf{w}}^m - \mathbf{g}_{\mathbf{w}}. \quad (5.23)$$

Thirdly, the acceleration is transformed from the world frame to the global frame using the $\mathbf{S}_{\mathbf{wg}}$ matrix. The transformation needed here is inverse to the one obtained from the orientation measurement, the matrix $\mathbf{S}_{\mathbf{wg}}$ must be therefore inverted:

$$\mathbf{a}_{\mathbf{g}}^m = (\mathbf{S}_{\mathbf{wg}})^{-1} \mathbf{a}_{\mathbf{w},\text{free}}^m, \quad (5.24)$$

the $\mathbf{a}_{\mathbf{g}}^m$ stands for the gravity free acceleration vector expressed w.r.t. to global coordinate frame. The $\mathbf{a}_{\mathbf{g}}$ is further used in position calculations.

Eventually, the measured orientation expressed w.r.t. the NED frame needs to be transformed to the global frame as well. In other words, we would like to have orientation that describes transformation from the global frame to the local frame at each time step. Such transformation would be denoted as $\mathbf{S}_{\mathbf{gl}}^m$ and based on the rules of transformation it can be expressed using the $\mathbf{S}_{\mathbf{wl}}^m$ and $\mathbf{S}_{\mathbf{wg}}$:

$$\mathbf{S}_{\mathbf{gl}}^m = (\mathbf{S}_{\mathbf{wg}})^{-1} \mathbf{S}_{\mathbf{wl}}^m. \quad (5.25)$$

The orientation in form of Euler angles or quaternions can be then derived from the $\mathbf{S}_{\mathbf{gl}}$ matrix directly using (5.3) and (5.4).

■ Double integration

The vector of acceleration $\mathbf{a}_{\mathbf{g}}$ represents the acceleration of a particle - the origin of the local coordinate frame of the IMU sensor - in three axes of the fixed global coordinate frame. The aim is to calculate the position vector $\mathbf{r}_{\mathbf{g}}$ that would describe the motion of the particle. For that purpose, equations (5.8) and (5.9) from section 5.3 were used.

For the calculation of discrete values of the components of the position vector $\mathbf{r}_{\mathbf{g}}$, the a_x , a_y , a_z in the equation (5.8) are substituted with components of measured $\mathbf{a}_{\mathbf{g}}$. The superscript indicates the time step, or rather the measurement index, going from $k = 1$ to $k = K$, where K is the number of measurements. The expression $(t^{k+1} - t^k)$ stands for the length of the k -th time step, that is calculated from the vector of time stamps corresponding to the $\mathbf{a}_{\mathbf{g}}$ data, referred in section 5.5.4. The initial conditions $v_x^1, v_y^1, v_z^1, x^1, y^1, z^1$ were assumed zero.

■ Filtering

The procedure of position calculation from acceleration measurements described in section has certain shortcomings. The goodness of approximation of derivatives depends

on the length of time steps ($t^{k+1} - t^k$), getting worse with longer time steps. This can be influenced by the used sampling frequency during measurement. The main flaw already mentioned in 5.5.1 lies in the fact, that any small inaccuracy in the initial conditions estimate or acceleration measurements propagates through time and eventually leads to very inaccurate position data. For that reason, digital filtering was applied. All filtering was applied offline during post-processing.

Firstly, the post-processing script handles possible outliers using matlab *filloutliers* function with 'quartiles' method, because the data are not normally distributed [50]. Secondly, I filtered the acceleration measurements \mathbf{a}_g using Savitzky-Golay filter. It is a smoothing filter based on least-square fitting of polynomial to data in a moving window. It is similar to standard averaging FIR (Finite Impulse Response) filters or moving average filters (special case of FIR filters), but recommended for signals with a large frequency span [51]. Smoothing was used in place of low-pass filtering to remove high-frequency noise and implemented using *sgolayfilt* Matlab function available in the Signal Processing Toolbox. I decided for Savitzky-Golay filter based on several experiments, where it outperformed the general moving average filter. The parameters of the filter are the polynomial order, which I mostly set to equal five, and window size, which I adjusted based on the sampling frequency of the measurement and the nature of the captured motion.

Thirdly, it was needed to handle the measurement drift. Without that, the measurements and therefore, even the position calculations were non-zero even when the sensor was static. For that purpose, I applied a high-pass filter, with cut-off frequency below 1 Hz. The cut-off frequency parameter has to be tuned based on the nature of the motion. Also, it must be taken into account, that the drift varies in time and therefore even the length of the motion should be considered when designing the high-pass filter. The experiments showed that it is useful to apply the high-pass filter not only to the acceleration data but also to the integrated velocity data and further to the integrated position data. The repeated filtering handles the drift introduced by the mathematical operation of integration based on approximations in (5.8) and (5.9) and can help to get better results.

There are several implementations of high-pass filter in Matlab Signal Processing Toolbox, of which I implemented and tested high-pass FIR filter designed using *designfilt* with parameter 'highpassfir' [52], Butterworth high-pass filter using *butter* with parameter 'high' [53] and a general high-pass filter using *highpass* [54]. The filtering and double integration is done by a function script called *double_int_filter.m* and controlled by parameters configurable from the main script *mbientlab_data_post.m*. The parameter *filtering* determines what signals should be high-passed filtered (acceleration and/or velocity and/or position) and what filters to use (Savitzky-Golay and/or high-pass FIR or Butterworth high-pass or general high-pass). The

simulation and later the experiments showed that the optimal combination is the acceleration data smoothing by Savitzky-Golay filter and high-pass filtration of all data signals with general high-pass filter.

■ 5.5.5 Simulation

A useful tool for analysis of how the designed solution of motion tracking task using IMU will work offers a measurement simulation. I made use of Matlab Sensor Fusion and Tracking Toolbox *imuSensor* function that allows the user to create an IMU simulation model based on the datasheet characteristics [55].

I designed two models - one of an ideal IMU, measuring without offset and noise, and one of the Mbientlab MMR sensor based on the characteristics from the Tab. 5.1. The object created by the *imuSensor* function takes motion data in the NED frame as arguments and simulates the sensor measurements of such motion. The simulation produces acceleration, gyroscope and magnetometer data in the body (local) frame. The examples in [55] recommend to generate motion data so, that the user defines motion in the local frame and uses *kinematicTrajectory* function [56] to transform them to the NED frame. In such way, the ground truth data for position, orientation, velocity, acceleration and angular velocity are obtained and available for later comparison. This fact motivated me to use the NED frame for simulated measurements reference frame instead of the initial one, used during the experiments.

It is not possible to simulate the MMR Sensor Fusion algorithm, because it is a commercial product of Bosch Sensortech and users do not have access to equations and mechanisms that stand behind it. Therefore, I did not try to simulate the orientation data output of the MMR sensor and focused on the acceleration data instead. I also added a portion of white Gaussian noise to some simulated measurements to match the real-world scenarios more closely.

The simulated measured acceleration data are further post-process the same way as it is intended with the real measured data, described in section 5.5.4 (transformation, filtering and double integration). As the ground truth orientation in the world frame, needed for acceleration transformation and gravity component subtraction, I used the available ground truth quaternions data. Also, it is important to note that the simulated measured acceleration returned by the IMU model, corresponding to measured specific force, is in the Matlab toolbox defined differently compared to the conventional definition [57]. Because of that, the simulated acceleration data have to change sign.

The simulation results are interesting for this work from three different points of view. Firstly, it shows how well the MMR IMU sensor performs compared to the ideal IMU sensor. Secondly, it shows how the designed post-processing works since the results can be compared

with simulated ground truth. Thirdly, it allows to observe how does the real MMR IMU sensor measure compared to the IMU model based on the MMR datasheet.

■ 5.5.6 Measurement scenarios

To test if the motion tracking can be done by the Mbientlab MMR IMU sensor I performed several simulations and experiments. First, I was interested in the measurements of steady state. The steady state data serve for noise inspection as described in [23]. Then I wanted to see how the sensor measures typical motions such as rotation about one axis without translation, harmonic motion in one or more axis without rotation and simple translation in one axis.

I also wanted to compare results obtained by three different types of acceleration measurements that the MMR IMU offers - linear acceleration with prior calibration, linear acceleration without calibration and raw acceleration. Lastly, I was interested if the sensor performance is the same when moved along all the three body frame axis (x,y,z).

■ Simulation

It is possible to simulate raw data outputs only, therefore I always simulated accelerometer, gyroscope and magnetometer readings.

I was able to simulate the steady state measurement by passing zero local acceleration vector and zero local angular velocity vector to *kinematicTrajectory* function, obtaining all global outputs equal to zero as well. The rotation about one axis was done by passing zero local acceleration vector and local angular velocity vector with one element equal to a nonzero constant. The harmonic motion was achieved by generating local position vector with elements equal to sine functions of time and double differentiating them to get the local acceleration vector. The local angular velocity was set zero in this case. I did not simulate the simple translation, because the assumption of constant acceleration would be oversimplified and modelling the real shape of acceleration curve too difficult.

■ Experiment

The steady state measurements were done by simply letting the sensor to lie on a flat surface.

Due to the fact that during the work on this thesis the access to laboratories was restricted, I was not able to use any positioning device such as rotational or linear motor for experiments. I had to construct my own measurement device to be able to perform translation of known distance and harmonic motion of different frequencies with known amplitude of



Figure 5.5: The slider-crank measurement device built to perform experiments with Mbientlab MMR sensor and RealSense T265 tracking camera. The device consists of three revolute joints and one sliding joint realized by four wheels. The MMR sensor can be placed in the paper boxes in three different configurations, the camera can be mounted with two screws at the end of the sliding part in three different configurations as well.

position. The measurement device was built as a slider-crank mechanism with three revolute joints and one sliding joint, driven by an electric hand drill with adjustable speed (slow/fast). I mounted a sort of a jig at the sliding end allowing to place the MMR sensor in three different configurations - having the body frame x -, y - or z -axis parallel to the direction of sliding. The length of the rotating link is 68 millimeters, therefore, the maximal length of the end point trajectory is equal to 136 millimeters. The device is displayed in Fig. 5.5.

■ 5.5.7 Results

I conducted many simulations and experiments based on scenarios described in section 5.5.6, post-processed the data according to section 5.5.4 and visualized the results using

Matlab `mbientlab_data_post.m` script. Based on the observed results I drew several conclusions that I present in this section. The conclusions are supported by visualizations showed in Fig. A.27 - A.58 in the appendix A.2.

In the presented figures, I use RGB-xyz convention to indicate coordinate frames. The acceleration, velocity and position plots show the original, partially filtered and fully filtered (acceleration, velocity and position) signals for comparison. The trajectory plots are final frames of animations done by the visualization part of the script, the large coordinate frame represents the global frame, the small one is the local frame and moves during the animation based on the measured orientation.

■ Conclusions form simulations

- The drift of MMR IMU measurements of steady state can be, in the worst case, significant and should not be ignored (Fig. A.27 and A.28).
- Inaccuracy in the accelerometer measurements can result in false nonzero data when they should be zero (Fig. A.30 and A.31). This effect leads to nonzero position calculations (Fig. A.32, A.33).
- The negative affect of noisy acceleration can be canceled using appropriate filtering methods (Fig. A.30).
- The drift is present in the velocity and position data even when the acceleration is filtered. The best results are obtained if also velocity and position are filtered (Fig. A.36 - A.38).

■ Conclusions from experiments

- The noise inspection of data measured in steady state confirms that the prior calibrated linear acceleration measurement has the lowest bias, followed closely by not calibrated linear acceleration. However, none of them is equal zero. The bias of raw acceleration is significantly higher (Fig. A.40 - A.42).
- The measured bias of all three types of acceleration data is greatly smaller than the worst case depicted by simulations (Fig. A.28, A.40 - A.42).
- The bias magnitude varies (mostly increases) in time (Fig. A.43).
- There is an observable drift in the orientation measurements as well, even though they are produced by the Sensor fusion algorithm that should handle it. However, the drift is minor (Fig. A.44 and A.45).

- Although both calibrated and not calibrated linear acceleration outperforms the raw acceleration data in terms of bias, all three data types result in very poor position data for translation movement (Fig. A.46 - Fig.A.48).
- The measured data are noisier by a significant portion than the simulated data. Smoothing is necessary (Fig. A.30, A.31, A.35 and A.49, A.50, A.51, A.54, A.55, A.56).
- The negative affect of noisy acceleration can be canceled using appropriate filtering methods (Fig. A.51, A.56).
- The drift is present in the velocity and position data even when the acceleration is filtered. Better results are obtained if also velocity and position are filtered (Fig. A.52, A.53, A.57, A.58).
- The high-pass filter tends to produce data that have zero mean (Fig. A.52, A.53, A.57, A.58).

■ General conclusions

- All types of acceleration data suffer from bias and need to be filtered. I chose to continue the measurements using raw acceleration data, because they can be sampled by much higher frequency and I can fully control how they are post-processed compared to the Sensor fusion software outputs.
- The orientation data either in form of quaternions or Euler angles are of sufficiently high accuracy and can be used further without any adjustment. I chose to use quaternion format, since it is universal compared to Euler angles that are dependent on the sequence of rotation and can lead to loss of information due to an effect called gimbal lock [23].
- The cut-off frequency of high-pass filter needs to be adjusted based on the type of motion and may not be optimal with respect to data in all axis during the whole motion.
- There is no observable difference in performance when compared between measurements of movements aligned with x-, y- or z-axis.

■ 5.6 Visual and inertial sensor fusion

As mentioned earlier in section 5.5, an inertial sensor alone is not good enough to solve a motion tracking task. The reason for it is that it provides measurements that are accurate only for a short time. When used for longer measurements, the error accumulates and the

data drift [23]. A possible solution to this issue is to combine a high-frequency IMU with another sensor of lower sampling frequency, that would correct the effect of drift. One such solution includes a visual sensor (camera) and the resultant motion tracking technique is called visual and inertial sensor fusion, or visual inertial odometry [58].

The visual inertial odometry belongs to a family of SLAM (Simultaneous Localization and Mapping) problems. SLAM problem is defined as a task for mobile robot to localize itself in an unknown environment, while moving in the environment [59]. In our case, the mobile robot represents the moving spraying gun in worker's hand.

5.6.1 Method

The visual part of the described system refers to acquisition of pictures of surroundings and extracting relevant information from it. A picture taken by a camera consists of many (hundreds of thousands) pixels, depending on the camera resolution. A feature detection algorithm is applied to the picture to obtain more condensed information. Features can be any visually interesting parts such as corners or edges, seen in the picture. This way the amount of points to consider lowers from hundreds of thousands pixels to few hundred points (features). Each feature point is labeled, mapped to 2D or 3D space and arranged to a feature vector. When a new picture is taken, a feature tracking algorithm compares labeled features in the new vector with those of previous pictures and computes the change in position based on that [58].

The described procedure is computationally expensive and thus cannot be performed at high frequency. The visual part is therefore complemented with an inertial sensor, that can sample at high rate and offers accurate data in short time range (tens of milliseconds) [58]. The data from the two parts are fused repeatedly, which results in accurate high-frequency position and orientation estimates.

Regarding the task of spraying gun motion tracking, a visual and inertial sensor fusion seems to be a valid option. It has potential to overcome the problems with drift that the IMU, when used alone, has. At the same time, the additional visual sensor (camera) can be also relatively small, light and inexpensive. It can be therefore placed directly on the spraying gun without affecting the worker's motions in an intense negative way. However, this measurement set-up contains issues that make it suboptimal.

Firstly, the sensor fusion algorithm is computationally expensive and is performed in real time. The device therefore needs a power supply and can hardly be realized wireless. Secondly, the sensor needs an appropriate lighting conditions to operate properly. Thirdly, the optical system that is part of the camera is not resistant to the harsh environment of a painting workplace. The paint present in the environment in form of aerosol would stick

	Cameras	Accelerometer	Gyroscope
Range (field of view)	$163 \pm 5^\circ$	$\pm 4g$	$\pm 2000^\circ/s$
Sample (frame) rate	30 <i>fps</i>	62.5 <i>Hz</i>	200 <i>Hz</i>

Table 5.3: Intel RealSense tracking camera T265 sensors characteristics [60].

to the camera, corrupt the acquired pictures and eventually destroy the whole sensor. To prevent this, it is necessary to place the camera in a disposable protecting case. The case should be transparent and thin to not to influence the optical properties of the camera though. [58]. In practice optical sensors and cameras are used in harsh environments even for painting applications similar to ours with commercial success [11], [10].

What influences the sensor performance further is the visual richness of the captured scene (a smooth, unicolor surface lacks features to track), dynamics of the scene (too many moving objects in the scene lead to algorithm confusion) and the speed of sensor motion (too low or too high speed decrease the confidence of pose estimates) [58]. None of these issues should have any strong impact on our application.

5.6.2 Hardware specification

As a representative of a device performing visual and inertial sensor fusion, I chose the tracking camera T265 by Intel RealSense [21] (Fig. 5.6). It is a stand-alone compact device that includes visual and inertial part as well as a processing unit performing SLAM algorithm. The vision is provided by two OV9282 monochrome imagers with fisheye lenses with fixed focus. Included inertial sensor is BMI055 IMU by Bosch Sensortech [60]. Some important sensor features are shown in Tab. 5.3. Further, the T265 includes a processing unit Intel Movidius Myriad 2.0 VPU. The camera connects with micro B USB 3.1 cable to a computer.

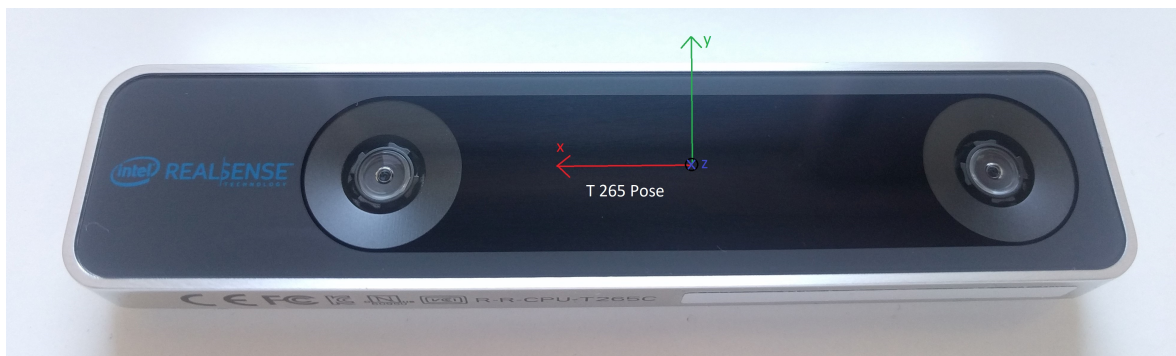


Figure 5.6: The tracking camera T265 by Intel RealSense. Dimensions $108 \times 24.5 \times 12.5$ millimeters, weight 60 grams [60]. The orientation of the body coordinate frame is shown (following the RGB-xyz convention).

■ 5.6.3 Software specification

The T265 is equipped with a processing unit and is able to run the sensor fusion algorithm on board. The software is optimized by Intel and user is not introduced to any details of the algorithm, neither has an option to configure its functioning or adjust it. Regarding the accuracy of pose estimates, the drift is declared to be smaller than 1% of traveled distance [60].

Intel RealSense provides an opensource software interface for working with their products including the T265 called Intel RealSense Software Development Kit (SDK) 2.0 [61]. It can be installed on Linux, Windows, Mac OS or Android and supports many programming languages (Python, ROS, Matlab etc.) through included wrappers. Part of the SDK are librealsense tools providing basic interaction between the user and the sensor and a large amount of examples that are ready to use and well documented.

■ Streaming and logging data

I installed the SDK on Linux and automatically gained access to the tools and examples through terminal commands. For the purpose of my application I utilized two tools, *realsense-viewer* [62] and *rs-data-collect* [63]. The realsense viewer [62] is a tool with GUI, which I used for data streaming. It shows outputs from the accelerometer, gyroscope and two fisheye cameras in real-time. Also, it visualizes position and orientation estimates in 2D and 3D view together with the confidence level, which is useful for determination of motion speed limits. It allows for quick check if everything is working fine and for rough estimate of the location of measurement reference coordinate frame.

The tool *rs-data-collect* [63] allows for data logging into csv file. It is a low latency profiler with zero frames drops and it logs a timestamp in thousandths of milliseconds for each data record (hardware in unix time and host in measurement relative time). The tool does not store the camera images, but these are not needed for our application. The measurement parameters are set using a configuration cfg file where user specifies the demanded data streams, stream format, frequency of streaming and data format. For the T265 it is possible to log accelerometer, gyroscope, fisheyes, and pose streams. The pose stream consists of the 6DoF data and its maximum streaming frequency is equal to 200 Hz .

The *rs-data-collect* tool is then run using command line arguments *-c* specifying the cfg file, *-f* specifying the output csv file and *-t* or *-m* specifying the minimum measurement length in seconds or in number of received frames respectively. An example of the usage of the tool looks as follows:

Stream type	Index	Frame number	HW timestamp [ms]	Host timestamp [ms]	Data			
<i>Fisheye</i>	1	1 ...	unix time	time				
<i>Fisheye</i>	2	1 ...	unix time	time				
<i>Gyro</i>	0	1 ...	unix time	time	x-axis [rad/s]	y-axis [rad/s]	z-axis [rad/s]	
<i>Accel</i>	0	1 ...	unix time	time	x-axis [m/s ²]	y-axis [m/s ²]	z-axis [m/s ²]	
<i>Pose, translation</i>	0	1 ...	unix time	time	x-axis [m]	y-axis [m]	z-axis [m]	
<i>Pose, rotation</i>	0	1 ...	unix time	time	$x[i]$	$y[j]$	$z[k]$	$w[1]$

Table 5.4: Structure of the csv file with downloaded logged data from the Intel RealSense tracking camera T265.

```
rs-data-collect -c data_collect.cfg -f log.csv -t 60 -m 1000
```

It will apply streaming configuration from `data_collect.cfg`, then stream and collect data for 60 seconds or 1000 frames (whatever comes first). The resulted data will be saved into `log.csv` file. Data from all streams are saved into one csv file. The structure of the csv file is illustrated in Tab. 5.4.

■ 5.6.4 Post-processing

Compared to the data from the MMR IMU sensor (section 5.5), the tracking camera performs sensor fusion that outputs 6DoF pose data. The 6DoF information about position and orientation of the camera during measurement is exactly what we need for the motion tracking application and thus it is not necessary to apply any further post-processing algorithm. As a result it was not necessary to create any simulation and it even would not be possible since the Intel RealSense fusion algorithm is a commercial product and the user have no access to details about it. However, the data still need to be arranged to a required format in order to be later used for robot trajectory generation.

All the post-processing was done in Matlab R2019b and consists of synchronization and coordinate frames transformation. The main script for tracking camera T265 data post-processing is called `realsense_data_post.m`. All source code is available in the appendix (C.3). The code is configurable through several parameters available at the beginning of the script, of which the most important is the *name*. By the *name* the unique csv file is identified (*name* corresponds to the csv file name defined as an argument for `rs-data-collect`) and its content is loaded into `data_all` structure. The script also performs extensive data visualization, including trajectory animation.

■ Synchronization

The *data_all* structure contains data from several streams, based on the passed measurement configuration file. In case only pose data were collected, it is not necessary to perform any synchronization step, since the 6DoF data are produced by the fusion algorithm at one time and each data entry has its timestamp in sufficient resolution (thousandths of milliseconds). In case also raw data were logged and should be further used to produce user defined outputs, they need to be synchronized.

The individual streams do not output data in the very same time and with the same frequency. Therefore, the post-processing script first finds the latest start timestamp of all data streams and the earliest end timestamp of all data streams. Then, it crops data in all data streams so, that only the data between the latest start and the earliest end timestamp are kept. Further, it identifies the stream with the highest number of samples and defines its timestamp vector as a reference one. Then the script upsamples the other data streams based on this reference timestamp vector, using *interp1* function in a "sample and hold" manner (using the parameter *previous*).

■ Transformation

The most important part of the T265 data post-processing is the coordinate frames transformation. The 6DoF pose data are expressed w.r.t. a measurement reference frame. The measurement reference frame for T265 is created at the beginning of a measurement and is dependent on the initial camera position and orientation. The origin of the frame is aligned with the origin of T265 pose frame (depicted in Fig. 5.6). The y-axis of the reference frame is antiparallel to gravity vector pointing towards the sky. The x- and z-axis are determined based on the camera orientation, forming a right-handed system, but are not directly aligned with the axis of T265 pose coordinate (body) frame [60].

The creation of the measurement reference (world) frame is in general deterministic, but hard to reconstruct and also depends on the camera initial orientation. In case the body y-axis initially points upwards (the camera lenses point forward), the world frame is almost exactly aligned with the initial body frame. In case it is the body z-axis that is initially aligned with gravity vector (the camera lenses upward or downward), the world frame is initialized arbitrarily and the measured orientation data are confused and not reliable. Therefore, it is recommended to not to initialize the sensor pointing upwards or downwards [64], which can be limiting for many applications. This issue has to be remembered when designing the jig for sensor initial positioning before measurement.

The way the world frame is initialized makes it unsuitable for our tracking application,

because the world frame cannot be reconstructed. The T265 pose data are expressed w.r.t. this world frame and therefore need to be transformed to another reference coordinate frame, that is stable and easy to reconstruct based on the sensor initial position and orientation. I chose to express the data w.r.t. the initial body coordinate frame, consistently with the IMU sensor data. Designing the transformation calculations, I followed the theory described in [33].

First, I calculate the transformation matrix that transforms the world frame into the initial body frame (global). In this case I have access to not only orientation but also position data, and the transformation matrix can be full, including the translation part. The matrix of directional cosines $\mathbf{S}_{\mathbf{wg}}$ is formed using the initial quaternion data q_w according to (5.1) and extended by the initial position vector $\mathbf{r}_w = [x_w \ y_w \ z_w]^T$ to result in the transformation matrix $\mathbf{T}_{\mathbf{wg}}$:

$$\mathbf{T}_{\mathbf{wg}} = \begin{bmatrix} \mathbf{S}_{\mathbf{wg}} & \mathbf{r}_w \\ 0 \ 0 \ 0 & 1 \end{bmatrix}. \quad (5.26)$$

Note that this way it is possible to choose a different then initial body frame as the reference one. It can be helpful in cases when it is desired to have the global frame aligned with the camera frame when pointing upwards or downwards. The user is being forced to initialize the camera facing forward, but can later move it to the desired upward/downward position and set the global frame to be aligned with this later one.

Further, all position data \mathbf{r}_w are transformed from the world to the global frame using the inverse of $\mathbf{T}_{\mathbf{wg}}$:

$$\mathbf{r}_g^m = (\mathbf{T}_{\mathbf{wg}})^{-1} \mathbf{r}_w^m, \quad (5.27)$$

where the superscript m denotes the measurement (data entry) index.

Next, the quaternions have to be transformed from the world frame to the global frame. With that, orientation of the local frame expressed in the global frame will be obtained. To do so, I form the transformation matrix $\mathbf{T}_{\mathbf{wl}}^m$ for each time step using quaternions q_w^m and position vectors \mathbf{r}_w^m according to (5.26). The desired matrix $\mathbf{T}_{\mathbf{gl}}^m$ can be expressed based on the transformation rules as follows:

$$\mathbf{T}_{\mathbf{gl}}^m = (\mathbf{T}_{\mathbf{wg}})^{-1} \mathbf{T}_{\mathbf{wl}}^m. \quad (5.28)$$

The part of the matrix $\mathbf{T}_{\mathbf{gl}}^m$ that represents the matrix of directional cosines as indicated in (5.26) is then used to calculate the quaternions w.r.t. the global frame q_g^m according to (5.3) and (5.4).

■ 5.6.5 Measurement scenarios

I conducted several experiments with the Intel RealSense tracking camera T265 in order to examine its performance and suitability for the desired motion tracking task. Since there was no simulation data available, I prepared such scenarios that can be realized with the constructed measurement device described in more detail in section 5.5.6 and shown in Fig. 5.5. I was interested in steady state measurements, because of the possibility to inspect the sensor's noise characteristics. Further, I tested the camera measuring typical motions such as translation in one axis and harmonic motion in one axis.

The experiments were done in a similar manner as with the MMR IMU sensor, using the slider-crank mechanism. The camera can be mounted on the device so, that the movement is then realized along one of its body axes and can be rotated so, that movements in all three axes are examined. At this moment, I faced the problem of arbitrarily initialized reference world frame. When the camera was mounted facing upwards, I had to first rotate the whole device, initialize the measurement, and then manipulate it back to get the camera to the desired upward position. The global frame was then calculated not as the initial body frame but as a later already upward facing one. When performing the harmonic motion, slow and fast speed was tested.

I was able to mount the T265 camera on the measurement device together with the MMR IMU sensor. I conducted experiments with both sensors mounted with their body axis aligned and was therefore able to compare their performance directly.

■ 5.6.6 Results

I conducted several experiments described in 5.6.5 with the slider-crank device, post-processed measured data and visualized them using Matlab *realsense_data_post.m* script. Based on the observed results I drew several conclusions presented in this section. The conclusions are supported by visualizations showed in Fig. A.59 - A.65 in the appendix A.3.

In the presented figures, I use RGB-xyz convention to indicate coordinate frames. There was no filtration applied to the measured data, the position plots therefore show only the original signal.

■ Conclusions from experiments

- When the camera's initial position is facing straight up or down, the orientation measurements are unreliable. It is necessary to tilt the camera slightly before initialization and put it on desired initial place later. Afterwards, the chosen global frame can be one

of the later ones already in the desired position, instead of the very first one (Fig. A.62 - A.65).

- The reference coordinate frame is not aligned with geographical or initial frame and coordinate transformation is therefore essential (Fig. A.59).
- The measured steady state data are very precise with absolutely inconsiderable bias (Fig. A.60 and A.61).
- Although a drift can appear due to a varying bias, it can be compensated during a following steady part of the motion (Fig. A.62).
- Fast swinging motion can lead to drift in position (Fig. A.65).
- The results are influenced by the nature of the scene seen by cameras. The position estimates are better when there are more objects in middle range distance in the scene.
- The sensor fusion software included in the camera registers warnings when the speed of the motion is too high. Despite that, the position estimates work well (Fig. A.63, A.65).
- No further filtration of data is needed.
- There is no observable difference in performance when compared between measurements of movements aligned with x-, y- or z-axis.

■ 5.7 Comparison

The slider-crank device allows to mount MMR IMU sensor and RealSense T265 tracking camera together, therefore I was able to perform several measurements for direct comparison of performance of both sensors. The data were post-processed and visualized using Matlab in the same manner as by individual sensors. Based on the observed results and results of individual sensors measurements showed in section 5.5.7 and 5.6.6 I drew several conclusions presented in this section. The conclusions are supported by visualizations showed in Fig. A.66 - A.71 in the appendix A.4.

■ Conclusions from experiments

- The MMR IMU sensor is not suitable for measurements of translation compared to the camera, that depicts the motion very well. The position data from IMU show trends in motion, but fail to describe the trajectory precisely (Fig. A.66, A.67).

Tracking method	Technique	Accuracy level	Environment resistance	Ease of set-up	Cost
Lase trackers	optical	high	medium	low	high
IMU	inertial	low	high	high	low
IMU with camera	inertial and optical	high	low	high	low
IMU with light pulses	inertial and optical	medium	medium	medium	medium
IMU with ultrasound sensor	inertial and acoustic	high	high	medium	medium
Active beacons	Bluetooth or ultra-wideband	low	high	low	medium
GNSS	radio	low	high	low	low

Table 5.5: Comparison of presented motion tracking techniques with respect to the use case of tracking of a hand-held spray gun.

- A harmonic motion can be depicted by both sensors. Both can register the same frequency in motion and mostly even the same amplitude. However, due to the high-pass filtering of IMU data that tends to shift data to have zero mean, the IMU position data are distorted and can not be trusted (Fig. A.68 - A.71).
- There is no observable difference in performance when compared between measurements of movements aligned with x-, y- or z-axis.

■ General Conclusions

- The IMU sensor only is not sufficient source of data for 6DoF motion tracking (position and orientation).
- The combination of IMU and camera using visual and inertial sensor fusion works well and is suitable for the 6DoF motion tracking task.

The knowledge of motion tracking techniques, available technologies, sensors and algorithms, gained in this chapter, can be finally summed up. Many techniques have been studied and some of them tested in experiments with real hardware. The analysis of gained experience and results was done with respect to the intended use case, a motion tracking of a hand-held spray gun in an industrial painting workplace. Concluded comparison is shown in Tab. 5.5.

Assuming all four performance categories in Tab. 5.5 being of the same importance, the overall score of each tracking method can be calculated. Combinations of IMU and camera or IMU and an ultrasound sensor are evaluated as the best.

Chapter 6

Robot control program synthesis

Chapter 5 was dedicated to the sensory system and the role of sensor operator and painter in the process of automation of industrial painting. The output of that part of the process is a trajectory of spraying tool expressed in Cartesian coordinate system w.r.t a known coordinate frame. In this chapter role of remaining elements of the process, robot and robot operator, is covered.

The goal of this part is to generate a control program for the robot based on received spray gun trajectory data. Fulfilment of the task consists of data transformation to a correct coordinate frame (section 6.1), solving inverse kinematic problem to get values of joint variables of the robot required to perform the trajectory (section 6.2) and based on them generating of control commands (section 6.3). Details of this procedure depend on the characteristics of used robot and configuration of the painting cell. These are variable, some possible examples are presented in chapter 8. I provide a general description of robot control program synthesis, not designed to fit any particular robot or company.

One essential assumption I made is, that the working range of chosen robot in the painting cell is sufficient for completing the painting task, i.e. positions of all points in the trajectory data are feasible.

6.1 Transformation

The trajectory data need to be transformed from the measurement reference coordinate frame to the robot reference coordinate frame. The robot, painted object and measurement reference coordinate frames are fixed and their mutual location is known and can be expressed in form of transformation matrices, as described in section 5.2. The matrix that transforms data from measurement reference frame to robot reference frame will be called \mathbf{T}_{RM} .

The trajectory data capture motion of a point defined by the motion sensor, e.g. the center of its accelerometer, that is fixed to the painting tool. The transformation between this

point and a chosen reference point on the tool, for example the tip of the spray gun, is again known and will be called $\mathbf{T}_{\mathbf{MT}}$. Since the goal is to reproduce trajectory of the painting tool, the resultant transformation matrix $\mathbf{T}_{\mathbf{RT}}$ used to express position and orientation of the tip of painting tool w.r.t. the robot reference coordinate frame is:

$$\mathbf{T}_{\mathbf{RT}} = \mathbf{T}_{\mathbf{RM}}\mathbf{T}_{\mathbf{MT}}. \quad (6.1)$$

The painting tool attached to the robot as an end-effector is a part of a known kinematic chain that starts in the robot reference coordinate frame and describes robot's kinematics. The position and orientation of the tool expressed w.r.t. the robot frame represents therefore desired end-effector trajectory and is the input to the inverse kinematic problem.

6.2 Inverse kinematic problem

An industrial robot consists of several links connected together by joints. A type of robot that is considered in this chapter is a serial stationary robot with 6 degrees of freedom corresponding to its 6 revolute joints. The state of the robot is described by rotations of individual joints, the joint variables, angles Θ . Knowing the state of the robot and the robot's geometry and dimensions, position and orientation of its end-effector is unique and can be determined. Determination of end-effector pose from joint variables is called forward kinematic problem. Its solution requires composition of elementary transformation matrices between local coordinate frames of connected links and results in final transformation matrix:

$$\mathbf{T}_{\mathbf{06}}(\Theta) = \mathbf{T}_{\mathbf{01}}(\Theta_1)\mathbf{T}_{\mathbf{12}}(\Theta_2) \dots \mathbf{T}_{\mathbf{56}}(\Theta_6), \quad (6.2)$$

that is used to calculate end-effector position $\mathbf{r}_{\mathbf{R}}^{\text{ee}}$ w.r.t. the robot reference frame:

$$\mathbf{r}_{\mathbf{R}}^{\text{ee}} = \mathbf{T}_{\mathbf{06}}(\Theta)\mathbf{r}_{\mathbf{6}}^{\text{ee}}. \quad (6.3)$$

The value of $\mathbf{r}_{\mathbf{6}}^{\text{ee}}$ is constant and known from robot geometry. The orientation of end-effector is given by the directional vector between the center of last link's local coordinate frame and the end-effector's local frame [65].

Finding joint variables when end-effector pose is given is reversed task called inverse kinematic problem. The inverse problem is usually more complicated than the forward one. Solution may not always exist but there may be also multiple solutions for one end-effector pose. Solving inverse kinematic problem means finding elements of matrix $\mathbf{T}_{\mathbf{06}}(\Theta)$ such that equation (6.3) holds and then eliminating values of individual joint variables (Θ) from them, which leads to solving of set of 16, usually nonlinear, equations.

In practice both analytical (closed form) and numerical ways of finding solution to inverse kinematic problems exist. The closed form solution is preferred, because calculating of joint variables is then generally faster and it allows for design of a reasonable mechanism for choosing one among more possibly existing solutions. Many analytical solvers are however limited to robots with maximum of 6 DoF and are based on kinematics decoupling of translation and rotation [65].

Many examples of implemented numerical algorithms are based on Inverse kinematic library by Samuel Buss [66], including widely used robotics simulation module PyBullet [67]. An example of solver performing analytical solution is IKFast algorithm by OpenRAVE framework [68]. Commercial robot control software packages are also equipped by some form of inverse kinematic solver that is optimized for the particular robot.

■ Numerical solution - Damped least squares

Numerical solution is based on approximation of relationship (6.3). This relationship in general says, that end-effector position \mathbf{r}_R^{ee} is a function of joint variables Θ , such that $\mathbf{r}_R^{ee} = \mathbf{r}_R^{ee}(\Theta)$. The following derivation refers to [69]. Jacobian matrix $\mathbf{J}(\Theta)$ can be expressed as follows:

$$\mathbf{J} = \frac{\partial(\mathbf{r}_R^{ee})}{\partial(\Theta_j)}, \quad (6.4)$$

where $j = 1 \dots M$ is an index of joint variable for an M-DoF robot.

Expression (6.4) leads to forward dynamics equation:

$$\dot{\mathbf{r}}_R^{ee} = \mathbf{J}\dot{\Theta} \quad (6.5)$$

and its approximation by:

$$\Delta \mathbf{r}_R^{ee} \approx \mathbf{J}\Delta\Theta. \quad (6.6)$$

The aim of inverse kinematics is to find such joint variables Θ , that the end-effector position \mathbf{r}_R^{ee} equals target position \mathbf{t}_R . The change in end-effector position needed to achieve the target position can be expressed as:

$$\mathbf{e} = \mathbf{t}_R - \mathbf{r}_R^{ee} \quad (6.7)$$

and approximated by:

$$\mathbf{e} = \Delta \mathbf{r}_R^{ee} \approx \mathbf{J}\Delta\Theta. \quad (6.8)$$

The problem is therefore formulated as a problem of choosing values of $\Delta\Theta$ iteratively until equation (6.8) is satisfied. One possible way of doing so is called damped least squares or Levenberg-Marquardt method. It finds the value of $\Delta\Theta$ that minimizes:

$$\|\mathbf{J}\Delta\Theta - \mathbf{e}\|^2 + \lambda^2\|\Delta\Theta\|^2, \quad (6.9)$$

where λ is a chosen non-zero damping constant.

Equation (6.9) can be rewritten as:

$$(\mathbf{J}^T \mathbf{J} + \lambda^2 \mathbf{I}) \Delta \Theta = \mathbf{J}^T \mathbf{e}, \quad (6.10)$$

leading to damped least square method solution for $\Delta \Theta$:

$$\Delta \Theta = (\mathbf{J}^T \mathbf{J} + \lambda^2 \mathbf{I})^{-1} \mathbf{J}^T \mathbf{e}. \quad (6.11)$$

■ Analytical solution - IKFast

Closed form solution refers again to equation of forward kinematics. The following derivation refers to [70]. From the relationship (6.3) set of equations is obtained, where each equation is of a polynomial form:

$$\sum_i a_i \prod_j \cos_j^{p_{i,j}} \sin_j^{q_{i,j}} \Theta_i^{r_{i,j}} = 0, \quad (6.12)$$

where a is a constant coefficient and p , q , and r are exponents and $p_{i,j} + q_{i,j} + r_{i,j} \leq 1$. Further, trigonometric constraints $\cos_j^2 + \sin_j^2 - 1 = 0$ must not be violated.

IKFast implements algorithm of searching for roots of polynomial equations obtained from forward kinematics. The algorithm evaluates solution and numerical complexity and starts solving those joint variables that are the simplest and most unambiguous first. It can find all possible discrete solutions and can handle even robots with more than 6 DoF by methodically selecting those joints that are above 6 as free joints [70].

■ 6.3 Control commands generation

A robot control system takes the outputs from inverse kinematics and forms the final sequence of commands that can be sent to robot's actuators. The system has to choose the most optimal joint poses, when there are redundant solutions available for one end-effector pose. Aspects to be optimized for are for instance amount of change in joint variables compared to previous pose or energy needed to set these joint variables.

The control system is also responsible for performing collision checks and adjusting the joint poses so, that collisions are avoided. Further, it incorporates joint range, velocity and acceleration limits and ensures that the sequence of required joint poses is feasible. It also controls the speed of motion, which can be either prescribed by user or optimized automatically.

The values of joint variables from the resultant sequence are successively sent as inputs to sophisticated control algorithm that performs feedback control of corresponding actuators of each joint. The joints are set and the robot performs required trajectory [65].

The process of commands generation is automated by the robot control software. However, a human intervention is needed in some cases. Some parts of the trajectory might reveal to be not optimal or not feasible but may be adjusted to become at least suboptimal. Alternatively, it may be necessary to divide the trajectory into shorter sections, when it is not possible to perform it in one continuous movement. The role of the robot operator is to review generated robot trajectory in real or in simulation and adapt it as needed based on the operator's experience and knowledge of performed task.

■ 6.4 Simulation

For the purpose of the simulation, I recorded several sets of data with the Intel RealSense tracking camera T265 (described in section 5.6) held by hand. I performed random movements imitating those possibly done during industrial painting.

There are many manufacturers of industrial robots and each of them provides own control software for their robots. Their functionality is similar but implementation differs. I did not use any commercial software in order to keep this work general and avoid details about software that may be for most applications useless. Instead, I prepared simulation of robot control program synthesis and execution in two open-source programs.

First software called PyBullet [67] is a tool often used by researchers in field of robotics and machine learning. It includes numerical inverse kinematic solver, simulates and visualizes robot control. Second software is Robot operating system (ROS) [71] that includes a number of libraries and tools for robot control, simulation and visualization and enables also commands execution on real robots and real hardware.

■ 6.4.1 PyBullet

PyBullet [67] is a python implementation of simulator for robotics simulations and machine learning focused on sim-to-real transfer. It even contains implementation of several industrial robots that can be easily included in user's application.

I chose to use a model of industrial robot KUKA LBR iiwa. This robot has 7 DoF, its working range is relatively small but thanks to its light weight and small footprint it can be mounted on a carriage and become mobile. Besides, the robot is collaborative and allows teaching by demonstration, meaning that the operator can easily guide the robot through desired trajectory. It can be helpful when a part of trajectory is not feasible or optimal and needs to be adjusted. It is also suitable for painting operations [72]. The Pybullet model includes information about its joint range, velocity and acceleration limits and rest poses.

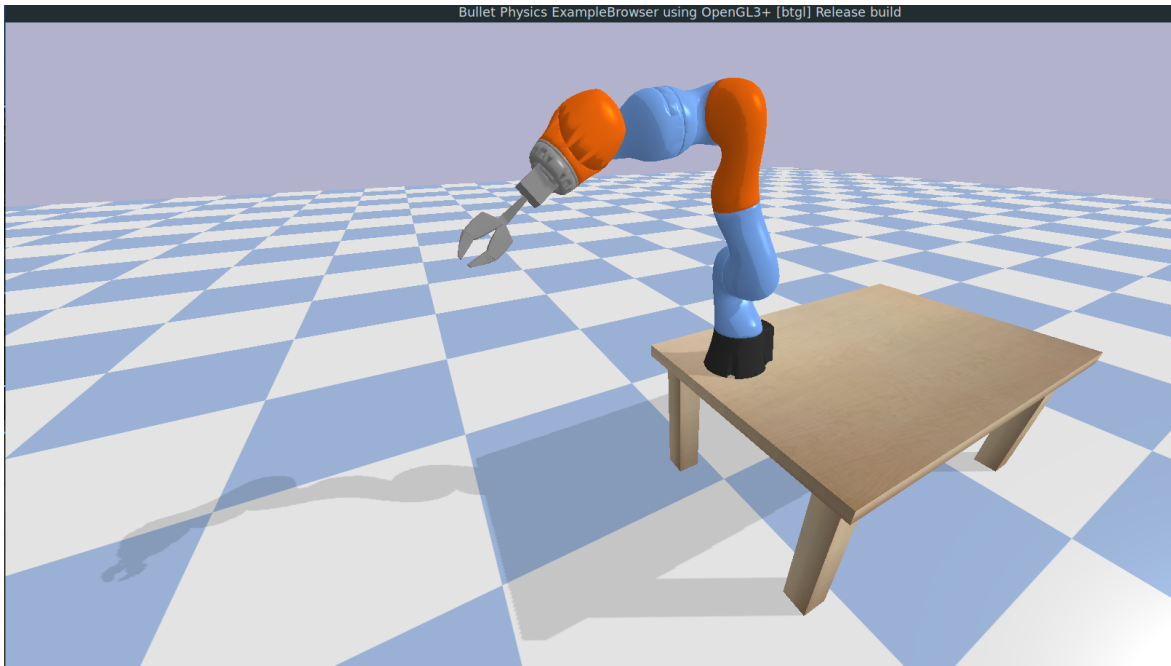


Figure 6.1: Simulation of KUKA LBR iiwa in PyBullet.

All source code including python dependencies in a `.yaml` file needed for the PyBullet simulation is to be found in the appendix C.4. The main script `pybullet_kuka_simulation.py` loads trajectory data from provided csv file and transforms them to the robot reference coordinate frame. It creates an instance of class `KukaGymEnv` and sends the trajectory data to it in a for loop. The `KukaGymEnv` class defined in `KukaGymEnv.py` is a custom definition of simulation set-up. It initializes instance of `Kuka` class and includes basic methods to control it. The `Kuka` class (`kuka.py`) contains necessary information about the real robot.

One of the Kuka methods I implemented is `apply_accurate_IK` that solves inverse kinematics by numerical method of damped least squares described in section 6.2. It allows user to set tolerances on achieved end-effector position and orientation and upper limit of iterations. The algorithm tries to find solution iteratively, until the pose is within the tolerances or maximum iteration is achieved. The velocity is indirectly controlled by specifying a duration of each simulation step (one loop), that corresponds to the sampling time of recorded trajectory data. I also used null-space parameter to make the algorithm to take into account joint limits and optimized towards robot's rest poses.

The robot executes a movement each simulation step, even when the desired tolerance was violated, and eventually evaluates error of each step in terms of difference between desired and achieved position and orientation of end-effector.

6.4.2 ROS - Descartes

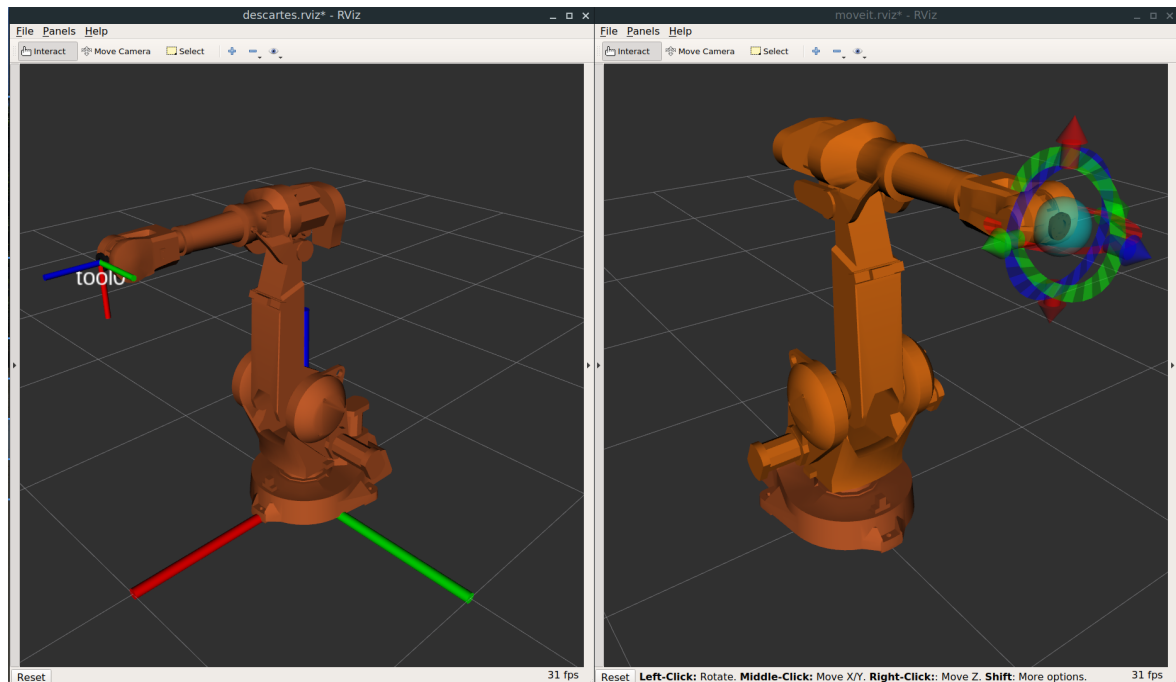


Figure 6.2: Simulation of ABB IRB 2400 in ROS.

Descartes [73] is a package introduced by industrial consortium called ROS-Industrial that extends capabilities of often used platform MoveIt [74]. It is meant for robotic trajectory planning (joint variables) based on Cartesian trajectory data (end-effector position and orientation) given as a function of time.

The path planner available in MoveIt is more suitable for standard trajectory planning tasks, when user specifies only several waypoints and lets robot interpolate between them as needed. In our application we prescribe sequence of many waypoints, the trajectory discretization is therefore very dense, and we demand visiting next waypoint each time step, that is shorter than one second. For these reasons, I decided to use Descartes. Another useful feature of this package is option to plan for only 5DoF trajectories, meaning that it allows the robot to rotate around the longitudinal axis of the end-effector freely, which makes the planning more flexible and is suitable for the case of industrial painting with a spray gun. Descartes can also avoid collisions with the robots itself and other specified objects in the environment [73].

I followed Descartes tutorials [75] and prepared *robo_painting.cpp* script written in C++ that loads csv file with end-effector position and orientation data, transforms them to the reference coordinate frame of the robot, calculates the inverse kinematics, extracts final

sequence of joint variables, simulates its execution and sends the respective commands to real robot if available. The source code is to be found in appendix C.5.

The robot used in this simulation is IRB 2400 by ABB, a commonly used 6DoF robot with large working range that is also available with increased environment protection [76]. The model of the robot and data about its kinematics, joint range, velocity and acceleration limits are included in the ABB MoveIt package, that is also maintained by ROS-Industrial.

The inverse kinematics is calculated using IKFast algorithm (section 6.2) implemented in the Descartes package. The solutions for the whole sequence of end-effector pose data are arranged in a graph. In case there are points for which no solution exists, the algorithm returns error and IDs of these points. In case one or multiple solutions exists for each point, the solutions are represented by nodes in the graph [77].

The algorithm then tries to connect nodes of two consecutive points by edges. Two nodes can be connected only if the transition between them is feasible in time given by the time step with respect to the joint limits. If there are two consecutive points for which no pair of nodes exists that would be connected with an edge, the algorithm returns error and IDs of such points [77].

Lastly, the algorithm performs Dijkstra's graph search algorithm to find the sequence of nodes (joint variables) while optimizing a cost function. The cost function is a sum of costs of edges in the sequence. An edge cost is defined as a norm of the difference between joint variables (nodes) that are connected by the edge. In other words, the algorithm enforces smooth robot trajectories [77].

The robot trajectory defined by found sequence of joint variables is then performed by the robot simulation and may be executed by real robot if available.

6.5 Conclusions from simulations

I conducted several simulations using both PyBullet and ROS - Descartes. The aim of the simulations was to proof if the trajectory of a hand-held tool sampled by a sensor can be transformed into robotic trajectory and executed. With respect to this aim, I made simulations with two different robots using two different kinds of software and tested both on the same recorded data, to enhance general validity of achieved results.

I was able to transfer sets of relatively complex recorded trajectories into both simulated robots and proofed that the concept of trajectory planning introduced in this thesis is meaningful. The results in form of graphs and videos are available in the appendix B. In addition to that, I derived several conclusions, which I present in this section.

- The trajectory data recorded by Intel RealSense tracking camera T265 do not need

any additional filtering, the robot movement generated based on them is smooth and reasonable.

- It is very important to choose configuration of the robot w.r.t. the painted object carefully, so that the recorded trajectory transformed to the robot coordinate frame is for the robot feasible. A small change in configuration may change the feasibility.
- There may be parts of trajectory that are not feasible by a standard industrial robot. A human arm has more DoF and is capable of poses that robot with 6 or 7 DoF may not be able to achieve.
- Relaxation of 6DoF to 5DoF planning with free rotation around tool's longitudinal axis helps to find more feasible solutions of inverse kinematic problem.
- PyBullet simulation with KUKA iiwa using numerical inverse kinematic solver was more robust and able to imitate more complex trajectories than the one done using Descartes.
- In Descartes there are situations when a continuous rotational movement is interrupted, the last link rotates by almost 360° in the opposite direction to the movement and then continues the movement further. This movement is not achievable within the given time limit and the algorithm returns error. I believe it is because the not optimally defined orientation by Euler angles that is discontinuous.

Chapter 7

Software project design

The aim of this work is to take the problem of automation of industrial painting process as a complex task and provide a solution for it as a whole. A solution of such complex task includes several hardware elements such as sensors, that further need a software for control. The solution is required to provide multiple functionalities, be generalized enough and complete in terms of coverage of the whole automation process. An indispensable part of such solution is naturally a software program that frames all mentioned features into one arranged, easy to use package, providing the necessary support for completing the painting automation task from the beginning till the end.

I approached this need and created a software study that precisely determines all wanted and needed features and functionalities. The study represents a framework that brings together tools designed in chapter 5 and 6 of this work. I worked out most of the parts of this software study, however, the attention is still paid mainly to the engineering challenges and solving of practical problems. Therefore, the software is a working version for engineers and developers rather than for general public, but should be later upgraded to a robust and user friendly software based on the study presented in this chapter. I call the designed software Robo-Painter.

7.1 Introduction

The Robo-Painter software is a tool that helps with automation of industrial painting tasks. The software provides user with a possibility to record trajectory of a hand-held painting tool using a sensory system and transform it into robot control commands. The software post-processes the recorded data, based on user defined configuration and generates input data for a robot control system. It also allows user to visualize and optimize generated data. The robot control system transforms input data into commands and creates robot control program. The robot running such control program from the Robo-Painter software

will imitate the recorded painting trajectory.

The main functionalities of the software are trajectory data collection and visualization, post-processing, optimization of the trajectory and generation of commands for the robot control program.

7.2 Software specification

The software should be used in the workplace where industrial painting is performed. The Robo-Painter is designed as a supporting software for workers (painters), automation engineers and robot programmers. It allows them to cooperate on design of a solution for automation of a selected painting operation. Each process of automation of a particular painting operation is well documented and all respective data are stored under a project in the database.

The Robo-Painter software consists of two main parts – human painter data collection with analysis, and robotic painting data synthesis and generation. Both parts are further divided into subsections and these can be performed separately at different times, by different users. Each subsection represents a certain operation over data. It takes data from the database as an input, modifies them in a defined way and stores them back into the database.

The database is an essential element of the whole system. The basic organization unit in the database is a project. One project represents one attempt to automatize the painting of one product. The project is uniquely described by the user defined name and the database keeps the information about when it was created and modified. A project stores data about the product in the main section. These are part identification number and the painting process details such as the catalogue number of the used paint, the thickness of the painted layer, required time of drying. Further, it stores all recorded tool trajectory data into the section raw data under a unique identifier based on the start time of the measurement. When the data are further processed, the database creates new section and stores the modified data in there together with the information about the configuration of the processing. The sections in the database correspond to the functional subsections of the system.

The functional subsections of the system are: tool trajectory data collection, tool trajectory data post-processing, robot trajectory synthesis, robot trajectory optimization, robot control. Further, the software offers functions such as data visualization and trajectory simulation, and report generation from the robot control system. The post-processing involves data synchronization, filtration, cropping and optional mathematical operations such as double integration. The optimization involves data comparison, averaging, copy, paste and replace. The robot control involves robot control commands generation, robot program generation

and loading.

The software is meant to be locally installed on regular personal computers of involved users (automation engineers, robot programmers, managers. . .) with an access to the common database located either on a shared hard drive or a cloud server, based on the user's choice. The amount of copies of the software installed locally is not limited and the system does not expect any restricted access in form of a login or user registration. The access rights to the database are in the competence of the user and are not enforced by the system. Each user can work with all functions of the software.

Further, the Robo-Painter software exists in a minimal install version with capabilities limited to data recording and storing only. This version should be installed on a simple small single-board computer and should be used by the worker while painting a product and recording the painting tool trajectory. In this scenario, the regular PC would not be suitable because of its size, weight and mainly the risk of being destroyed by the aerosol in the environment. Also, the painter is supposed not to interact with the software further beyond the stage of the trajectory recording, therefore this minimal version should optimally cover painter's use case.

■ 7.2.1 User roles, agents and actions

The software does not recognize different types of users and all users can therefore access all functionalities. The software further does not expect any restricted access in form of a login or user registration. However, different types of use cases can be distinguished, corresponding to different types of user roles. For this reason, I differentiate between the user roles from the point of view of the software and its usage, rather than based on real people or their job titles. Most likely, one person will occupy more user roles alongside. Furthermore, the software interacts with a sensory system and a robot.

The overview of user roles and their relationship to the system is showed in Fig. 7.1.

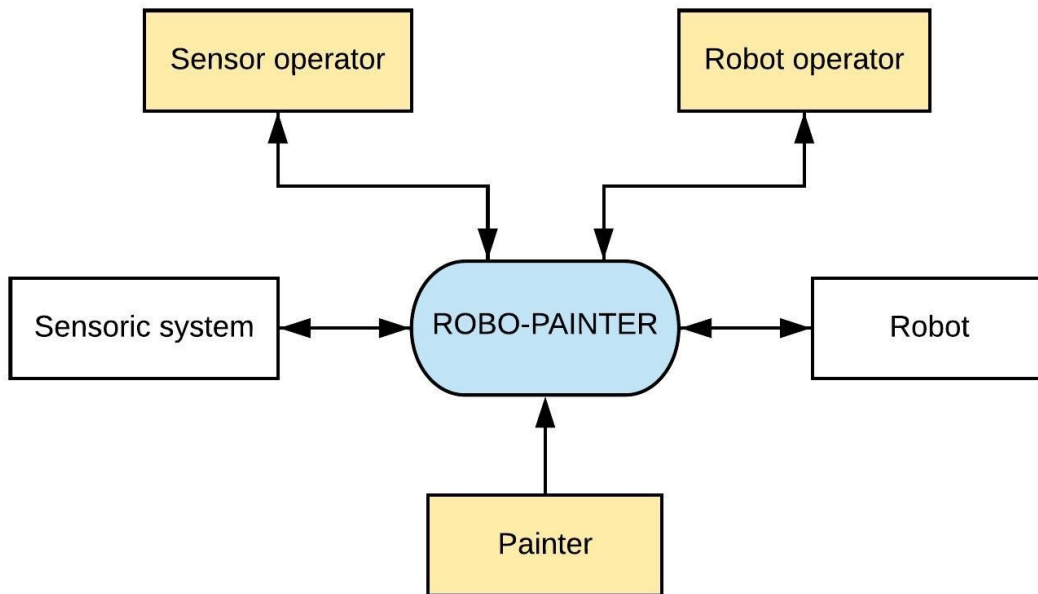


Figure 7.1: Context model of the Robo-Painter software.

■ Painter

The painter is a worker who uses the minimal install version of the software. He or she is involved in the subsection tool trajectory data collection only. The painter can start the measurement and end the measurement (by sending commands to the sensory system). He or she can also set up a new project in the database.

■ Sensor operator

The sensor operator is involved in the subsections tool trajectory data collection, tool trajectory data post-processing and robot trajectory synthesis. The sensor operator can set up a new project in the database. He or she can create a measurement configuration file and save it into the database, or view an existing configuration file. The sensor operator can visualize the raw data stored in the database. Further, he or she can create a post-processing configuration file and save it into the database or view an existing one. The sensor operator can post-process the raw data based on the configuration, visualize them, and save them into the database or delete them. After the post-processing, the sensor operator can ask the software to synthesize the robot trajectory and store it into the database.

■ Robot operator

Functional subsections that belong to robot operator are robot trajectory optimization and robot control. The robot operator creates the optimization configuration file or views an existing one, loads the robot trajectory data, and optimize them based on it. He or she can then visualize the trajectory data, store them or delete them. The robot operator can ask the software to generate robot control commands, view the simulation of robotic painting process and load the control program into the robot. He or she can also view the reports generated by the robot and stored in the database.

■ Sensory system

The sensory system receives commands from the Robo-Painter system. It collects the tool trajectory data according to received configuration file for the period of time controlled by the configuration file. It stores the raw trajectory data into the database.

■ Robot

The robot receives commands from the robot control system in form of a program and performs them. It sends the system report that is stored into the database.

■ 7.2.2 Hardware and software requirements

The full version of the Robo-Painter software is meant to be installed on personal computers, while an average performance level should suffice. The operation system needs to be enough flexible to handle interaction with the sensory system, computing system, robot control system and the robot. The software is not responsible for any time or mission critical tasks.

The computing and robot control systems need a support of third-party software, that will be integrated into the Robo-Painter software.

The minimal install version of the software should be installed on a single-board computer with a wireless connection module, memory card slot, external power supply and operation system identical with the one for the full version.

The database should be stored on a shared hard drive or a cloud service. The expected amount of stored data is approximately 0.5 GB per project.

Name	Type	Description
project_name	string	Unique name of project, user defined
product_id	unsigned short int	Product identification number, match the user's system
paint_id	unsigned short int	Paint catalogue number
layer_thickness	unsigned short int	Thickness of layer to be painted on the product, in micrometers
drying_time	unsigned short int	Time needed for the painted product to dry, in seconds
data_raw_traj	array	Tool trajectory data recorded by the sensory system
data_post_traj	array	Tool trajectory data post-processed by sensor operator
data_robot_traj	array	Robot trajectory generated by the sensor operator
data_opt_robot_traj	array	Optimized robot trajectory by the robot operator
data_robot_command	array	Commands for the robot generated by the robot operator
robot_program	object	The control program loaded into the robot
config_meas	record	Configuration file used for tool trajectory data recording
config_post	record	Configuration file used for the post-processing
config_opt	record	Configuration file used for the optimization
report_robot	record	Report file returned by the robot

Table 7.1: Data glossary of the Robo-Painter software.

7.3 Software analysis

To identify and describe important aspects of the software an analysis in form of a detailed context model (Fig. 7.2) with data glossary (Tab. 7.1), entity relationship diagram (Fig. 7.3), data flow diagram (Fig. 7.4 - 7.6) and state transition diagram (Fig. 7.7 - Fig. 7.9) was made.

The entity relationship diagram ERD (Fig. 7.3) represents the database structure, that resembles the functional subsections of the software.

The dataflow diagram DFD (Fig. 7.4 - 7.6) is depicted at different levels of detail. Level 0 (Fig. 7.4) shows the system overview, understanding each functional subsection of the software as a system process. The system processes are further analyzed at level 1 (Fig. 7.5) and extended by the additional processes at level 2 (Fig. 7.6).

The transitions between states in the STD State transition diagram (Fig. 7.7 - 7.9) represent mostly commands coming from user interaction with the graphical user interface (GUI). Design of the GUI is beyond the scope of this work, however, the STD of this form represents a draft of the GUI, listing most of the input controls and navigational components. For clarity, the software STD is divided into sections.

Since the GUI design is not part of this work, the user commands, such as set up new project, measure, post-process, are not included in the data glossary (Tab. 7.1).

7.4 Software testing

From the software analysis it is obvious that the Robo-Painter software can be decomposed into functional subsections. Each of these subsections – measurement, post-processing, optimization, robot control, will be designed and developed separately and should be also tested separately using dummy test data. When the subsections are ready, the communication with agents outside of the system boundary is implemented and tested. The sections are tested again on a real test data. After the connection between the software subsections is established, the data flow through the whole system is tested.

1. Subsection tests with dummy test data
 - a. Measurement section
 - (i) Measurement
 - (ii) Visualization
 - b. Post-processing section
 - (i) Post-processing
 - (ii) Robot trajectory generation
 - (iii) Visualization
 - c. Optimization section
 - (i) Optimization
 - (ii) Robot commands generation
 - (iii) Visualization
 - d. Robot control section
 - (i) Robot simulation
 - (ii) Robot program generation
 - (iii) Robot reports
2. Interaction tests with agents and users
 - a. Sensoric system/measurement section interaction
 - b. Robot/robot control section interaction
 - c. User/database interaction
3. Integration tests with real data
 - a. Subsection tests (as in 1.)
 - b. Complete data flow test

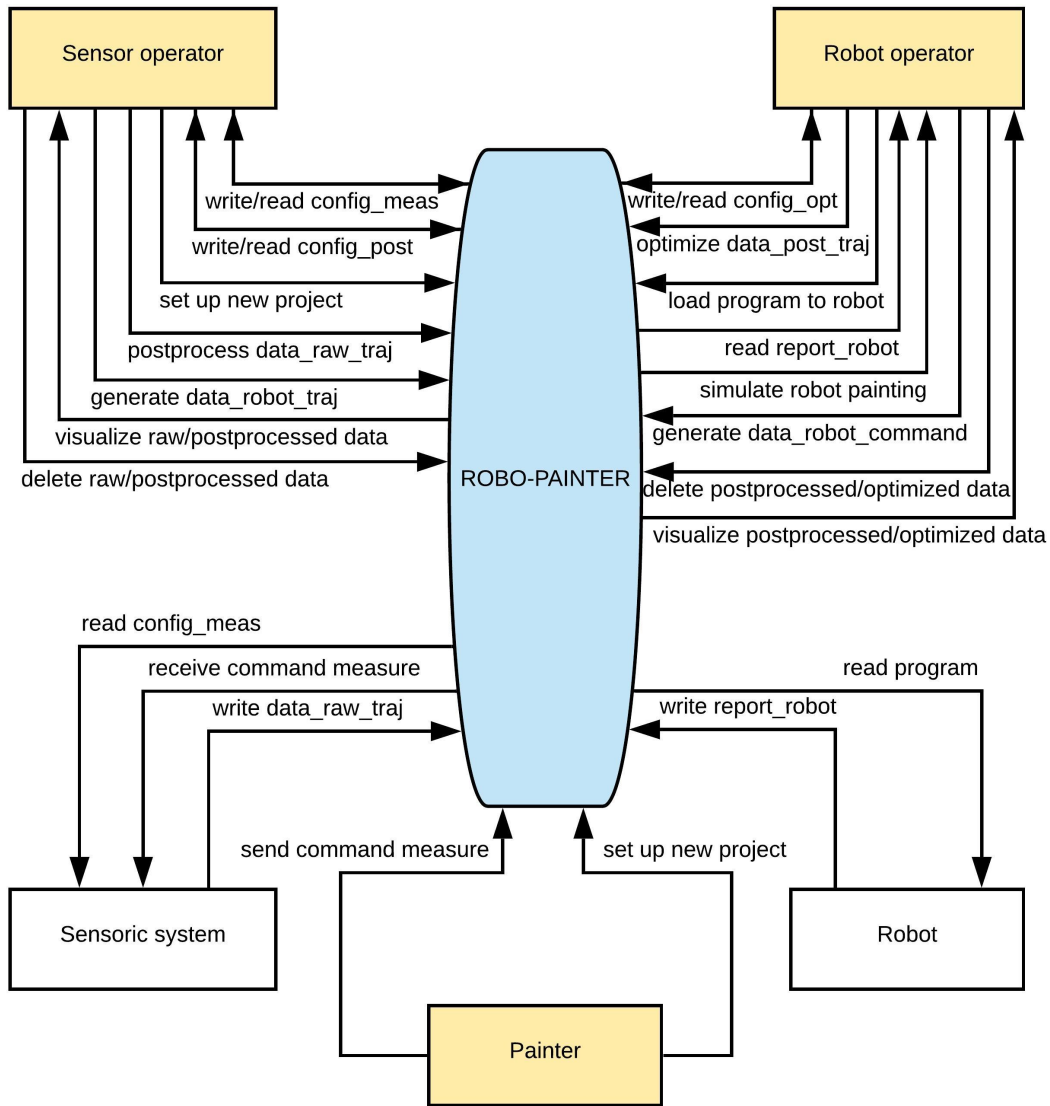


Figure 7.2: Context model of the Robo-Painter software, detail.

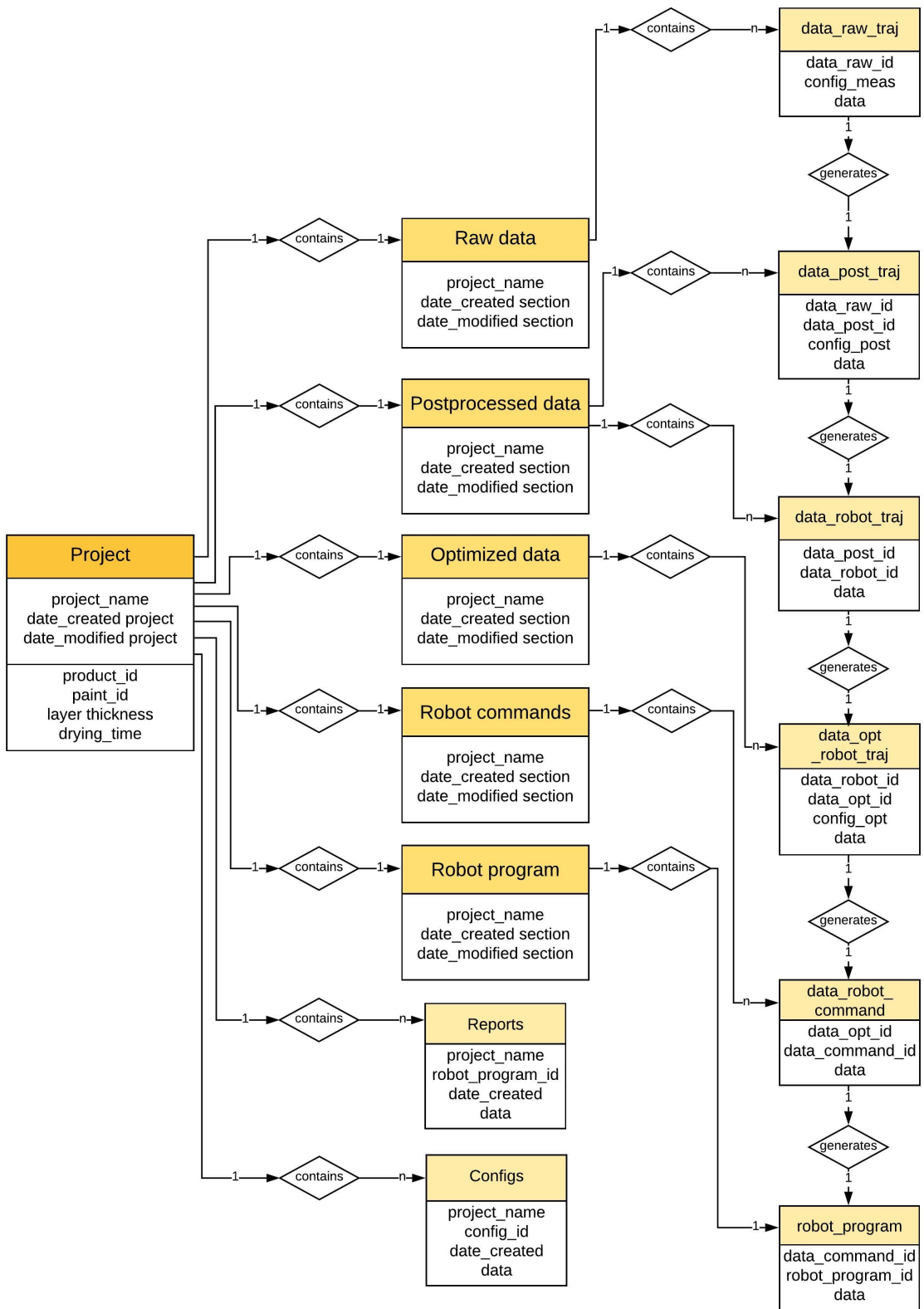


Figure 7.3: Entity relationship diagram of the Robo-Painter software.

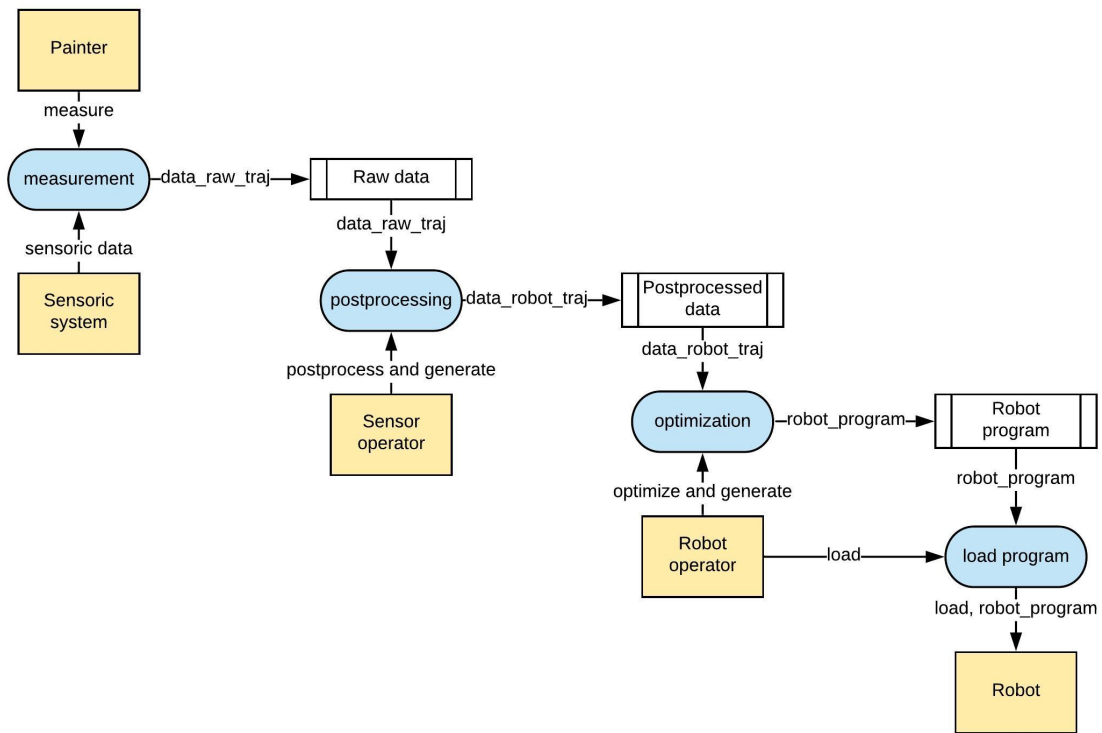


Figure 7.4: Data flow diagram of the Robo-Painter software at level 0, system overview.

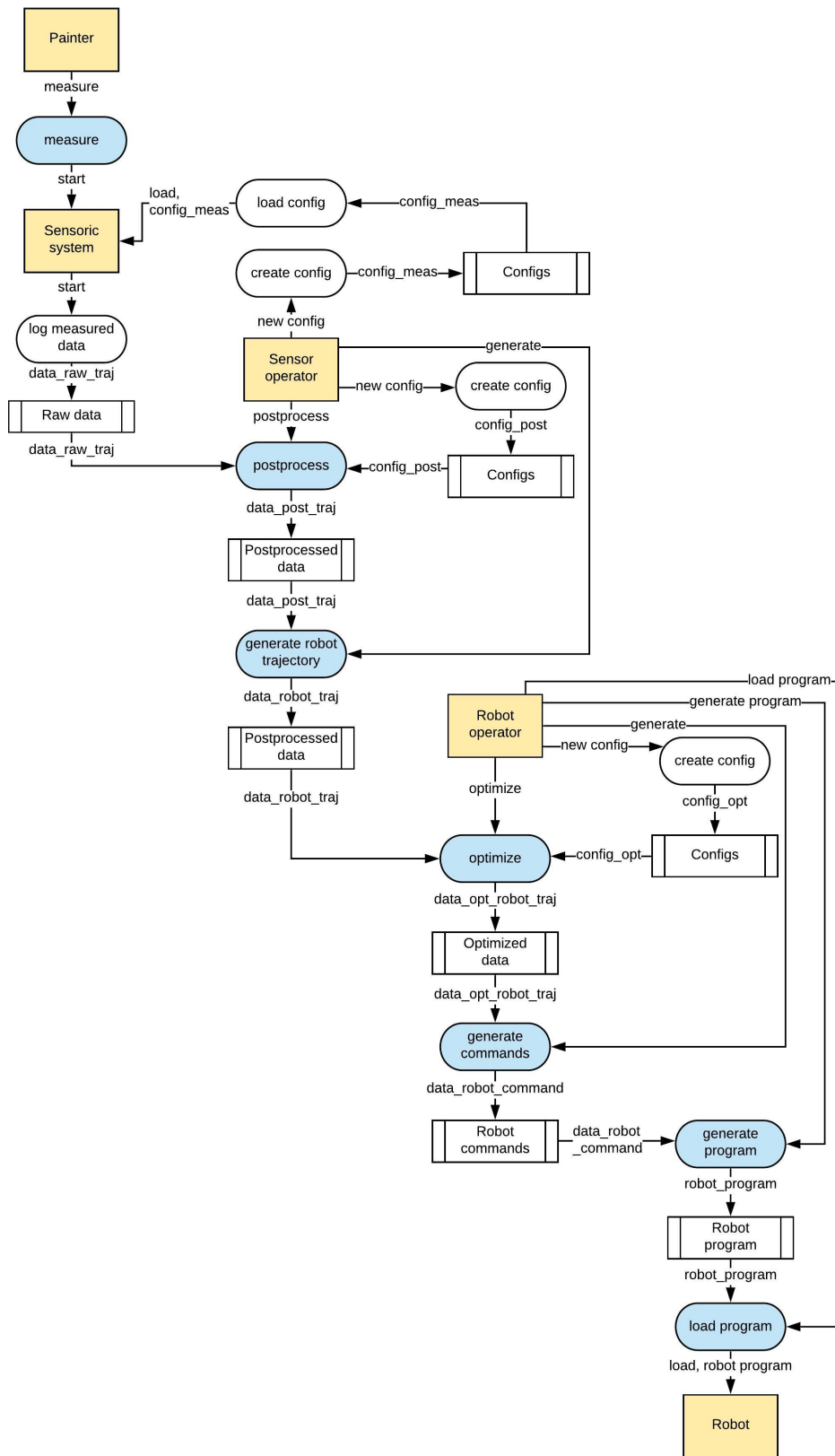


Figure 7.5: Data flow diagram of the Robo-Painter software at level 1, software processes.

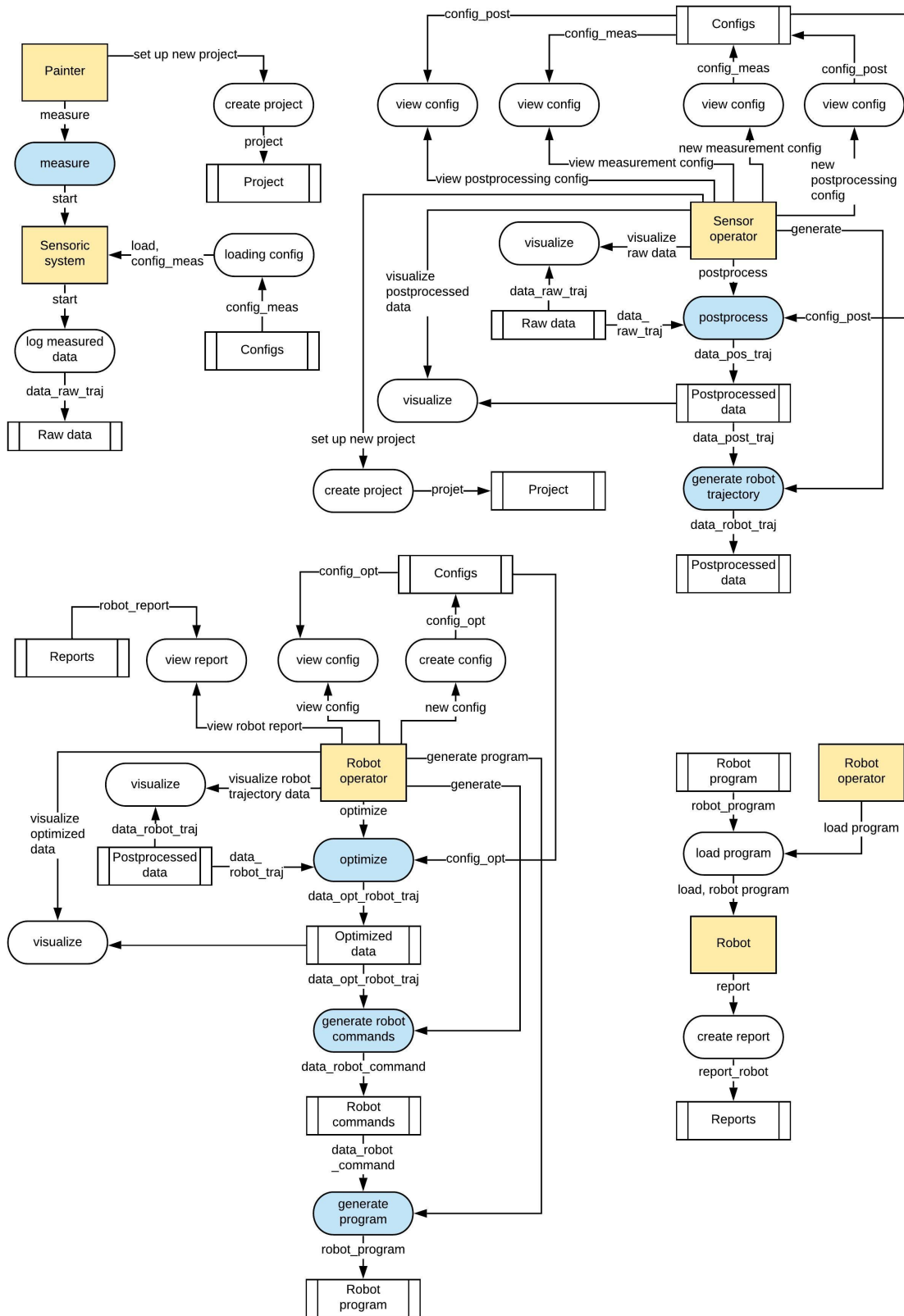


Figure 7.6: Data flow diagram of the Robo-Painter software at level 2, software processes from the user's (agent's) perspective.

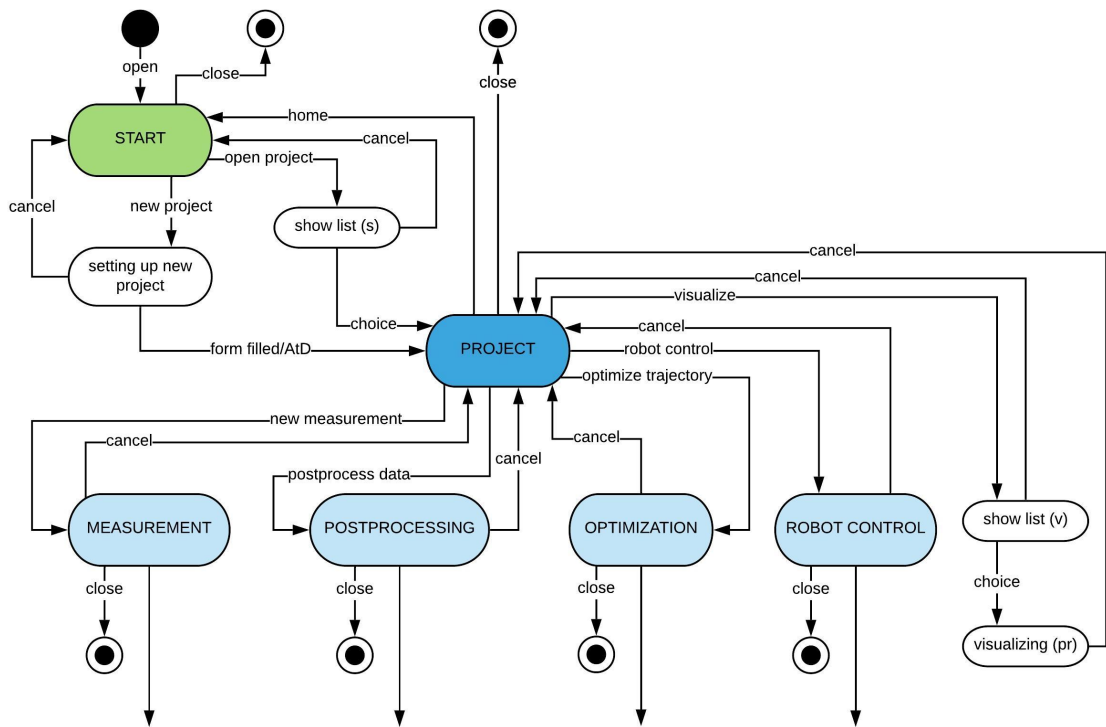


Figure 7.7: State transition diagram of the Robo-Painter software at level 0.

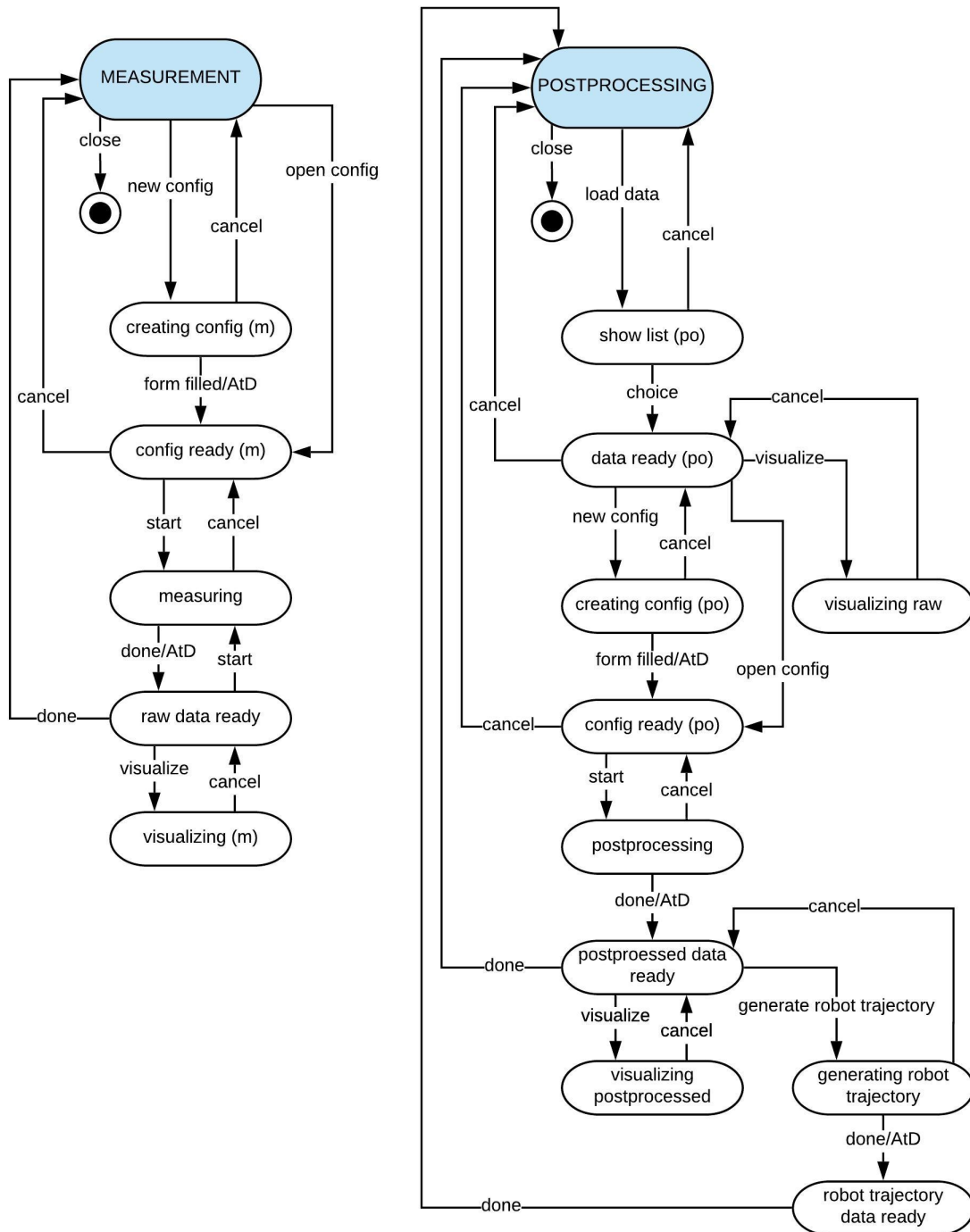


Figure 7.8: State transition diagram of the Robo-Painter software at level 1, from the functional subsection’s perspective (measurement and post-processing).

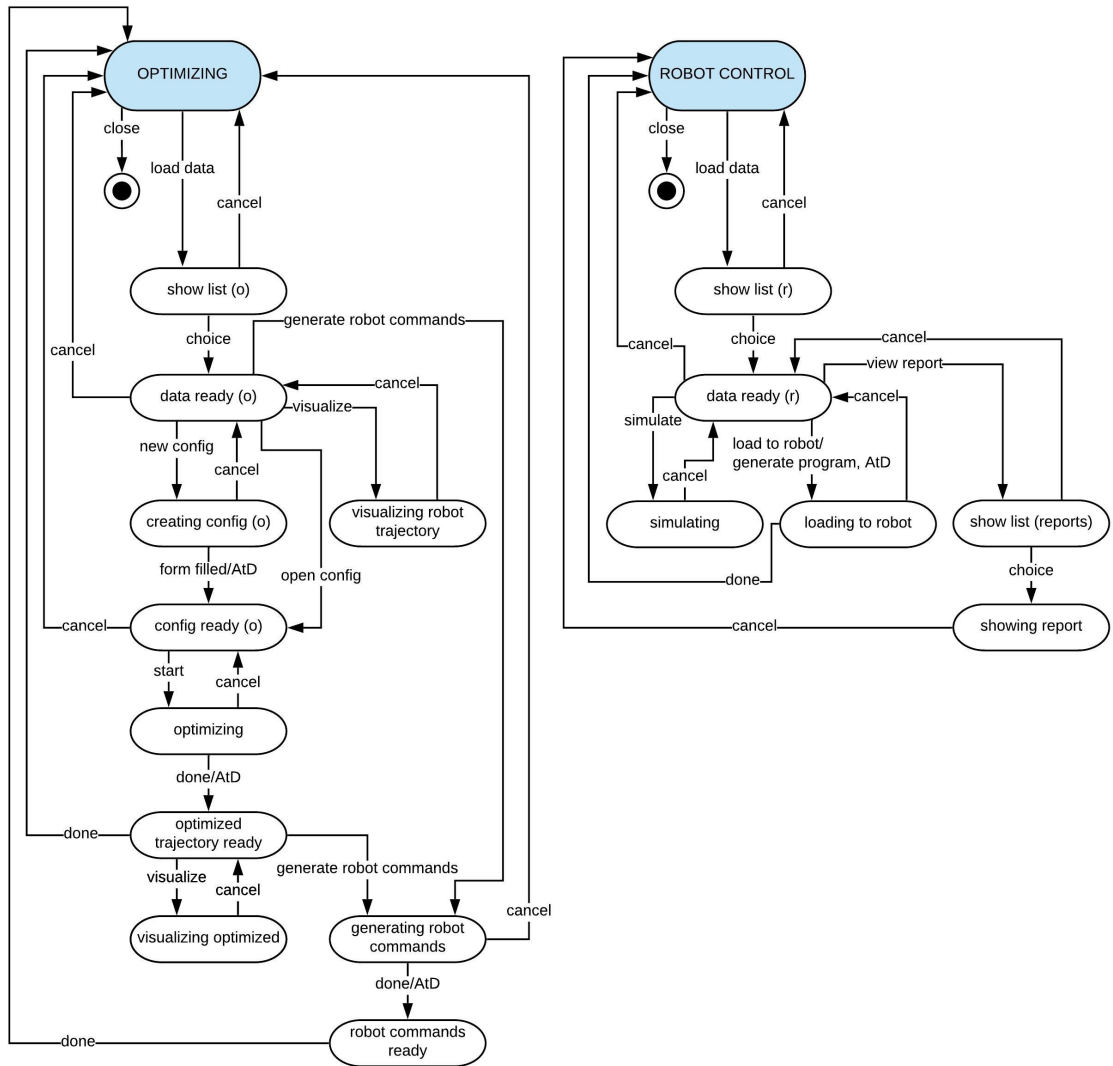


Figure 7.9: State transition diagram of the Robo-Painter software at level 1, from the functional subsection’s perspective (optimization and robot control).

Chapter 8

Robotic painting cell design and control

The final part of a design of an automation of any manufacturing process is the design of a real workplace with hardware and control software needed to carry out the manufacturing task. I elaborate on this final part in this chapter.

I designed a robotic painting cell that demonstrates how the automated painting operation would eventually look like. My intention was not to design a workplace that perfectly fit to a real manufacturer or a particular product, while that would require a careful choice of the used robot and hardware.

There are multiple options how to design the workplace with respect to the robot and product relationship. Either, the robot can be stationary with reachable area large enough to be able to paint a whole static product. Or, the product can be movable and moved when the robot needs to reach a different part of it. There could be also two or multiple stationary robots placed at the opposite sides of a stationary product, whose total range would be sufficient. Another option is to use a mobile robot, that would move around a stationary product, either along a given track or freely, using a navigational system.

The design I chose assumes one stationary robot and a movable product, realized by a turn table. The particular type of robot is not specified, since it depends on the manufacturer's choice. In the design, I included the assumption that a painting workplace is classified as possibly explosive and its design is therefore subject to safety restrictions. No electric drives were placed inside the painting booth.

During the design I followed a common approach of a design of any automated manufacturing operation that starts with analysis of the automated task, choice of hardware and software and further design and realization of a control part - creating a PLC (programmable logic controller) program.

8.1 Task description

8.1.1 The painting operation description

At the beginning of the process, the gates open (2a, 2b) and the product (1) starts moving to the painting booth. After the product (1) arrives at the required position, the gates (2a, 2b) close, the water wall (5) is turned on and the robot (6) starts painting the first side of the product (1). Once the first side is finished, the product (1) is turned by 180° and the second side is painted. When the painting finishes, the gates (2a, 2b) open and the product (1) leaves the painting booth.

Automatic mode

When the manufacturing line is turned on in automatic mode (auto_sw), the gates (2a, 2b) open (pneumatic motors Aa, Ab) and the overhead conveyor (3) starts to move (electric motor B). The bar code scanner (4) identifies the product (1) before it enters the painting booth.

Once the product (1) reaches the desired position in the painting booth (photoelectric sensor (s3)), the conveyor (3) stops (B). The gates (2a, 2b) close (Aa, Ab) and the water wall (5) is turned on. Based on the product identification, the first painting program of the robot (6) is selected and activated.

After the first painting program finishes, the turn table switch (7) turns the product (1) by 180° (pneumatic motor C). The second painting program is activated.

After the second painting program finishes, the turn table switch (7) turns the product (1) by 180° back (C). The water wall (5) is turned off and the gates (2a, 2b) open (Aa, Ab). The overhead conveyor (3) starts to move (B). The product (1) leaves the painting booth.

If an error occurs during any of the above processes, the operator is notified and the process is suspended. After the operator quits the failure, the process continues.

If the operator activates the emergency stop (em_stop_sw) the robot (6) goes to stop mode (steady, spray gun off). The gates (2a, 2b) open, the conveyor (3) stops. The water wall (5) is turned off. The system waits until the operator handles the emergency and switch the emergency stop (em_stop_sw) off again.

Manual mode

When the manufacturing line is turned on in manual mode (manual_sw), the operator opens (gates switcher mAa_sw, mAb_sw) the gates (2a, 2b) and turns on (conveyor switcher

mB_sw) the overhead conveyor (3). If turned on (p4m), the bar code scanner (4) identifies the product (1) before it enters the painting booth.

Once the product (1) reaches the desired position in the painting booth, the operator stops (conveyor switcher mB_sw) the conveyor (3). Now the operator closes (mAa_sw, mAb_sw) the gates (2a, 2b) and can turn the water wall (5) on (p5m). The operator can choose the painting program of the robot (6) and activate it (p6m). The operator can turn the product (1) by rotating (mC_sw) the turn table switch (7).

After the operator finishes the desired operations, the operator opens (gates switcher mAa_sw, mAb_sw) the gates (2a, 2b) and turns the overhead conveyor (3) on (mB_sw). The product (1) leaves the painting booth.

If the operator activates the emergency stop (em_stop_sw) the robot (6) goes to stop mode (steady, spray gun off). The gates (2a, 2b) open, the conveyor (3) stops. The water wall (5) is turned off. The system waits until the operator handles the emergency and switch the emergency stop (em_stop_sw) off again.

Elements (see Fig. 8.1)

- 1 - product
- 2a, 2b - gates
- 3 - overhead conveyor
- 4 - bar code scanner
- 5 - water wall
- 6 - robot
- 7 - turn table switch

Motors

- Aa, Ab - pneumatic motors (gates)
- B - electric motor (conveyor)
- C - pneumatic motor (turn table switch)

Sensors

- Aa0, Aa1, Ab0, Ab1 - pneumatic motors limit switches (gates)
- C0, C1 - pneumatic motor limit switch (turn table switch)
- s2a, s2b - photoelectric sensors (gates)
- s3 - photoelectric sensor (conveyor, product on place)

Switchers

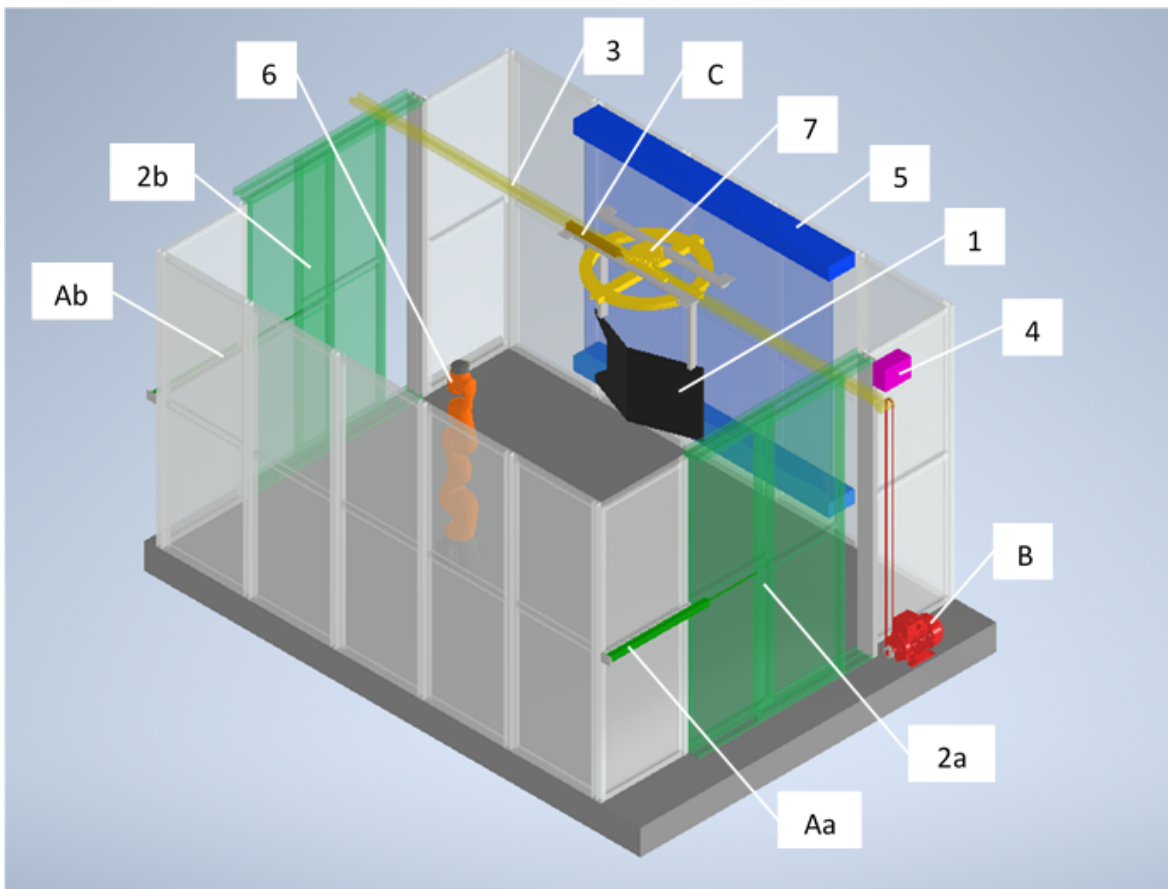


Figure 8.1: 3D model of the robotic painting cell with labeled elements.

start_sw - on/off switcher

mode_sw - mode switcher (auto_sw/manual_sw/service_sw/)

em_stop_sw - emergency stop switcher

mAa_sw, mAb_sw - gates manual open/close switcher

mB_sw - conveyor manual on/off switcher

mC_sw - turn table manual 180°/0° switcher

Buttons

fail_quit - operator quits an error

next - operator moves to next step in manual mode

Software processes

p4out - bar code scanner command (on/off)

p4m - bar code scanner manual command (on/off)

p4in - bar code scanner value (ready/id/error)
 p5out - water wall command (on/off)
 p5m - water wall manual command (on/off)
 p5in - water wall value (on/off/error)
 p6out - robot controller command (run program id/steady/home)
 p6m - robot controller manual command (run program id/steady/home)
 p6in - robot controller value (ready/running program id/error)

Timers

tAa, tAb - gates opening/closing
 tC - turn table switch turning
 ts3 - product arriving to desired place
 ts2b - product leaving the painting booth

■ 8.1.2 Fault diagnosis and isolation

Each state in the production mode has its associated error state. The transition to the error state is conditioned by the respective sensor readings and is specific for each state. The error state defines which error from the Tab. 8.1 is identified. Based on the identified error, the operator is notified and actions are taken. After the operator removes the error and press the quit button (fail_quit), the process returns to the state from which it was raised.

■ 8.1.3 Emergency stop

In any state in any mode (automatic/manual/service) it is possible to switch on the emergency stop switcher (em_stop_sw) and move to the emergency stop mode. In the emergency stop mode, the actions minimize possible danger and damage. Further, the operator is allowed to control the motors of the gates (Aa, Ab), conveyor (B) and the turning table (C). The operator can also give commands to the robot (p6out). See Tab. 8.2 for details.

■ 8.2 Task analysis

The task analysis was done in form of an SFC diagram. See diagrams in Fig. 8.3 - 8.6.

Identifier	Description	Condition	Action
etAa_o	opening 2a error (gates 2a)	tAa > tAa_max	Aa := 0
etAb_o	opening 2b error (gates 2b)	tAb > tAb_max	Ab := 0
etAa_c	closing 2a error (gates 2a)	tAa > tAa_max	Aa := 1
etAb_c	closing 2b error (gates 2b)	tAb > tAb_max	Ab := 1
etAa_c_m	closed gates 2a error (gates 2a)	mB_sw = 1 AND Aa0 = 1	B:=0
etAb_c_m	closed gates 2b error (gates 2b)	mB_sw= 1 AND Ab0 = 1	B:=0
etC_o	turning error opening (turn table 7)	tC > tC_max	C := 0
etC_c	turning error closing (turn table 7)	tC > tC_max	C := 1
es2a	gates closing when s2a detects an obstacle (gates 2a)	s3 = 1 AND s2a = 1 OR mAa_sw = 0 AND s2a = 1	Aa := 1
es2b	gates closing when s2b detects an obstacle (gates 2b)	s3 = 1 AND s2b = 1 OR mAb_sw = 0 AND s2b = 1	Ab := 1
ets3	product didn't arrive on place (product 0)	ts3 > ts3_max	B := 0
ets2b	product didn't leave the painting booth (product 0)	ts2b > ts2b_max	B := 0
ep4	bar code scanner error (scanner 4)	p4in = 'error'	-
ep5	water wall error (water wall 5)	p5in = 'error'	-
ep6	robot controller error (robot controller 6)	p6in = 'error'	-
ep5A_m	water running error (water wall 5)	mAa_sw = 1 AND mAb_sw = 1 AND p5in = 'on'	-
ep6A_m	robot operation error (robot controller 6)	mAa_sw = 1 AND mAb_sw = 1 AND p6in = 'id'	-

Table 8.1: Table of identified errors that can occur during a run of the PLC program.

Description	Condition	Action
emergency stop	em_stop_sw = 1	auto_sw := 0; Aa :=1; Ab :=1; B := 0; C :=0; p4out := 'off'; p5out := 'off'; p6out := 'steady'

Table 8.2: Details of the emergency stop mode that occurs when the emergency stop switcher is activated during a run of the PLC program.

8.2.1 Hardware design

All three pneumatic motors (Aa, Ab, C) were designed the same way. The desired movement is achieved by a single acting cylinder with spring return. The air inlet of the cylinder is connected via one-way flow control valve to a 3/2 mono stable valve. The 3/2 valve is electrically actuated and normally closed (closed when signal equals False). The 3/2 valve is connected to an air-preparation unit in form of an FRL (filter, regulator, lubricator) unit. The diagram of the pneumatic circuit is showed in Fig. 8.2.

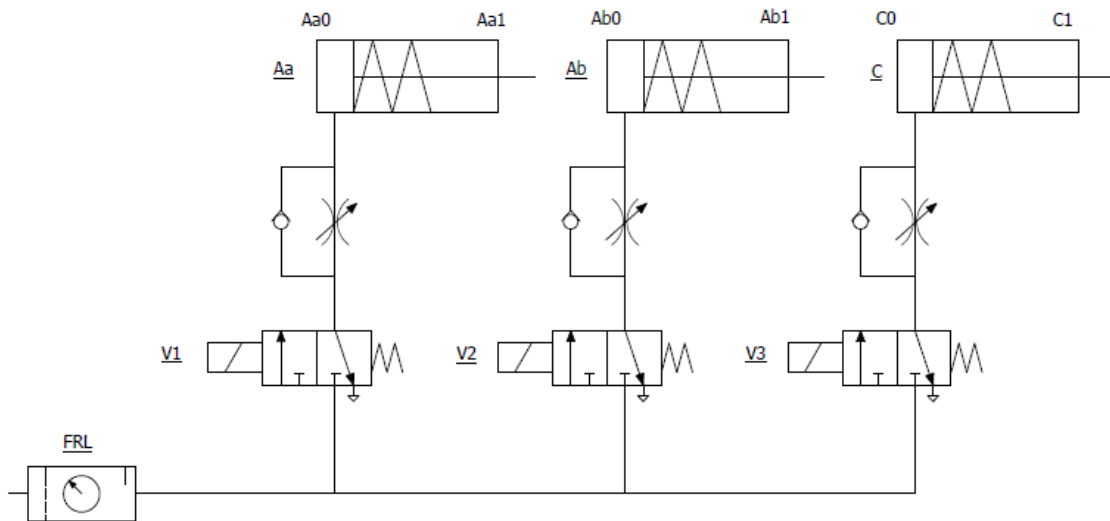


Figure 8.2: Diagram of the pneumatic circuit used in the robotic painting cell.

8.3 State model

State model of the task is shown in a form of an SFC diagram in Fig. 8.3. There are seven super states identified: stopped, starting, idle, emergency_stopped, service, automatic and manual. The emergency_stopped, automatic and manual states are elaborated further.

When the production is off, the production unit remains in the stopped state. After the start and initialization, the production unit goes to idle, from where it can continue to either service, automatic or manual mode, based on the choice of the operator. The operation unit stays in the set mode until the operator decides to stop or change it. After the operator decides to finish with the chosen mode, the necessary operations are finished and the unit returns to idle.

From any state or super state, it is possible to jump to the emergency_stopped super state. After the emergency is handled, the production unit transfers to idle.

8.4 State machine algorithm design

The individual SFC diagrams and the state model of the task were used to design the algorithm for PLC controller. The algorithm is based on the Moore state machine described for instance in [78].

The transitions between individual super states are controlled by super state control

transition index	cond_in	mask_in	cond_state	active_state
i

Table 8.3: Structure of the table of transitions.

state index	cond_out	mask_out
j

Table 8.4: Structure of the table of states.

algorithm. When the automatic/manual mode is activated, the unit is controlled by automatic/manual control algorithm respectively. The activation of desired control algorithm section is organized by local variable flag of type byte. Thanks to this system, the production unit can be easily extended by another modes (super states) as for example maintenance mode.

8.4.1 Moore state machine algorithm design

Each control algorithm works with a table of transitions (Tab. 8.3) and a table of states (Tab. 8.4). The number and order of rows in the table of transitions (Tab. 8.3) is equal to the identified transitions in the corresponding SFC diagram (Fig. 8.3 - 8.6). The columns represent the combination of input values (cond_in) and states (cond_state) needed to find a valid transition. The column mask_in chooses relevant input variables and ignores values of those that are irrelevant. The column active_state determines which states are activated when the valid transition is realized.

The output variable out of type word is set based on the table of states (Tab. 8.4). Each bit of the variable out represents one output variable. The column mask_out says which variables to change and which to keep unchanged. The column cond_out prescribes if the relevant variable should be set or reset. The values in the two tables are directly derived from the corresponding SFC diagrams (Fig. 8.3 - 8.6) for each control section.

At the beginning of each PLC cycle, the input variables and relevant local variables are loaded into an array input_array and converted to one variable in of type double word. Each bit represents one input variable. All input and output variables need to be of type bool. For this purpose, the string variables p4in, p5in, p5out etc. were replaced by Boolean variables p4in, p4in_error etc. At the end of each cycle, the algorithm writes values to output_array and then to the individual output variables based on the value of the output word out.

The main idea of the designed algorithm is presented in the code snippet below:

```

if flag = mode then
  i := 0;
  while i < n_mode_transitions do (*loop through table of transitions*)
    i := i + 1;
    if ((mask_in[i] and in) = cond_in[i]) and
        ((state and cond_state[i]) = cond_state[i])
      then state := active_state[i]; i := n_mode_transitions;
    end_if;
  end_while;
  if state = start_of_another_mode then (*flag set*)
    flag := new_mode;
    state := sl_in_new_mode;
  end_if;
end_if;

```

```

if flag = mode then
  for j := 1 to n_mode_states by 1 do (*loop through table of states*)
    if (state and all_state[j]) = all_state[j] then
      out := (cond_out[j] and mask_out[j]) or
              (out and not(mask_out[j]));
    end_if;
  end_for;
end_if;

```

■ 8.5 State machine algorithm implementation

The algorithm designed in section 8.4 was implemented, simulated and tested in OpenPLC Editor [79]. The code was written in Structured Text following the IEC 1131-3 standard. The tables of transitions and states for super state, automatic and manual mode are shown in Fig. 8.7 - 8.19.

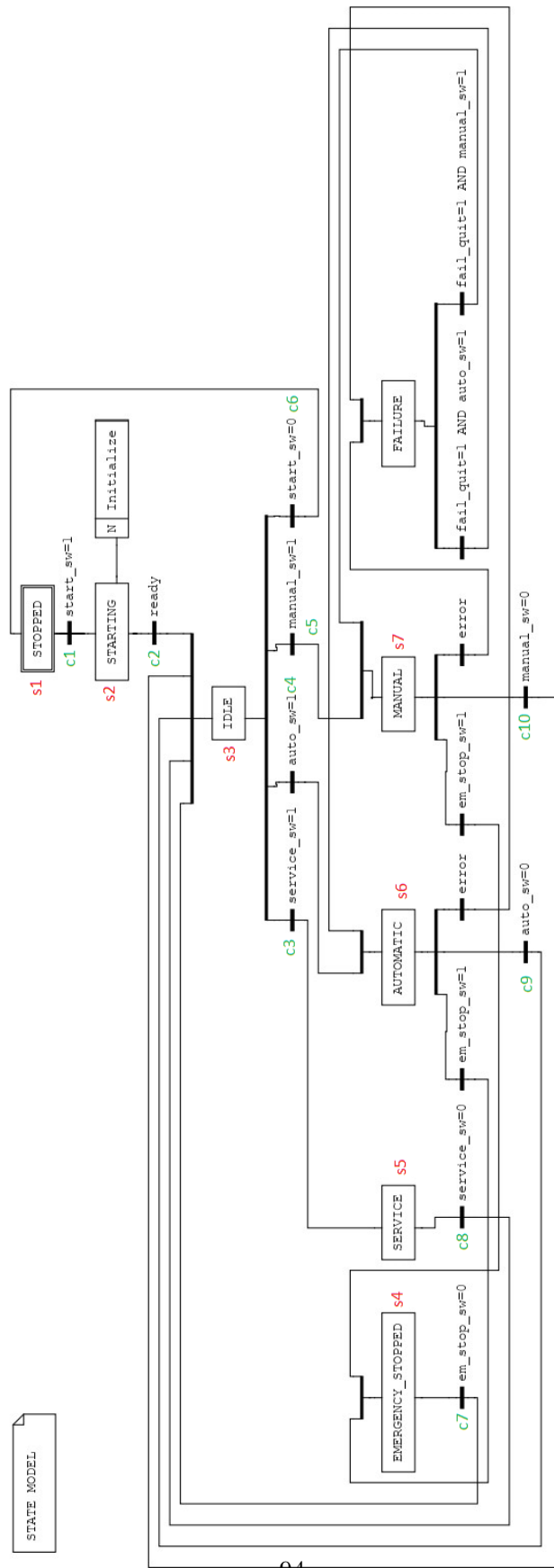


Figure 8.3: State model. Sequential function chart of the super state mode, with identified transitions (c) and states (s).

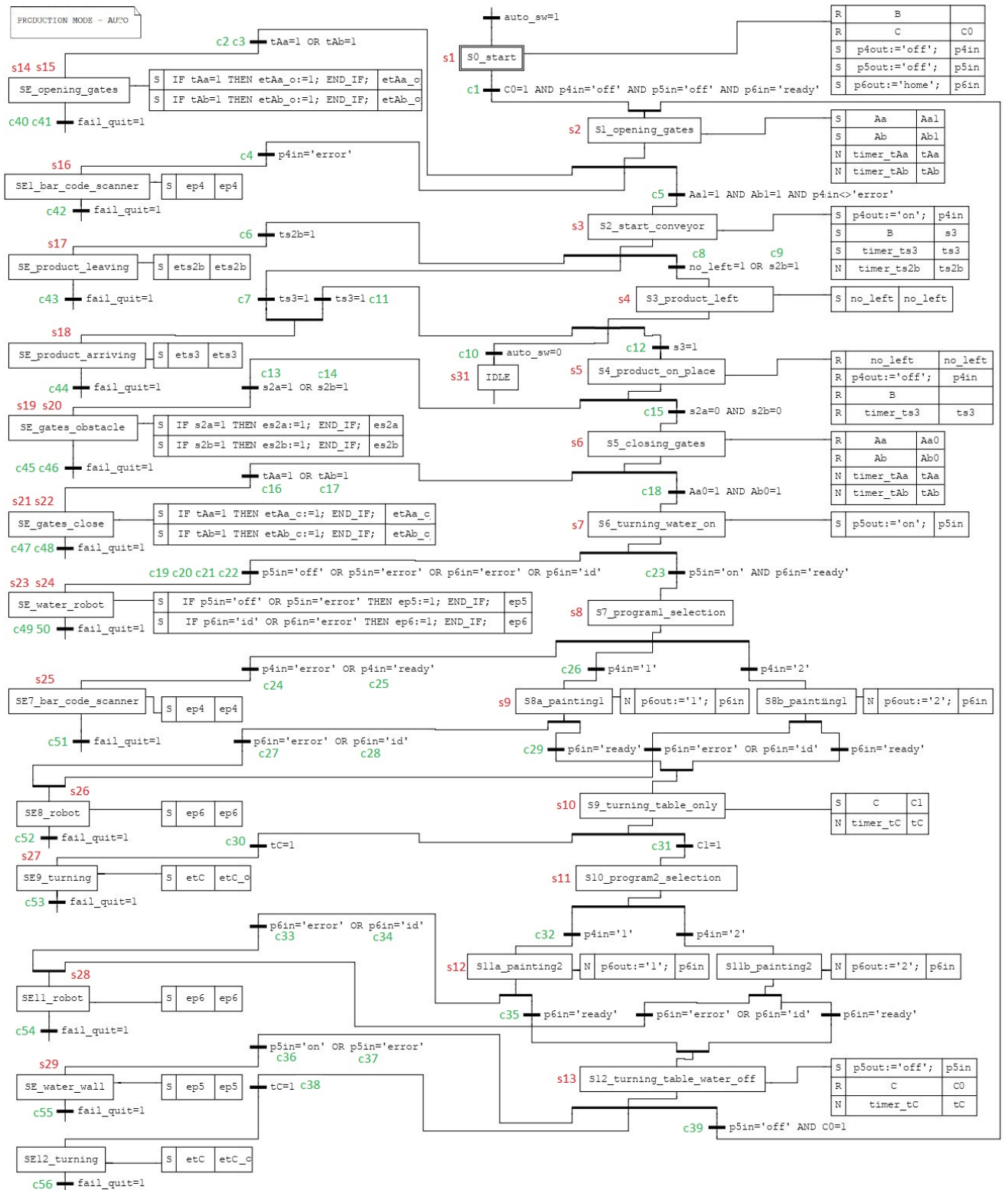


Figure 8.4: Sequential function chart of the automatic mode, with identified transitions (c) and states (s).

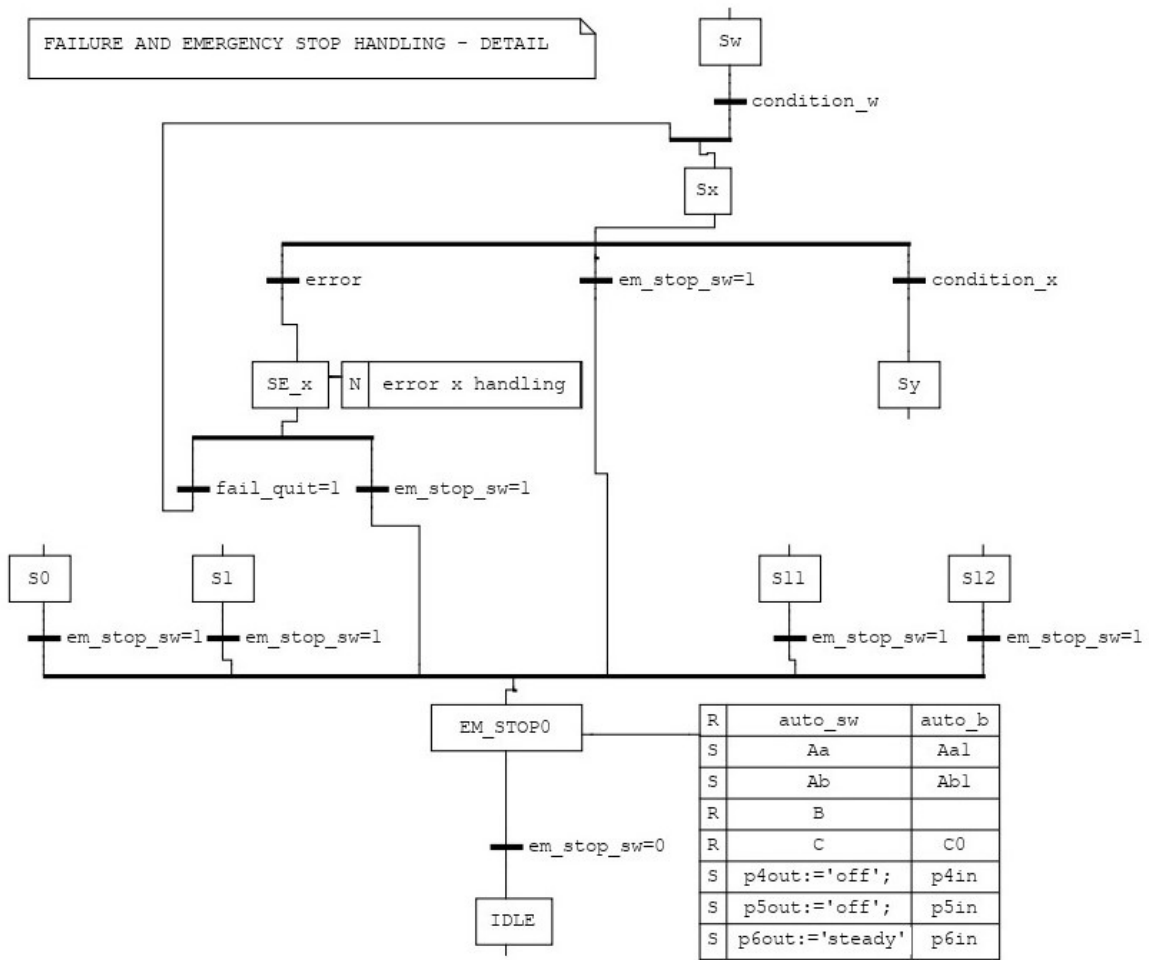


Figure 8.5: Sequential function chart of the automatic mode, detail of the failure and emergency stop handling part.

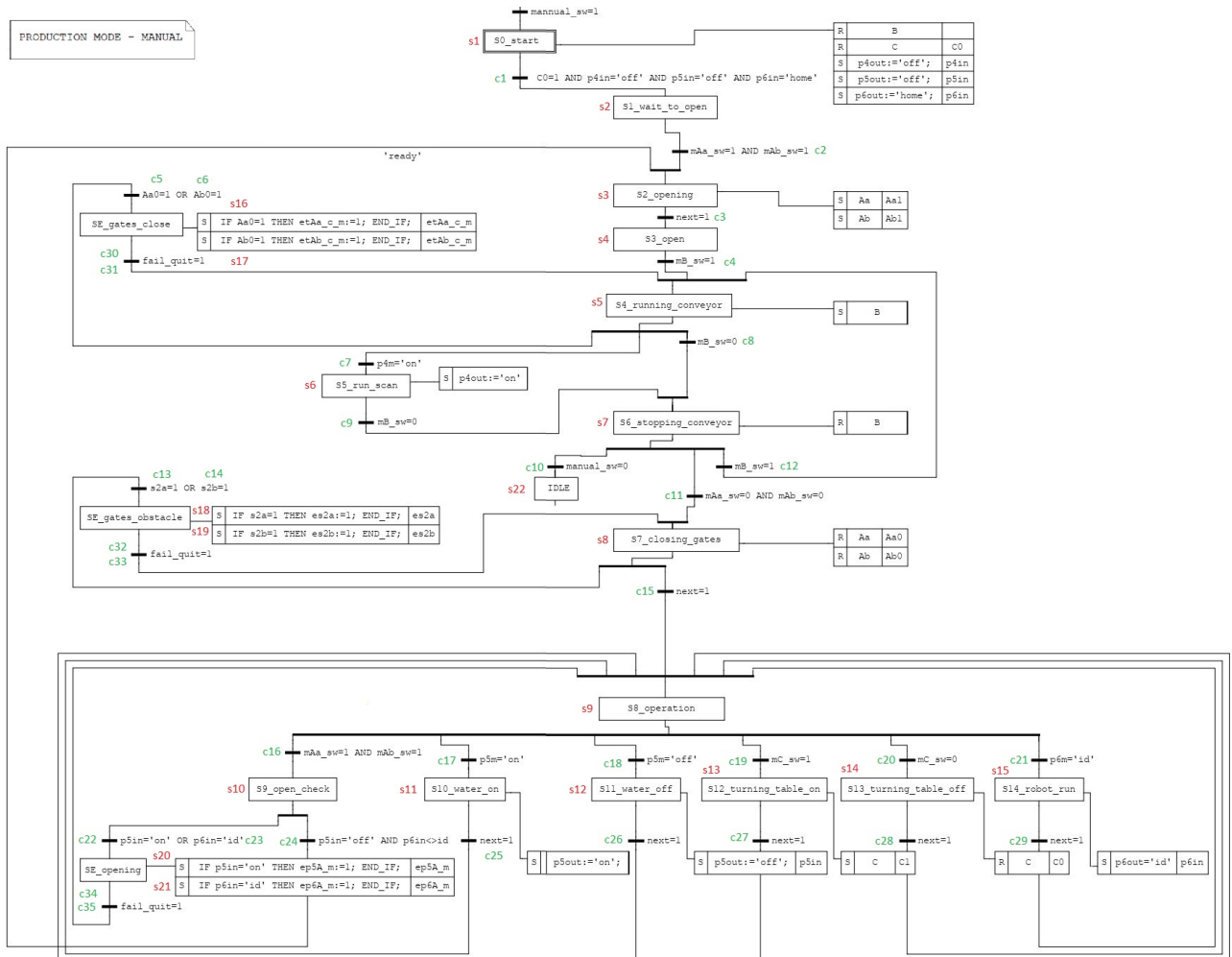


Figure 8.6: Sequential function chart of the manual mode, with identified transitions (c) and states (s).

		in	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	fail_quit	p6in_error	p6_in	p5in_error	p5_in	p4in_error	p4_in	em_stop_sw	service_sw	manula_sw	auto_sw	start_sw	s3	s2b	s2a	C1	C0	Ab1	Ab0	Aa1	Aa0					
i	cond_in_s																																							d			
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	512		
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4096	
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1024		
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2048		
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
i	mask_in_s																																							d			
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	512	
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	344080
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4096
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1024
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2048
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	512	
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8192	
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4096	
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1024	
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2048	

Figure 8.7: Table of transitions for the super state mode 1/2.

		state	7	6	5	4	3	2	1	
i	cond_state_s									d
1	0	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	0	1	0	2
3	0	0	0	0	0	0	1	0	0	4
4	0	0	0	0	0	0	1	0	0	4
5	0	0	0	0	0	0	1	0	0	4
6	0	0	0	0	0	0	1	0	0	4
7	0	0	0	0	0	1	0	0	0	8
8	0	0	0	1	0	0	0	0	0	16
9	0	0	1	0	0	0	0	0	0	32
10	1	0	0	0	0	0	0	0	0	64
i	active_state_s									d
1	0	0	0	0	0	0	0	1	0	2
2	0	0	0	0	0	0	1	0	0	4
3	0	0	0	0	1	0	0	0	0	16
4	0	0	1	0	0	0	0	0	0	32
5	0	1	0	0	0	0	0	0	0	64
6	0	0	0	0	0	0	0	0	1	1
7	0	0	0	0	0	0	1	0	0	4
8	0	0	0	0	0	0	1	0	0	4
9	0	0	0	0	0	0	1	0	0	4
10	0	0	0	0	0	0	1	0	0	4

Figure 8.8: Table of transitions for the super state mode 2/2.

out	timer_ts2b	timer_ts3	timer_tC	timer_tAb	timer_tAa	no_left	p6out	p5out	p4out	C	B	Ab	Aa	
j	cond out s												d	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0	0	0	0	0	0	64
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	1	1	3
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0	0	0	64
7	0	0	0	0	0	0	1	0	0	0	0	0	0	64
j	mask out s												d	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	1	1	1	1	0	0	124
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	1	1	1	1	1	1	127
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	1	1	1	1	0	0	124
7	0	0	0	0	0	0	1	1	1	1	1	0	0	124

Figure 8.9: Table of states for the super state mode.

out	timer_ts2b	timer_ts3	timer_tC	timer_tAb	timer_tAa	no_left	p6out	p5out	p4out	C	B	Ab	Aa	
j	cond_out_a												d	
1	0	0	0	0	0	0	1	0	0	0	0	0	0	64
2	0	0	0	1	1	0	0	0	0	0	0	1	1	771
3	1	1	0	0	0	0	0	0	1	0	1	0	0	6164
4	0	0	0	0	0	1	0	0	0	0	0	0	0	128
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	1	1	0	0	0	0	0	0	0	0	768
7	0	0	0	0	0	0	0	1	0	0	0	0	0	32
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	1	0	0	0	0	0	0	64
10	0	0	1	0	0	0	0	0	0	1	0	0	0	1032
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	1	0	0	0	0	0	0	64
13	0	0	1	0	0	0	0	0	0	0	0	0	0	1024
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	1	1
20	0	0	0	0	0	0	0	0	0	0	0	1	0	2
21	0	0	0	0	0	0	0	0	0	0	0	0	1	1
22	0	0	0	0	0	0	0	0	0	0	0	1	0	2
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	1	0	0	0	8
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0

out	timer_ts2b	timer_ts3	timer_tC	timer_tAb	timer_tAa	no_left	p6out	p5out	p4out	C	B	Ab	Aa	
j	mask_out_a												d	
1	0	0	0	0	0	0	1	1	1	1	1	0	0	124
2	0	0	1	1	1	0	0	0	0	0	0	0	1	1795
3	1	1	0	1	1	0	0	0	1	0	1	0	0	6932
4	1	0	0	0	0	1	0	0	0	0	0	0	0	4224
5	0	1	0	0	0	1	0	0	1	0	0	1	0	2196
6	0	0	0	1	1	0	0	0	0	0	0	0	1	771
7	0	0	0	1	1	0	0	1	0	0	0	0	0	800
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	1	0	0	0	0	0	0	64
10	0	0	1	0	0	0	1	0	0	1	0	0	0	1096
11	0	0	1	0	0	0	0	0	0	0	0	0	0	1024
12	0	0	0	0	0	0	1	0	0	0	0	0	0	64
13	0	0	1	0	0	0	1	1	0	1	0	0	0	1128
14	0	0	0	1	1	0	0	0	0	0	0	0	1	769
15	0	0	0	1	1	0	0	0	0	0	0	1	0	770
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	1	0	0	0	0	0	0	0	0	0	0	1	0	4100
18	0	1	0	0	0	0	0	0	0	0	0	1	0	2052
19	0	0	0	0	0	0	0	0	0	0	0	0	1	1
20	0	0	0	0	0	0	0	0	0	0	0	0	1	2
21	0	0	0	1	1	0	0	0	0	0	0	0	1	769
22	0	0	0	1	1	0	0	0	0	0	0	0	1	770
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	1	0	0	0	0	0	0	1	0	0	0	1032
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	1	0	0	0	0	0	0	1	0	0	0	1032
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8.14: Table of states for the automatic mode.

in	0	0	0	next	p6m	p5m	p4m	mC_sw	mB_sw	mAb_sw	mAa_sw	fail_quit	p6in_error	p6_in	p5in_error	p5_in	p4in_error	p4_in	em_stop_sw	service_sw	manula_sw	auto_sw	start_sw	s3	s2b	s2a	C1	C0	Ab1	Ab0	Aa1	Aa0			
i	cond_in_m																																d		
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16		
2	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6291456		
3	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456		
4	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8388608		
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1		
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	4		
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33554432		
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
12	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8388608	
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	64	
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	128	
15	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456	
16	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6291456	
17	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	67108864	
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16777216	
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	134217728	
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65536	
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	262144	
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
25	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456
26	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456
27	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456
28	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456
29	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576

Figure 8.15: Table of transitions for the manual mode 1/4.

	in	0	0	0	next	p6m	p5m	p4m	mC_sw	mB_sw	mAb_sw	mAa_sw	fail_quit	p6in_error	p6_in	p5in_error	p5_in	p4in_error	p4_in	em_stop_sw	service_sw	manula_sw	auto_sw	start_sw	s3	s2b	s2a	C1	C0	Ab1	Ab0	Aa1	Aa0			
i	mask_in_m																																d			
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	344080	
2	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6291456	
3	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456	
4	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8388608	
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33554432	
8	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8388608	
9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8388608	
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	2048	
11	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6291456	
12	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8388608	
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	64	
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	128	
15	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456	
16	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6291456	
17	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	67108864	
18	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	67108864	
19	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16777216	
20	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16777216	
21	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	134217728	
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65536	
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	262144	
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	327680	
25	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456	
26	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456	
27	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456	
28	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456	
29	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	268435456	
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576	
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576	
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576	
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576	
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576	
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576

Figure 8.16: Table of transitions for the manual mode 2/4.

state	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	d
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	4
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	8
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	16
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	16
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	16
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	16
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	32
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	64
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	64
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	64
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	128
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	128
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	128
16	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
17	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
18	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
19	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
20	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
21	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
22	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	512
23	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	512
24	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	512
25	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1024
26	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	2048
27	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	4096
28	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	8192
29	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16384
30	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32768
31	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65536
32	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	131072
33	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	262144
34	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	524288
35	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576

Figure 8.17: Table of transitions for the manual mode 3/4.

state	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
i	active_state_m																					d	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	4
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	8
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	16
5	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32768
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65536
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	32
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	64
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	64
10	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2097152
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	128
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	16
13	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	131072
14	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	262144
15	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
16	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	512
17	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1024
18	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	2048
19	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	4096
20	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	8192
21	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16384
22	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	524288
23	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1048576
24	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
25	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
26	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
27	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
28	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
29	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	16
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	16
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	128
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	128
34	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256
35	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	256

Figure 8.18: Table of transitions for the manual mode 4/4.

out	timer_ts2b	timer_ts3	timer_tC	timer_tAb	timer_tAa	no_left	p6out	p5out	p4out	C	B	Ab	Aa	
j	cond_out_m												d	
1	0	0	0	0	0	0	1	0	0	0	0	0	0	64
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	1	1	3
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	1	0	0	4
6	0	0	0	0	0	0	0	0	1	0	0	0	0	16
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	1	0	0	0	0	0	32
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	1	0	0	0	8
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	1	0	0	0	0	0	0	64
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	1	1
19	0	0	0	0	0	0	0	0	0	0	0	1	0	2
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0

j	mask out m												d	
1	0	0	0	0	0	0	1	1	1	1	1	0	0	124
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	1	1	3
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	1	0	0	4
6	0	0	0	0	0	0	0	0	1	0	0	0	0	16
7	0	0	0	0	0	0	0	0	0	0	1	0	0	4
8	0	0	0	0	0	0	0	0	0	0	0	1	1	3
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	1	0	0	0	0	0	32
12	0	0	0	0	0	0	0	1	0	0	0	0	0	32
13	0	0	0	0	0	0	0	0	0	1	0	0	0	8
14	0	0	0	0	0	0	0	0	0	1	0	0	0	8
15	0	0	0	0	0	0	1	0	0	0	0	0	0	64
16	0	0	0	0	0	0	0	0	0	0	1	0	0	4
17	0	0	0	0	0	0	0	0	0	0	1	0	0	4
18	0	0	0	0	0	0	0	0	0	0	0	0	1	1
19	0	0	0	0	0	0	0	0	0	0	0	1	0	2
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8.19: Table of states for the manual mode.



Chapter 9

Discussion and conclusions

This thesis is concerned with a problem of automation of an industrial painting operation. This technological operation is indeed a specific manufacturing operation and there are many aspects of it making it difficult to automate. However, there have been attempts to do so, their overview is presented in chapter 2.

The aim to automate industrial painting is motivated by several aspects. An automated manufacturing operation can be, after an initial investment, performed repeatedly without additional cost and effort, except the operating cost and maintenance. An automated robotic workplace can run constantly for a long time, it is efficient and works with guaranteed repeatability.

Although the economic aspects are self-evident, individual conditions should be assessed. A company painting products in only medium or small series or batches may hesitate if the investment is profitable. It would need to program the robot each time a new product comes, this procedure may be time and cost demanding and eventually not worth it. Another aspect to consider is, that the quality of end product depends directly on the planned robot trajectory and the quality of its controls. There is a risk that the performance will drop if the robot is not programmed well enough. This work aims to handle these issues and propose a solution that is suitable for companies doing middle and small batch production and is based on utilizing skills and know-how of human industrial painters. The solution is presented in chapter 3.

Another reason to automate industrial painting are the health concerns connected to it. The work of an industrial painter is physically demanding and the working environment is harsh, unpleasant and possibly unhealthy. At the end, this issue is projected back to the economical site of the problem, since it leads to a lack of work force, lack of people willing to do industrial painting and further results in lack of highly skilled workers in this area. Fear of destroying job opportunities by automation of manufacturing processes is odd in general, in this particular case especially. There is already a problem of too few people willing to do

tracking is one element of the painting cell, together with other necessary hardware to carry out automated painting. The resultant unit needs an industrial level control, and therefore this chapter includes control program for a PLC and so finishes the process of industrial painting automation.



Chapter 10

Future work

This work contains analysis of the problem of industrial painting automation, besides that, attempts have been made to provide a solution to it. Real hardware was chosen and tested and the final form of motion tracking system was designed.

The next step is to carry out measurements in a real painting workplace. All hardware and needed parts of software are ready, the measurement set-up is thought-out and described in this work. What needs to be done more, is the design of a custom holder for fixing the T265 camera to a spray gun and a jig for positioning the spray gun into during the measurement initialization. These items depend on the environment of the particular painting workplace.

If a suitable product is chosen for the experiment in a real painting workplace, the data can be used for robot control program generation. Then the robotic painting can be also experimentally tested, its performance can be compared with the provided simulation.

The contribution of this work could be further extended in the software part. The study of a software project done in this work should be used to create a robust and user friendly software. All individual necessary parts have already been elaborated, a graphical user interface and an architecture that would frame and unify them are yet still to be done.

A more detailed study of the layout of the robotic painting cell can be made in case a real industrial painting workplace gets involved. Specific requirements and remarks would be incorporated to the robotic painting cell design and a real working automated industrial painting workplace would be created.



Bibliography

1. WALLÉN, Johanna. The History of the Industrial Robot. *Linköping University Electronic Press*. 2008.
2. TRALLFA. *Photography of the first industrial robot* [online] [visited on 2020-07-08]. Available from: <https://trallfa.no/en/#history>.
3. FOTOVVATI, Behzad; NAMDARI, Navid; DEHGHANGHADIKOLAEI, Amir. On Coating Techniques for Surface Protection: A Review. 2019, vol. 3. Available from DOI: 10.3390/jmmp3010028.
4. SUH, S. -.; WOO, I. -.; NOH, S. -. Development of an automatic trajectory planning system (ATPS) for spray painting robots. 1991, pp. 1948–1955 vol.3.
5. WEIHUA SHENG; NING XI; MUMIN SONG; YIFAN CHEN; MACNEILLE, P. Automated CAD-guided robot path planning for spray painting of compound surfaces. 2000, vol. 3, pp. 1918–1923 vol.3.
6. HULÍNSKÝ, Lukáš. *Automated painting algorithms*. Czech Technical University in Prague, 2019.
7. PICHLER, Andreas; VINCZE, Markus; ANDERSEN, Henrik; MADSEN, Ole; HAUSLER, K. A Method for Automatic Spray Painting of Unknown Parts. 2002, vol. 1, pp. 444–449 vol.1. ISBN 0-7803-7272-7. Available from DOI: 10.1109/ROBOT.2002.1013400.
8. ANTONIO, J. K. Optimal trajectory planning for spray coating. 1994, pp. 2570–2577 vol.3.
9. AGT, Robotics. *Robotic Painting Cell with STL* [online] [visited on 2020-07-09]. Available from: <https://agtrobotics.com/machinery/robotic-painting-cell-with-slt/>.
10. FANUC. *Wooden chairs painted by FANUC M-20iA Robots* [online] [visited on 2020-07-09]. Available from: <https://en.industryarena.com/fanuc/tv/wooden-chairs-painted-by-fanuc-m-20ia-robots--7822.html>.

11. OKIBO. *Smart robotics for construction sites* [online] [visited on 2020-07-09]. Available from: <https://okibo.com>.
12. ROBNOR. *Virtual reality in reality* [online] [visited on 2020-07-09]. Available from: <https://www.robnor.se/virtual-reality>.
13. CHALLA, Subhash; MORELANDE, Mark R.; MUSICKI, Darko; EVANS, Robin J. *Fundamentals of Object Tracking*. USA: Cambridge University Press, 2011. ISBN 978-0-521-87628-5.
14. BORENSTEIN, Johann; EVERETT, Hobart R.; FENG, Liqiang; WEHE, David K. Mobile robot positioning: Sensors and techniques. *J. Field Robotics*. 1997, vol. 14, pp. 231–249.
15. *IPIN 2019 Tenth international conference on indoor positioning and indoor navigation* [online] [visited on 2020-07-07]. Available from: <http://ipin-conference.org/2019/>.
16. ALARIFI, Abdulrahman; AL-SALMAN, AbdulMalik; ALSALEH, Mansour; ALNAFESSAH, Ahmad; ALHADHRAMI, Suheer; AL-AMMAR, Mai; AL-KHALIFA, Hend. Ultra Wideband Indoor Positioning Technologies: Analysis and Recent Advances. *Sensors*. 2016, vol. 16, pp. 1–36. Available from DOI: 10.3390/s16050707.
17. *Three and Five Axis Laser Tracking Systems, PA 4714339A (US)*. Inventor KAM C. LAU, Robert J. Hocken. Available also from: <https://patents.google.com/patent/US4714339A/en>.
18. LIU, Xiaohan; MAKINO, Hideo; MASE, Kenichi. Improved Indoor Location Estimation Using Fluorescent Light Communication System with a Nine-Channel Receiver. *IEICE Transactions*. 2010, vol. 93-B, pp. 2936–2944. Available from DOI: 10.1587/transcom.E93.B.2936.
19. GREIFF, M.; ROBERTSSON, A.; BERNTORP, K. Performance Bounds in Positioning with the VIVE Lighthouse System. *2019 22th International Conference on Information Fusion (FUSION)*. 2019, pp. 1–8.
20. WOLF, J.; BURGARD, W.; BURKHARDT, H. Robust vision-based localization by combining an image-retrieval system with Monte Carlo localization. *IEEE Transactions on Robotics*. 2005, vol. 21, no. 2, pp. 208–216.
21. INTEL, Corporation. *Intel RealSense Tracking Camera T265* [online] [visited on 2020-07-01]. Available from: <https://www.intelrealsense.com/tracking-camera-t265/>.
22. ZHANG, Zhengyou. Microsoft Kinect Sensor and Its Effect. *IEEE MultiMedia* [online]. 2012, vol. 19, pp. 4–12 [visited on 2020-06-07]. Available from: <https://www.microsoft.com/en-us/research/publication/microsoft-kinect-sensor-and-its-effect/>.

23. KOK, Manon; HOL, Jeroen D.; SCHÖN, Thomas B. Using Inertial Sensors for Position and Orientation Estimation. *Foundations and Trends in Signal Processing*. 2017, vol. 11, no. 1-2, pp. 1–153. ISSN 1932-8354. Available from DOI: 10.1561/20000000094.
24. ANG, W. T.; KHOO, S. Y.; KHOSLA, P. K.; RIVIERE, C. N. Physical model of a MEMS accelerometer for low-g motion tracking applications. 2004, vol. 2, pp. 1345–1351 Vol.2.
25. KOWALCZUK, Zdzislaw; MERTA, Tomasz. Modelling an accelerometer for robot position estimation. *2014 19th International Conference on Methods and Models in Automation and Robotics, MMAR 2014*. 2014, pp. 909–914. Available from DOI: 10.1109/MMAR.2014.6957478.
26. HAVERINEN, Janne; KEMPPAINEN, Anssi. Global indoor self-localization based on the ambient magnetic field. *Robotics and Autonomous Systems*. 2009, vol. 57, no. 10, pp. 1028–1035. ISSN 0921-8890. Available from DOI: <https://doi.org/10.1016/j.robot.2009.07.018>. 5th International Conference on Computational Intelligence, Robotics and Autonomous Systems (5th CIRAS).
27. HEXAMITE, LTD. *xyz laboratory 3D studio* [online]. 2016 [visited on 2020-07-06]. Available from: <https://www.hexamite.com/pdfs/xyzlab.pdf>.
28. ZHOU, Bing; ELBADRY, Mohammed; GAO, Ruipeng; YE, Fan. BatTracker: High Precision Infrastructure-free Mobile Device Tracking in Indoor Environments. *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 2017.
29. JOHAN, Larsson. *Distance Estimation and Positioning Based on Bluetooth Low Energy Technology*. 2015. Available also from: <http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-174857>.
30. FOXLIN, Eric; HARRINGTON, Michael; ALTSHULER, Yury. Miniature 6-DOF inertial system for tracking HMDs. *Proceedings of SPIE - The International Society for Optical Engineering*. 2002. Available from DOI: 10.1117/12.317434.
31. WON, Seong-hoon; MELEK, W.W.; GOLNARAGHI, M. A Kalman/Particle Filter-Based Position and Orientation Estimation Method Using a Position Sensor/Inertial Measurement Unit Hybrid System. *Industrial Electronics, IEEE Transactions on*. 2010, vol. 57, pp. 1787–1798. Available from DOI: 10.1109/TIE.2009.2032431.
32. NASA, Facility Navigation Ancillary Information . *An Overview of Reference Frames and Coordinate Systems in the SPICE Context* [online]. 2020 [visited on 2020-07-08]. Available from: <https://naif.jpl.nasa.gov/pub/>

- naif/self_training/individual_docs/16_frames_and_coordinate_systems.pdf.
Tutorials.
33. STEJSKAL, Vladimír; VALÁŠEK, Michael. *Kinematics and dynamics of machinery*. New York: Marcel Dekker, 1996. ISBN 0-8247-9731-0.
 34. HENDERSON, D. M. *Euler Angles, Quaternions, and Transformation Matrices — Working Relationships*. Texas: NASA Technical report NASA-TM-74839, 1977.
 35. HIBBELER, R.C. *Engineering Mechanics: Dynamics*. 14th ed. Pearson, 2016. ISBN 9780133915389.
 36. HEXAGON, AB. *Laser Tracker Systems* [online]. 2020 [visited on 2020-07-02]. Available from: <https://www.hexagonmi.com/products/laser-tracker-systems>.
 37. LEICA, Geosystems AG. *Leica AT930/AT960 User Manual* [online]. Switzerland, 2017.
 38. HEXAGON, Metrology. *Leica Absolute Tracker AT901 Product Brochure* [online]. Germany, 2010.
 39. PAN, T.; KUO, C.; HU, M. A noise reduction method for IMU and its application on handwriting trajectory reconstruction. 2016, pp. 1–6.
 40. GUO, Hui; HONG, Huajie. Research on Filtering Algorithm of MEMS Gyroscope Based on Information Fusion. *Sensors*. 2019, vol. 19. Available from DOI: 10.3390/s19163552.
 41. DU, Jiaying. *Signal processing for MEMS sensor based motion analysis system*. 2016. ISBN 978-91-7485-256-1. Available also from: <http://www.es.mdh.se/publications/4358->.
 42. MBIENTLAB. *MMR* [online]. 2019 [visited on 2020-06-21]. Available from: <https://mbientlab.com/metamotionr/>.
 43. BOSCH, Sensortec. *BMI160* [online] [visited on 2020-06-21]. Available from: <https://www.bosch-sensortec.com/products/motion-sensors/imus/bmi160.html>.
 44. BOSCH, Sensortec. *BMM150* [online] [visited on 2020-06-21]. Available from: <https://www.bosch-sensortec.com/products/motion-sensors/magnetometers-bmm150/>.
 45. MBIENTLAB. *Introduction - MetaWear C++ API 0.17.0 documentation* [online]. 2018 [visited on 2020-06-21]. Available from: <https://mbientlab.com/cppdocs/latest/index.html>.
 46. MBIENTLAB. *Tutorials - MbientLab* [online]. 2019 [visited on 2020-06-22]. Available from: <https://mbientlab.com/tutorials/index.html>.

47. BOSCH, Sensortec. *Sensor Fusion Software* [online] [visited on 2020-06-21]. Available from: <https://www.bosch-sensortec.com/software-tools/software/sensor-fusion-software/>.
48. MBIENTLAB. *Sensor Fusion - MetaWear C++ API 0.17.0 documentation* [online]. 2018 [visited on 2020-06-22]. Available from: https://mbientlab.com/cppdocs/latest/sensor_fusion.html.
49. POIDS ET MESURES, Bureau International des. *The International System of Units (SI)*. 9th ed. Taylor & Francis, 2019. ISBN 978-92-822-2272-0.
50. MATHWORKS. *Detect and replace outliers in data* [online] [visited on 2020-06-25]. Available from: <https://www.mathworks.com/help/matlab/ref/filloutliers.html>. Help Center, Documentation.
51. MATHWORKS. *Savitzky-Golay filter design* [online] [visited on 2020-06-25]. Available from: <https://www.mathworks.com/help/signal/ref/sgolay.html>. Help Center, Documentation.
52. MATHWORKS. *Design digital filters* [online] [visited on 2020-06-25]. Available from: <https://www.mathworks.com/help/signal/ref/designfilt.html>. Help Center, Documentation.
53. MATHWORKS. *Butterworth filter design* [online] [visited on 2020-06-25]. Available from: <https://www.mathworks.com/help/signal/ref/butter.html>. Help Center, Documentation.
54. MATHWORKS. *High-pass filter signals* [online] [visited on 2020-06-25]. Available from: <https://www.mathworks.com/help/signal/ref/highpass.html>. Help Center, Documentation.
55. MATHWORKS. *IMU simulation model* [online] [visited on 2020-06-25]. Available from: <https://www.mathworks.com/help/fusion/ref/imusensor-system-object.html>. Help Center, Documentation.
56. MATHWORKS. *Rate-driven trajectory generator* [online] [visited on 2020-06-25]. Available from: <https://www.mathworks.com/help/nav/ref/kinematictrajectory-system-object.html>. Help Center, Documentation.
57. MATHWORKS. *What is the IMU Sensor measuring in the Sensor Fusion and Tracking Toolbox?* [online] [visited on 2020-06-25]. Available from: <https://www.mathworks.com/matlabcentral/answers/486192-what-is-the-imu-sensor-measuring-in-the-sensor-fusion-and-tracking-toolbox>. Matlab Answers.

58. GRUNNET-JEPSEN, Anders; HARVILLE, Michael; FULKERSON, Brian; PIRO, Daniel; BROOK, Shirit; RADFORD, Jim. *Introduction to Intel RealSense Visual SLAM and the T265 Tracking Camera* [online] [visited on 2020-07-01]. Available from: <https://dev.intelrealsense.com/docs/intel-realsensetm-visual-slam-and-the-t265-tracking-camera>. RealSense Documentation.
59. DURRANT-WHYTE, Hugh; BAILEY, Tim. Simultaneous localization and mapping: Part I. *Robotics & Automation Magazine, IEEE*. 2006, vol. 13, pp. 99–110. Available from DOI: 10.1109/MRA.2006.1638022.
60. INTEL, Corporation. *Intel RealSense Tracking Camera - Datasheet* [online]. 4th ed. 2019.
61. INTEL, Corporation. *Intel RealSense SDK* [online] [visited on 2020-07-02]. Available from: <https://github.com/IntelRealSense/librealsense>.
62. INTEL, Corporation. *Intel RealSense Viewer* [online] [visited on 2020-07-02]. Available from: <https://github.com/IntelRealSense/librealsense/tree/master/tools/realsense-viewer>. librealsense.
63. INTEL, Corporation. *re-data-collect Tool* [online] [visited on 2020-07-02]. Available from: <https://github.com/IntelRealSense/librealsense/tree/master/tools/data-collect>. librealsense.
64. VIDULLAN. *Downward facing T265 #4080* [online]. 2019 [visited on 2020-07-02]. Available from: <https://github.com/IntelRealSense/librealsense/issues/4080>. librealsense, Issues.
65. SPONG, M.W.; HUTCHINSON, S.; VIDYASAGAR, M. *Robot Modeling and Control*. 2nd ed. John Wiley Sons, 2020. ISBN 9781119523994.
66. BUSS, Samuel R. *Inverse kinematics demos* [online] [visited on 2020-07-19]. Available from: <https://www.math.ucsd.edu/~sbuss/ResearchWeb/ikmethods/index.html>.
67. ERWIN COUMANS, Yunfei Bai. *PyBullet Quickstart Guide* [online]. 2016-2020 [visited on 2020-07-19]. Available from: <https://docs.google.com/document/d/10sXEhzFRSvFc13XxNGhd4N2SedqwdAvK3dsihxVUA/edit#heading=h.2ye70wns7io3>.
68. OPEN, RAVE. *IKFast: The Robot Kinematics Compiler* [online] [visited on 2020-07-19]. Available from: http://openrave.org/docs/latest_stable/openravepy/ikfast/#ikfast-the-robot-kinematics-compiler.
69. BUSS, Samuel. Introduction to inverse kinematics with Jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Transactions in Robotics and Automation*. 2004, vol. 17.

70. DIANKOV, Rosen. *Automated Construction of Robotic Manipulation Programs*. Pittsburgh, Pennsylvania: Carnegie Mellon University, 2010.
71. OPEN, Robotics. *About ROS* [online] [visited on 2020-07-19]. Available from: <https://www.ros.org/about-ros/>.
72. KUKA. *Sensitive robotics_LBR iiwa* [online] [visited on 2020-07-19]. Available from: https://www.kuka.com/-/media/kuka-downloads/imported/9cb8e311bfd744b4b0eab25ca883f6kuka_lbr_iiwa_brochure_en.pdf?rev=7141546264dd45ca9f613adbe6393405.PB0004/EN/4/0317
73. ROS.ORG. *descartes* [online] [visited on 2020-07-19]. Available from: <http://wiki.ros.org/descartes>.
74. MOVEIT, ROS. *MoveIt* [online] [visited on 2020-07-19]. Available from: <https://moveit.ros.org/>.
75. ROS-INDUSTRIAL. *descartes_tutorials* [online] [visited on 2020-07-19]. Available from: https://github.com/ros-industrial-consortium/descartes_tutorials.
76. ABB. *IRB 2400* [online] [visited on 2020-07-19]. Available from: <https://new.abb.com/products/robotics/industrial-robots/irb-2400>.
77. DE MAEYER, J.; MOYAERS, B.; DEMEESTER, E. Cartesian path planning for arc welding robots: Evaluation of the descartes algorithm. 2017, pp. 1–8.
78. ŠMEJKAL, Ladislav. *PLC a automatizace*. Praha: BEN - technická literatura, 2005. ISBN 80-7300-087-3.
79. OPENPLC. *The OpenPLC project* [online] [visited on 2020-06-26]. Available from: <https://openplcproject.com/>.



Appendix A

Motion tracking measurements results

 **A.1 Laser Trackers**

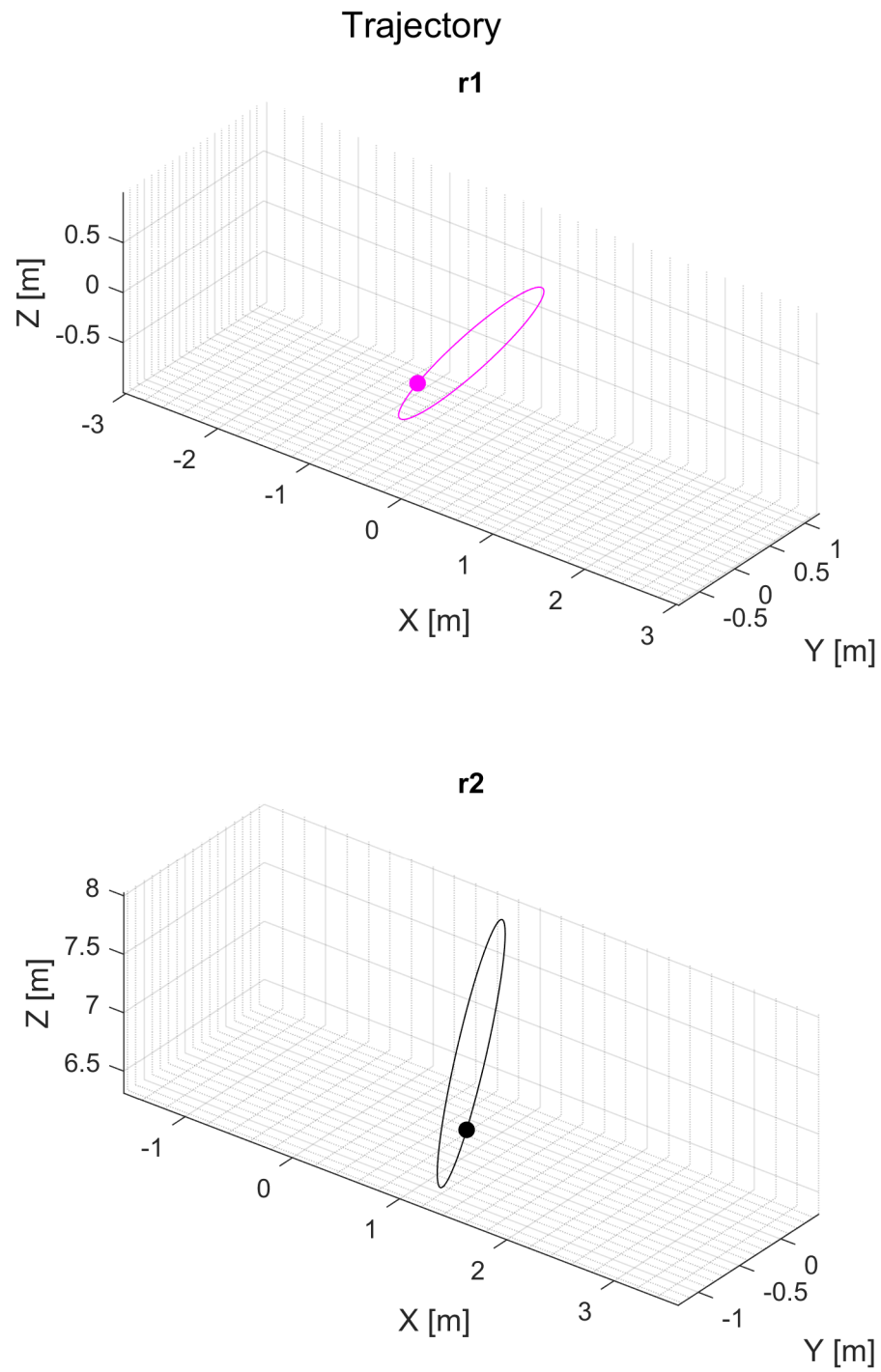


Figure A.1: Laser tracker simulation, dynamic calibration by circular movement, trajectory.

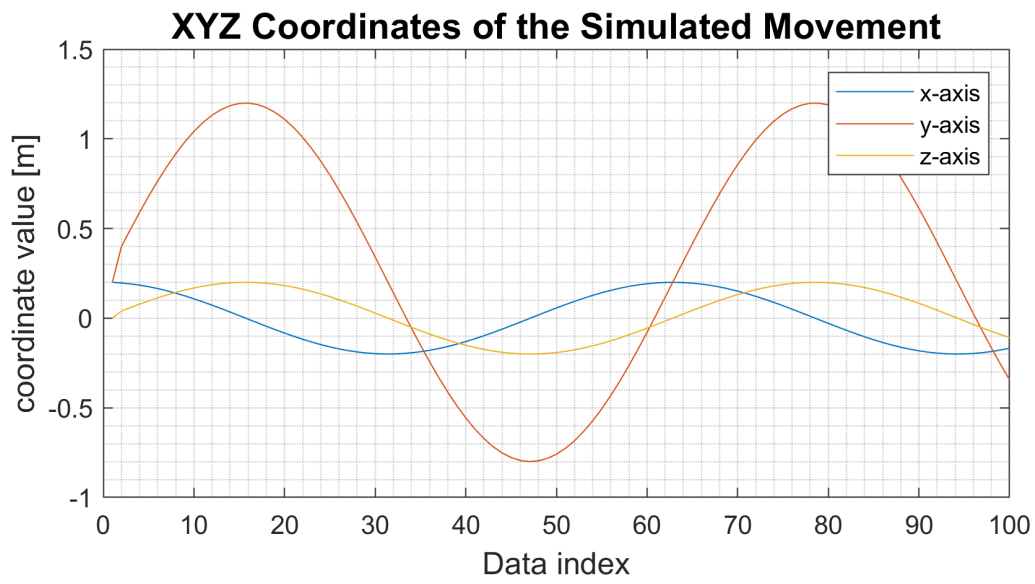


Figure A.2: Laser tracker simulation, dynamic calibration by circular movement, position vector.

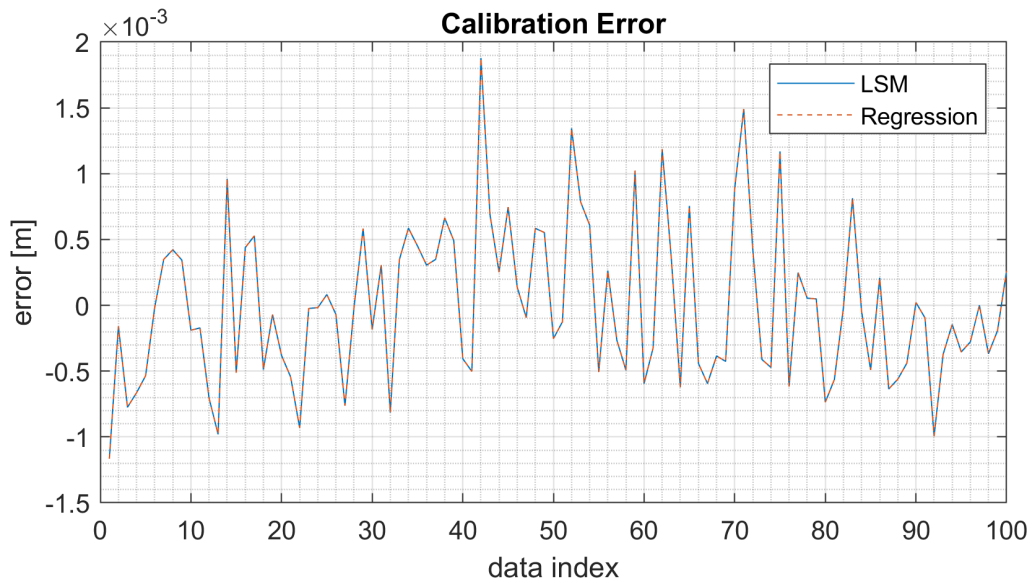


Figure A.3: Laser tracker simulation, dynamic calibration by circular movement with maximum noise, calibration error.

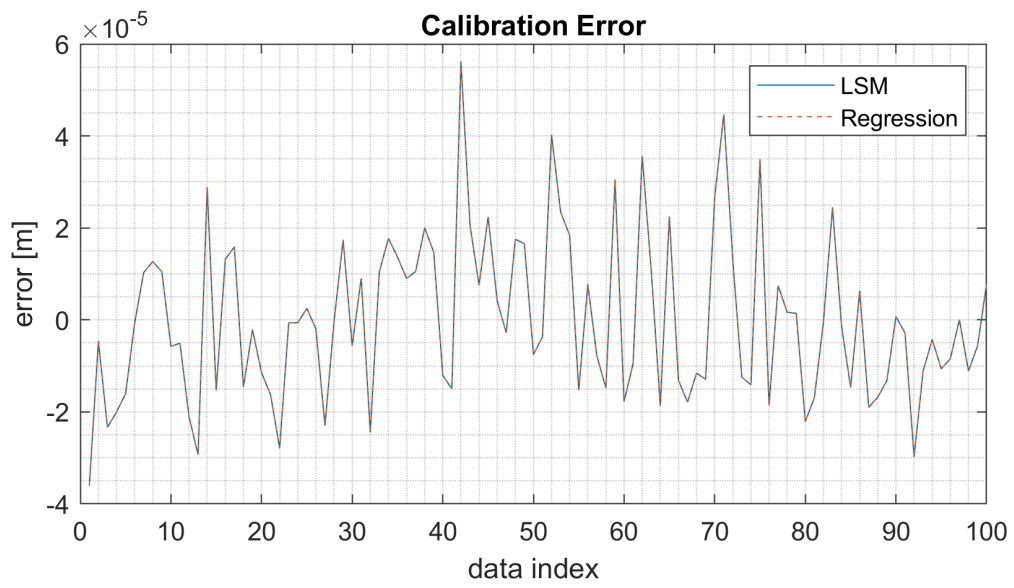


Figure A.4: Laser tracker simulation, dynamic calibration by circular movement with minimum noise, calibration error.

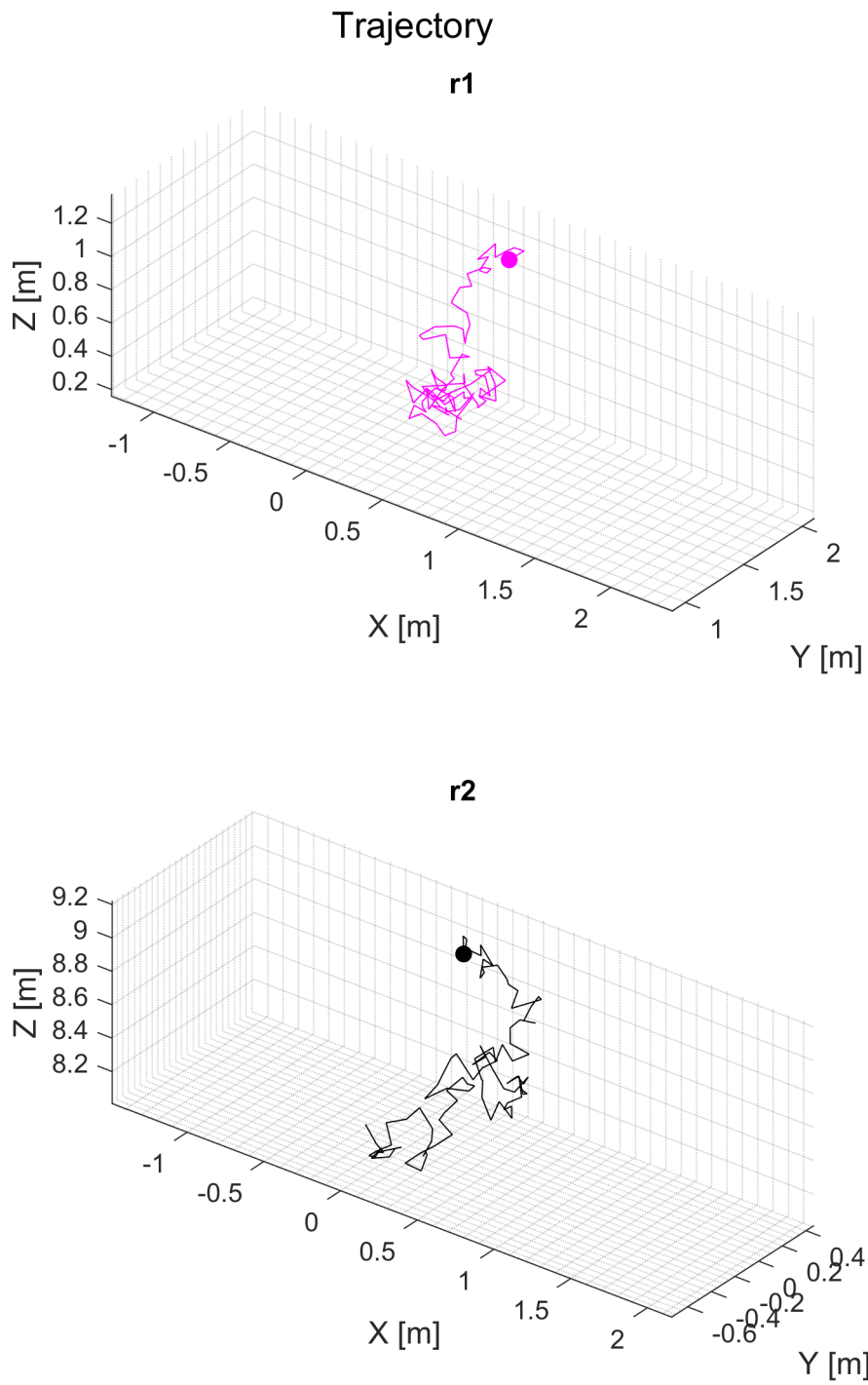


Figure A.5: Laser tracker simulation, dynamic calibration by random movement, trajectory.

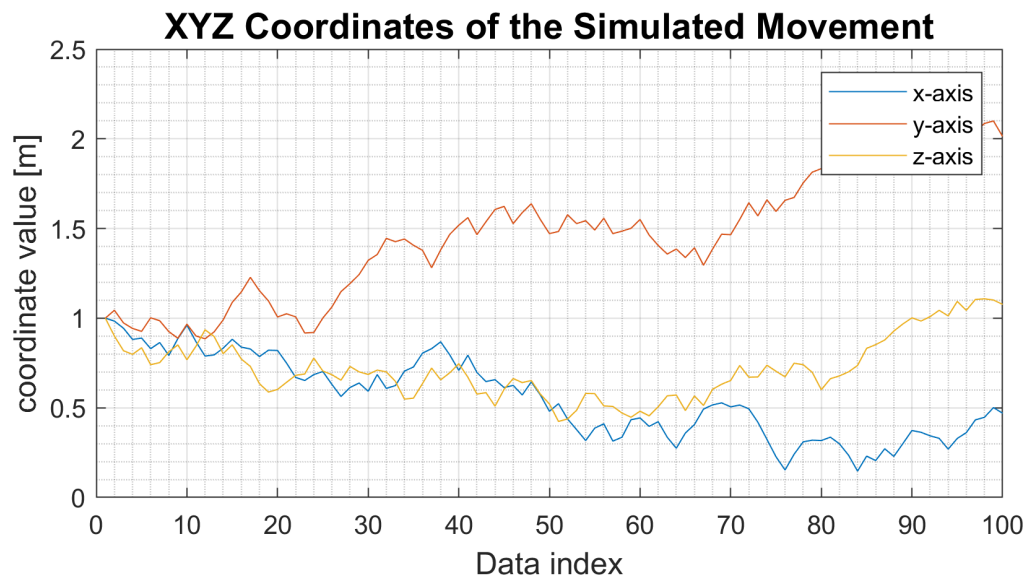


Figure A.6: Laser tracker simulation, dynamic calibration by random movement, position vector.

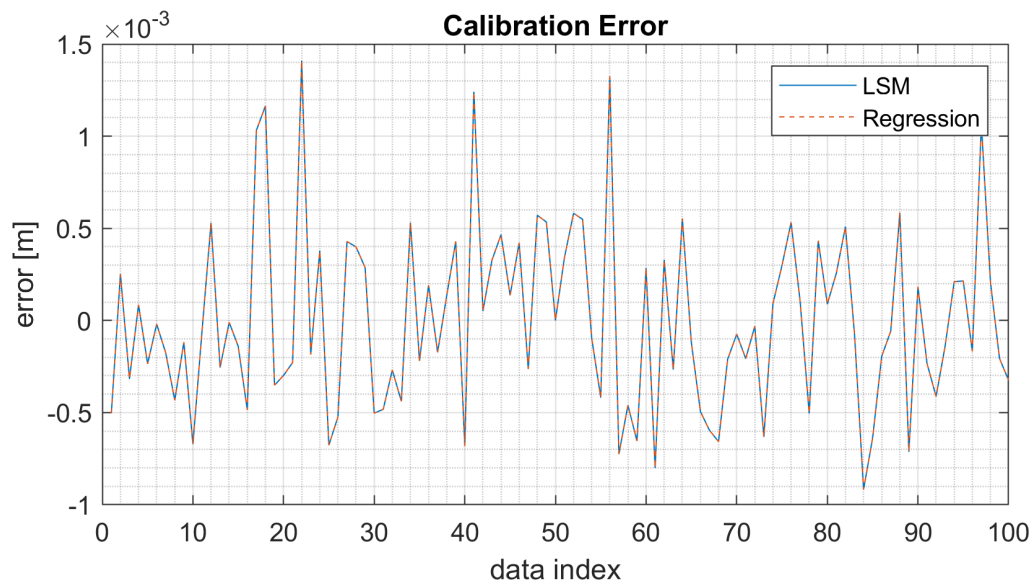


Figure A.7: Laser tracker simulation, dynamic calibration by random movement with maximum noise, calibration error.

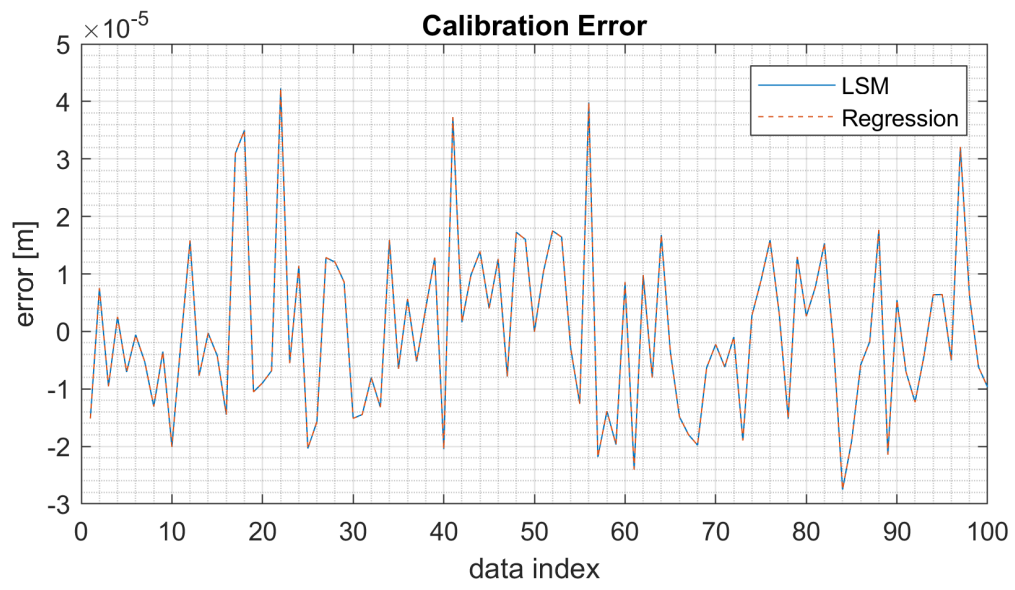


Figure A.8: Laser tracker simulation, dynamic calibration by random movement with minimum noise, calibration error.

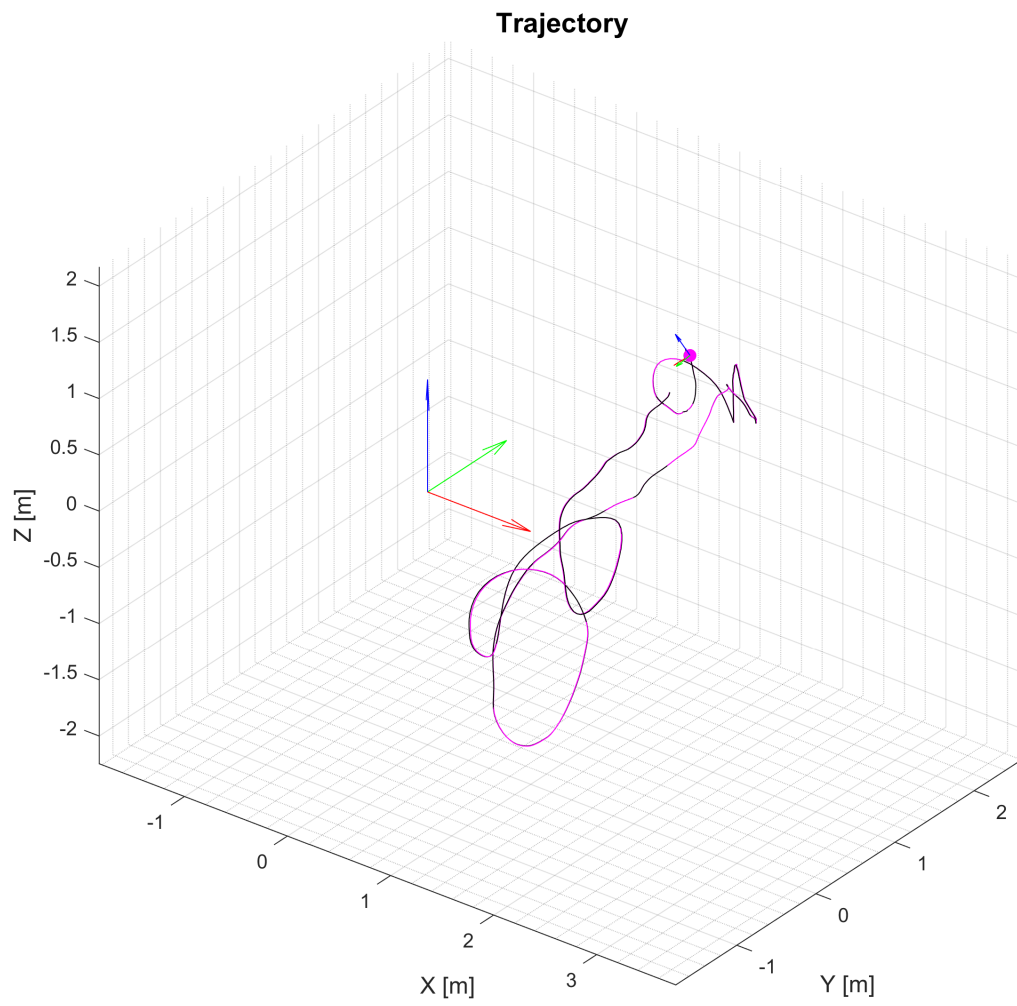


Figure A.9: Laser tracker measurements, dynamic calibration, trajectory.

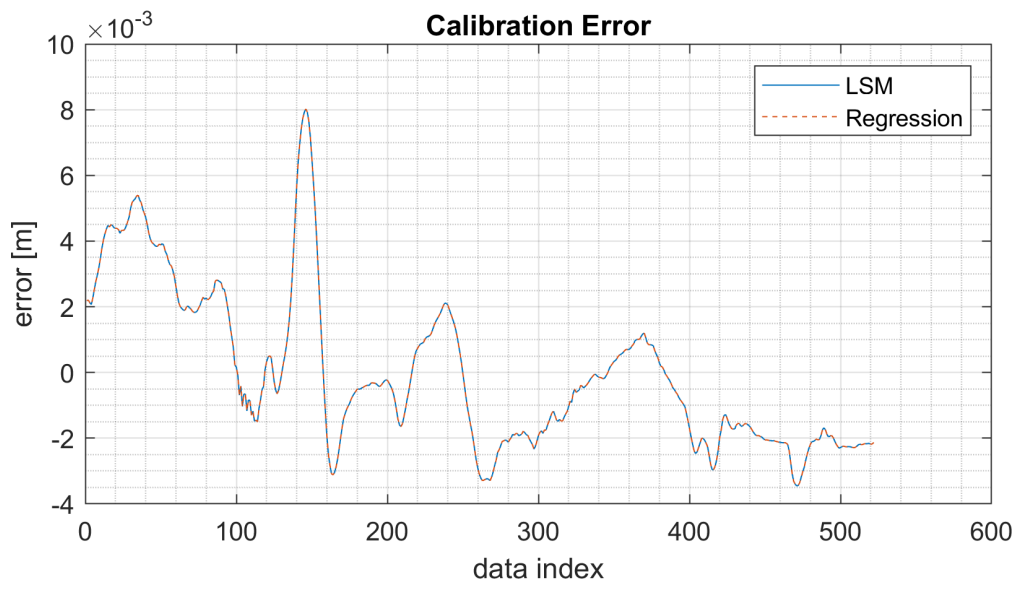


Figure A.10: Laser tracker measurements, dynamic calibration, calibration error.

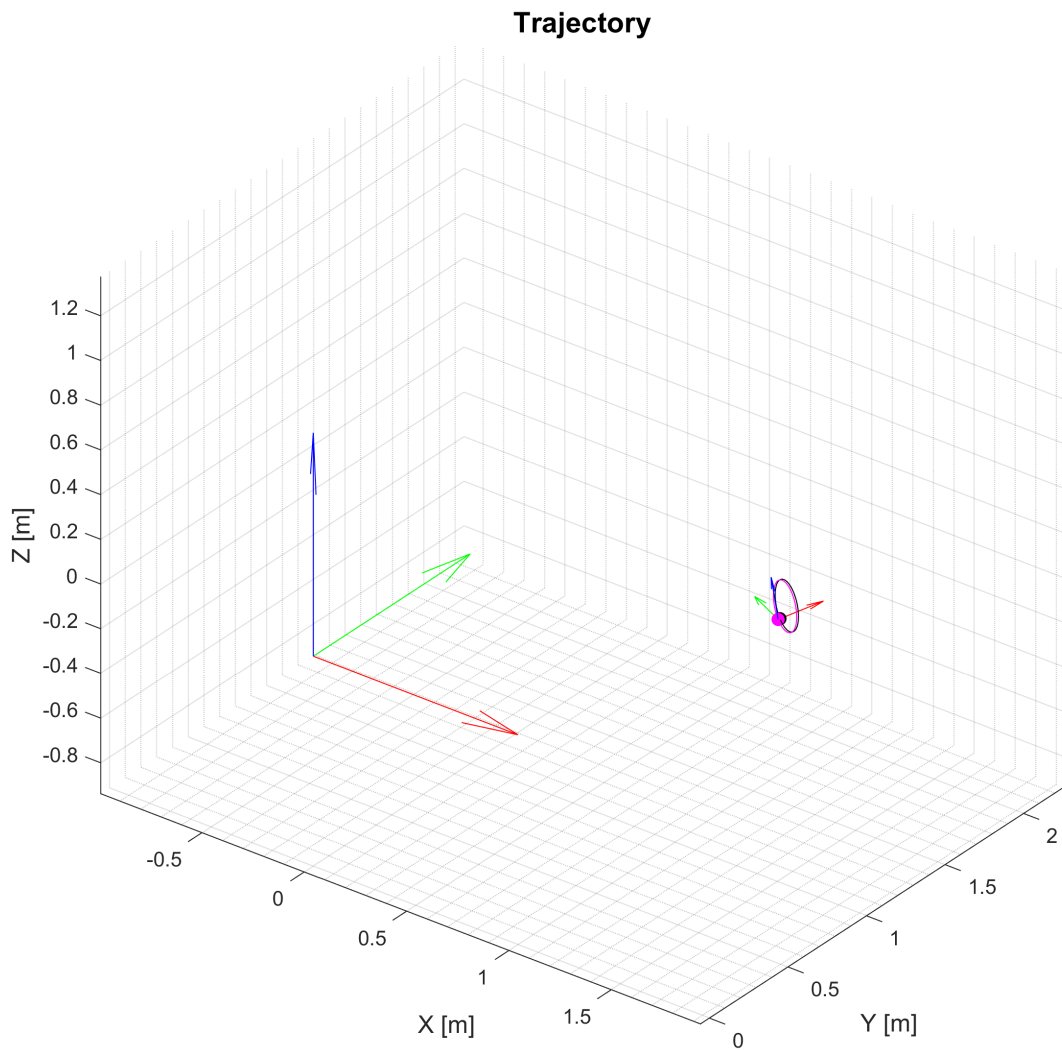


Figure A.11: Laser tracker measurements, circular movement with common reflector, trajectory. Data based on dynamic calibration are not coincident.

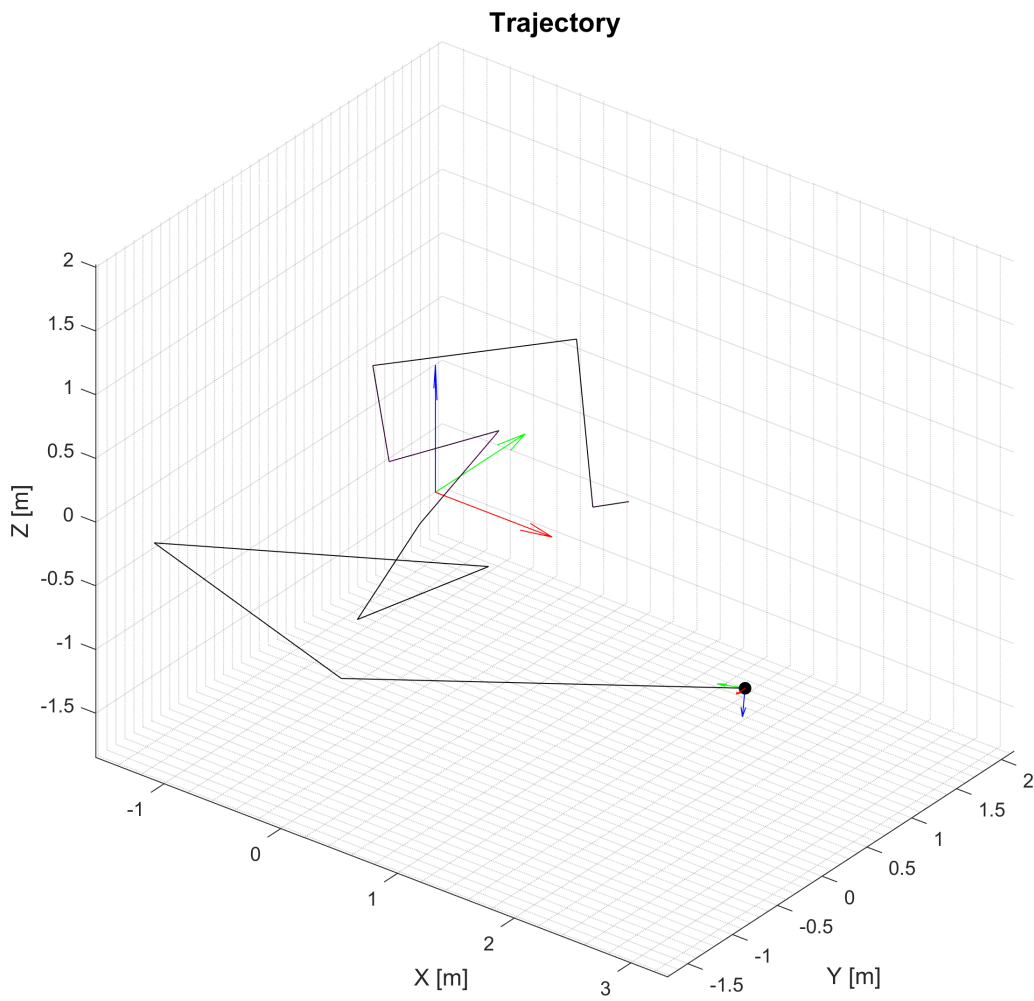


Figure A.12: Laser tracker measurements, static calibration, trajectory.

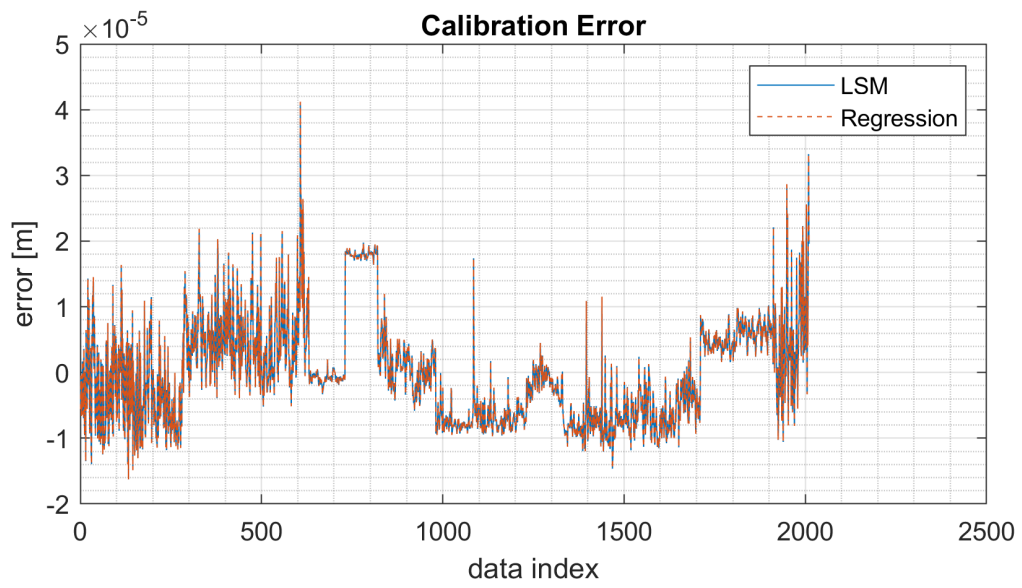


Figure A.13: Laser tracker measurements, static calibration, calibration error.

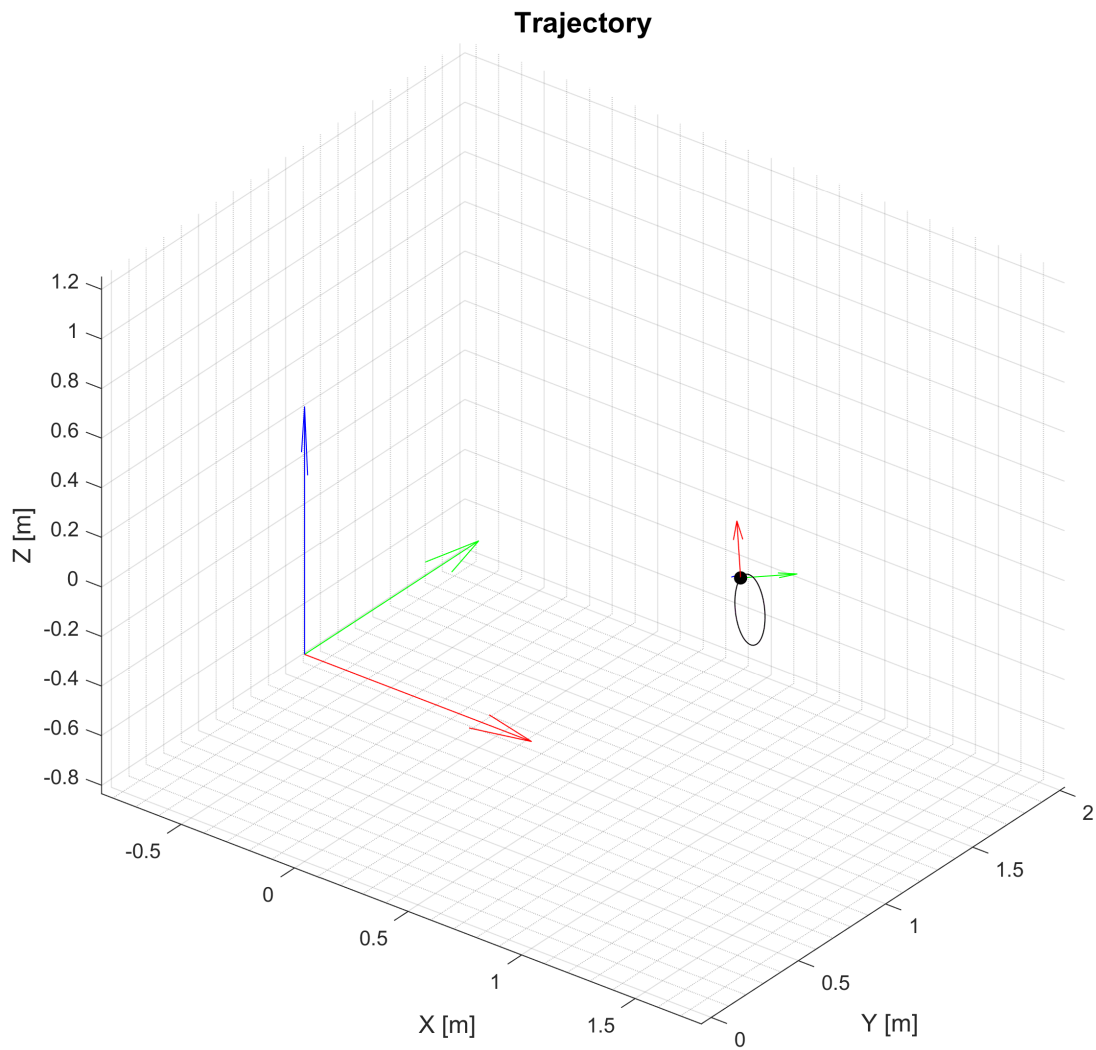


Figure A.14: Laser tracker measurements, circular movement with common reflector, trajectory. Data based on static calibration are coincident.

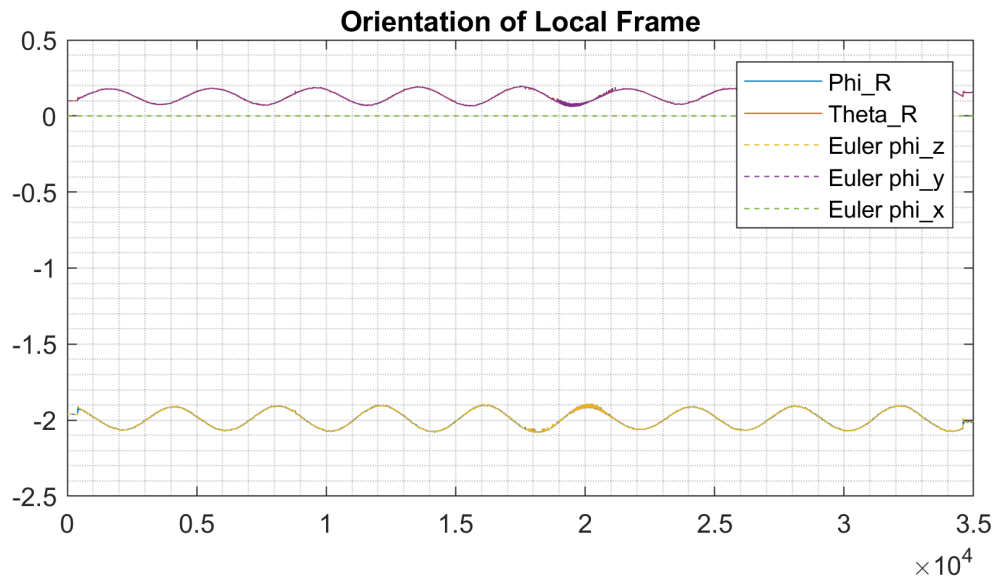


Figure A.15: Laser tracker measurements, circular movement with common reflector, orientation. Rotation around x-axis is not defined.

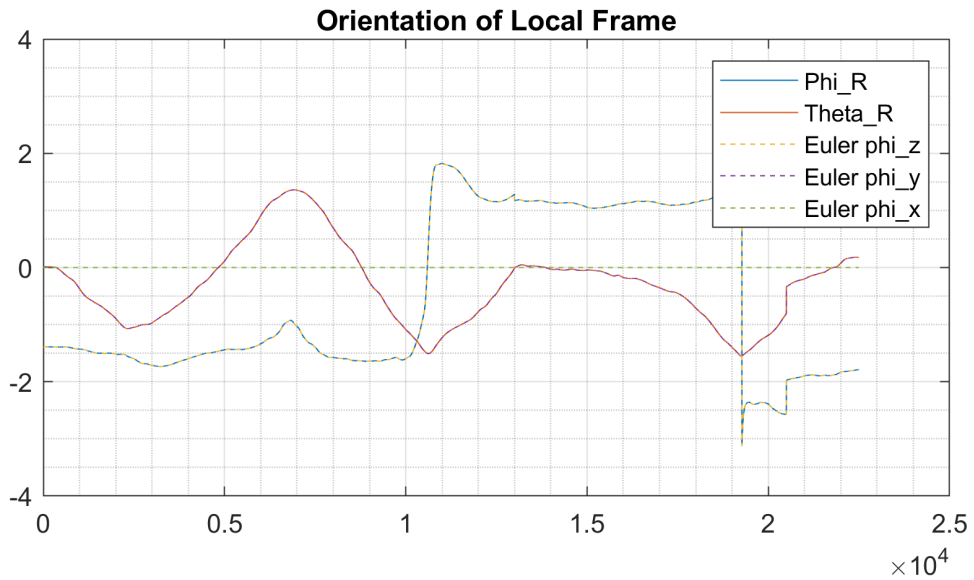


Figure A.16: Laser tracker measurements, random movement, orientation. Rotation around x-axis is not defined.

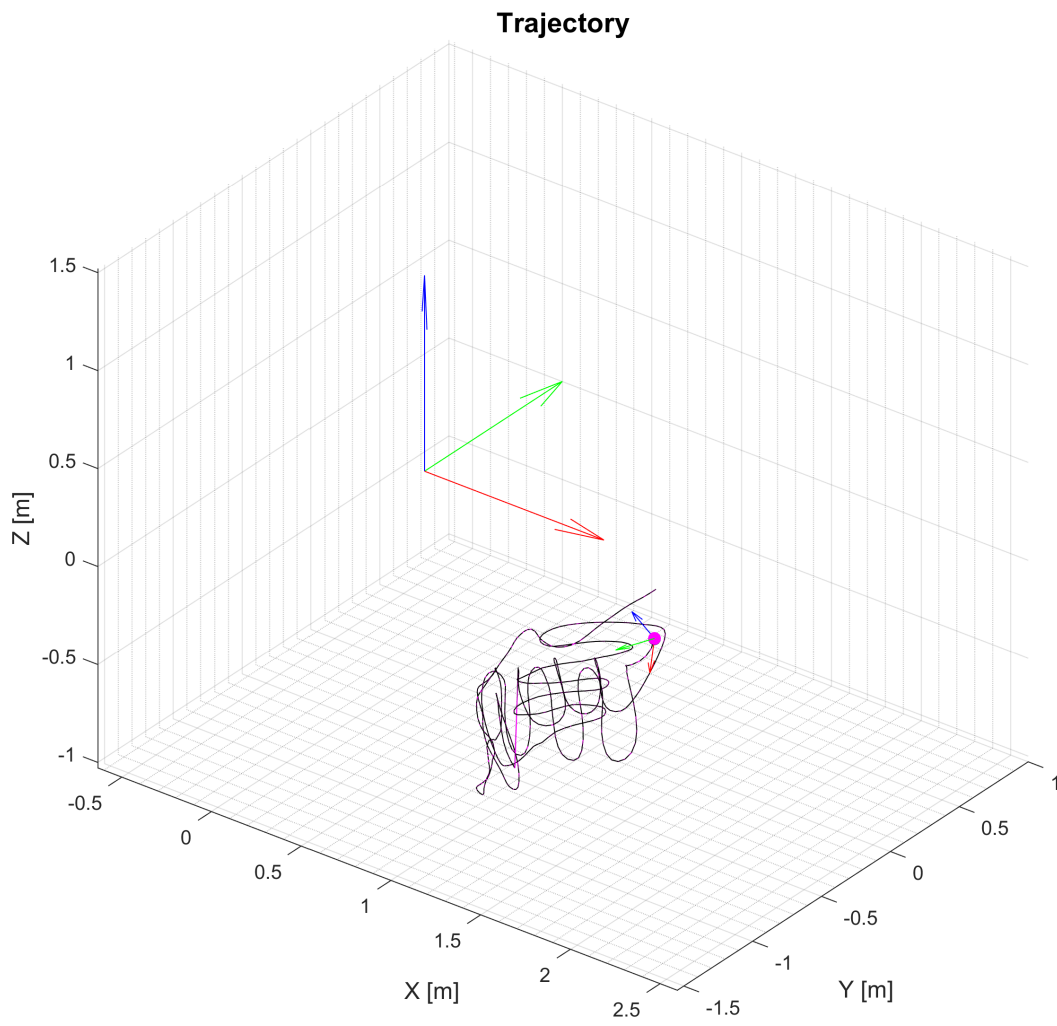


Figure A.17: Laser tracker measurements, random movement 1 with common reflector, trajectory.

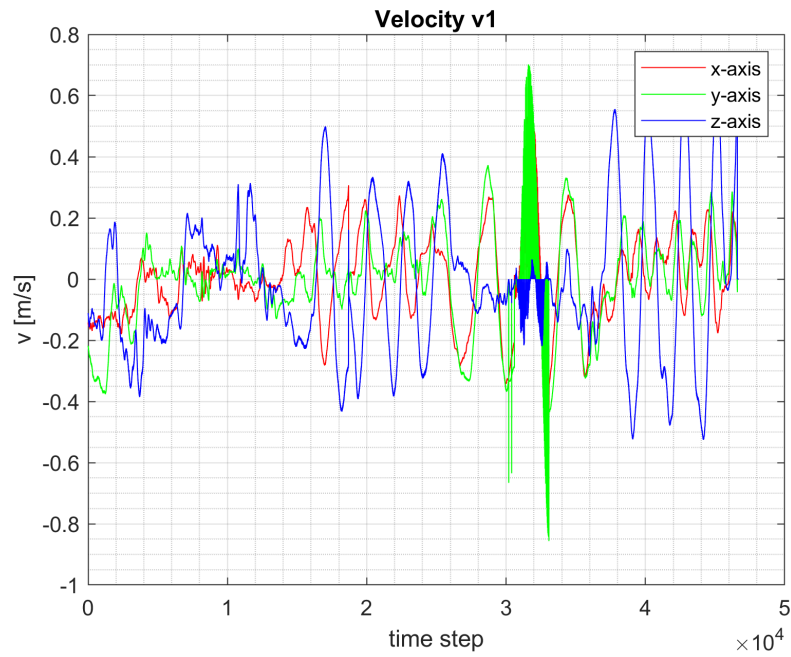


Figure A.18: Laser tracker measurements, random movement 1, velocity measured by tracker 1. The dense parts of the diagram are oscillations due to laser tracker's loss of signal.

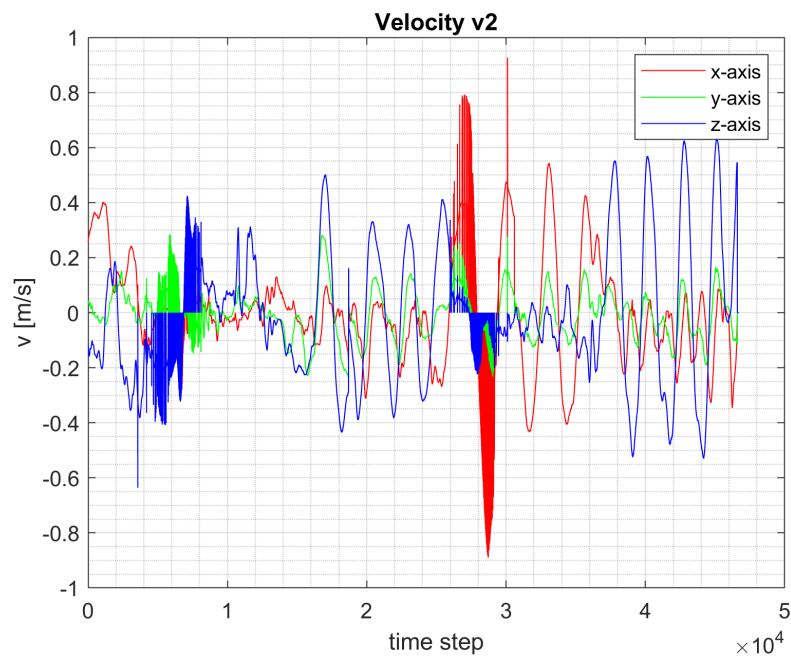


Figure A.19: Laser tracker measurements, random movement 1, velocity measured by tracker 2. The dense parts of the diagram are oscillations due to laser tracker's loss of signal.

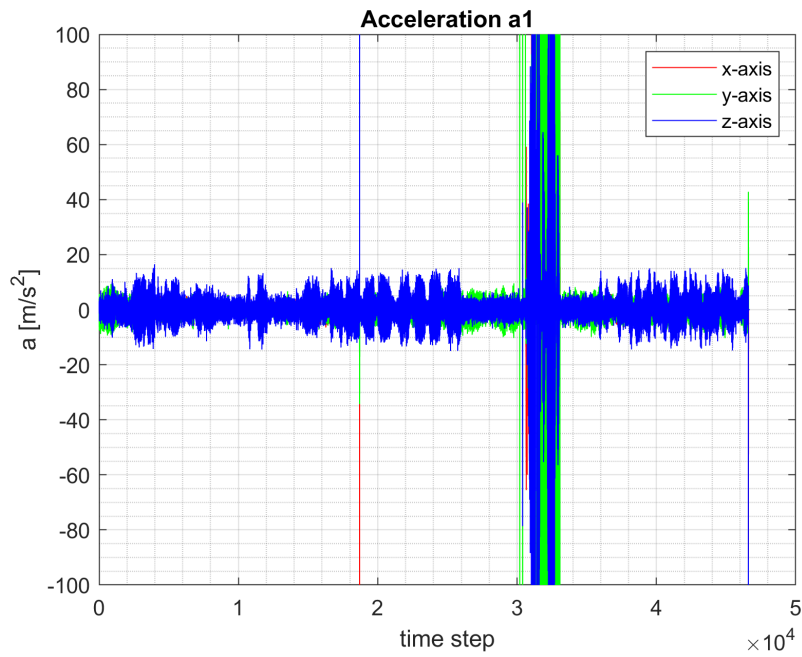


Figure A.20: Laser tracker measurements, random movement 1, acceleration measured by tracker 1. The dense parts of the diagram are oscillations due to laser tracker's loss of signal.

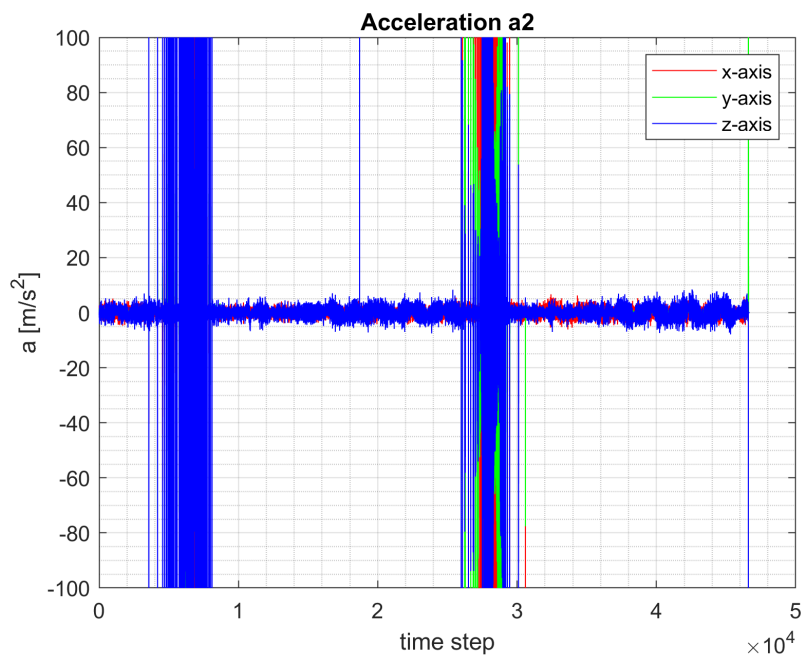


Figure A.21: Laser tracker measurements, random movement 1, acceleration measured by tracker 2. The dense parts of the diagram are oscillations due to laser tracker's loss of signal.

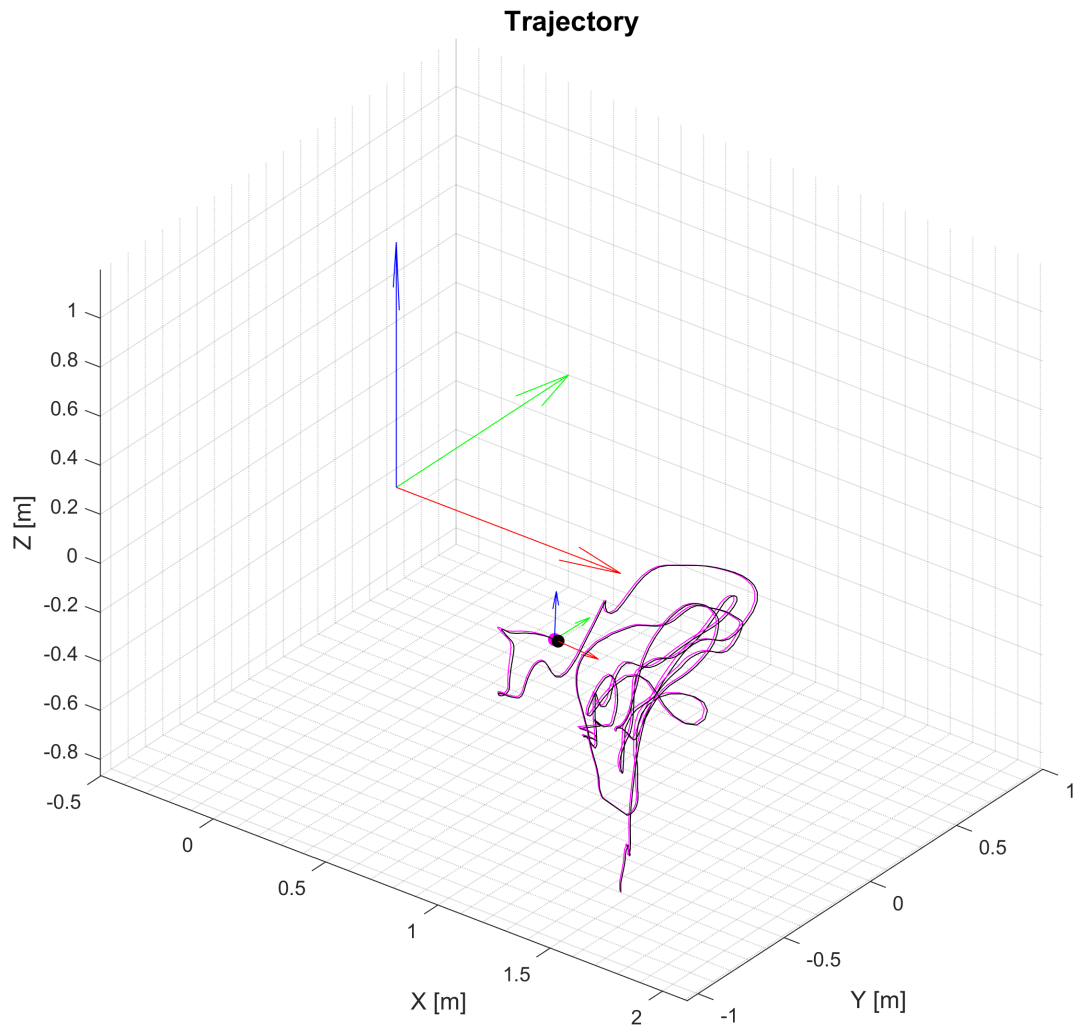


Figure A.22: Laser tracker measurements, random movement 2 with common reflector, trajectory.

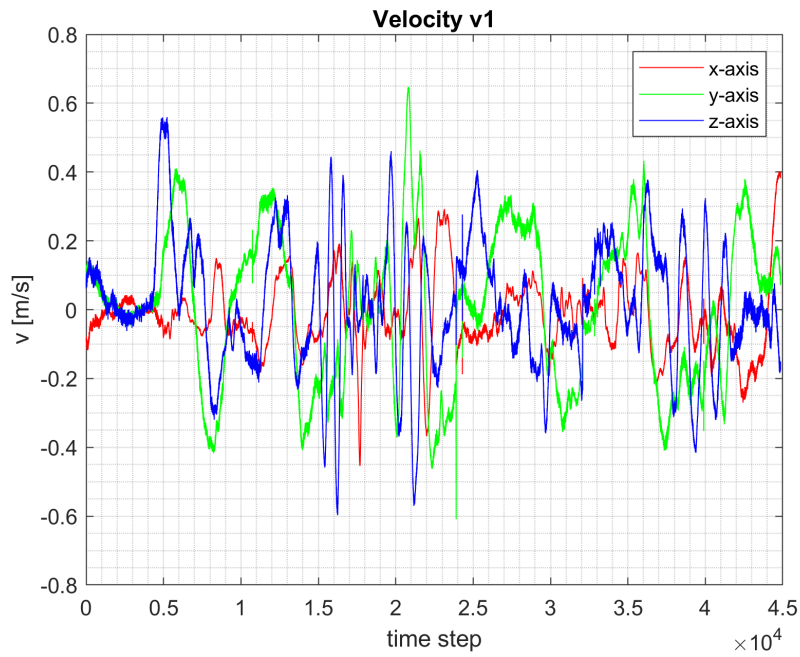


Figure A.23: Laser tracker measurements, random movement, velocity measured by tracker 1.

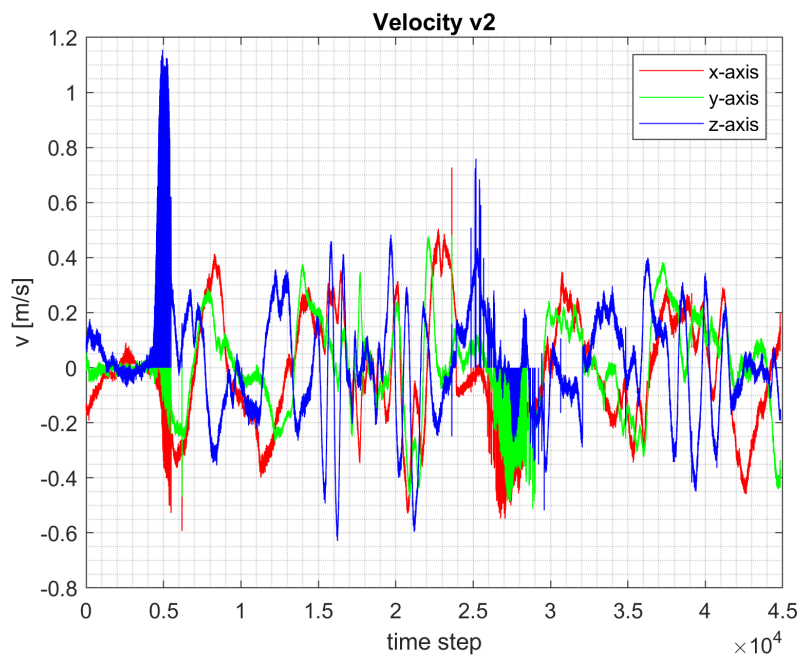


Figure A.24: Laser tracker measurements, random movement, velocity measured by tracker 2. The dense parts of the diagram are oscillations due to laser tracker's loss of signal.

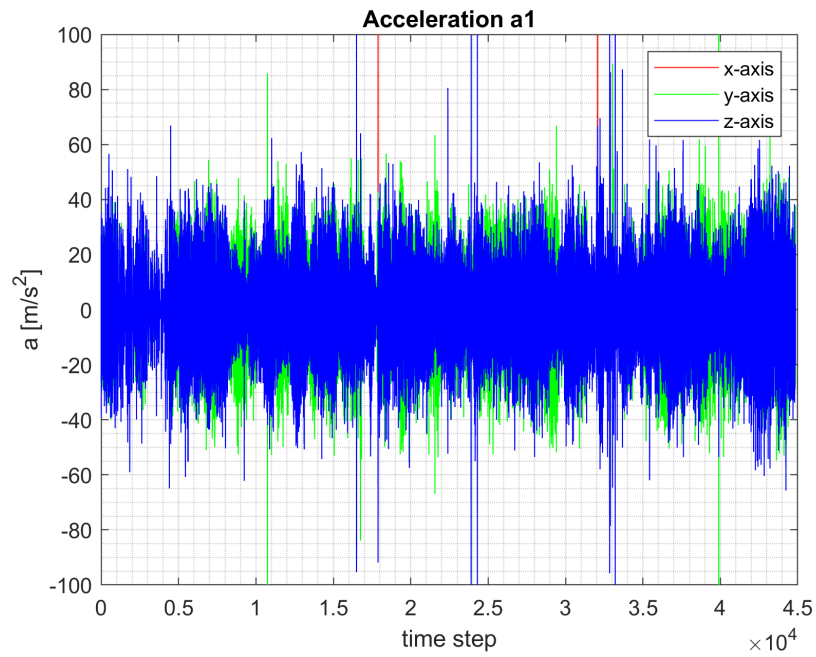


Figure A.25: Laser tracker measurements, random movement, acceleration measured by tracker 1.

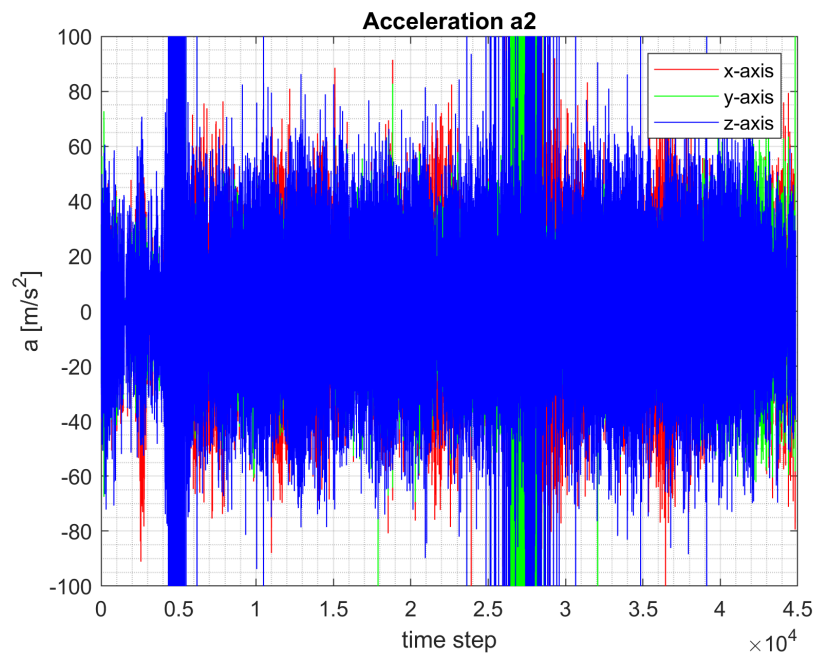


Figure A.26: Laser tracker measurements, random movement, acceleration measured by tracker 2. The dense parts of the diagram are oscillations due to laser tracker's loss of signal.

A.2 MMR IMU sensor

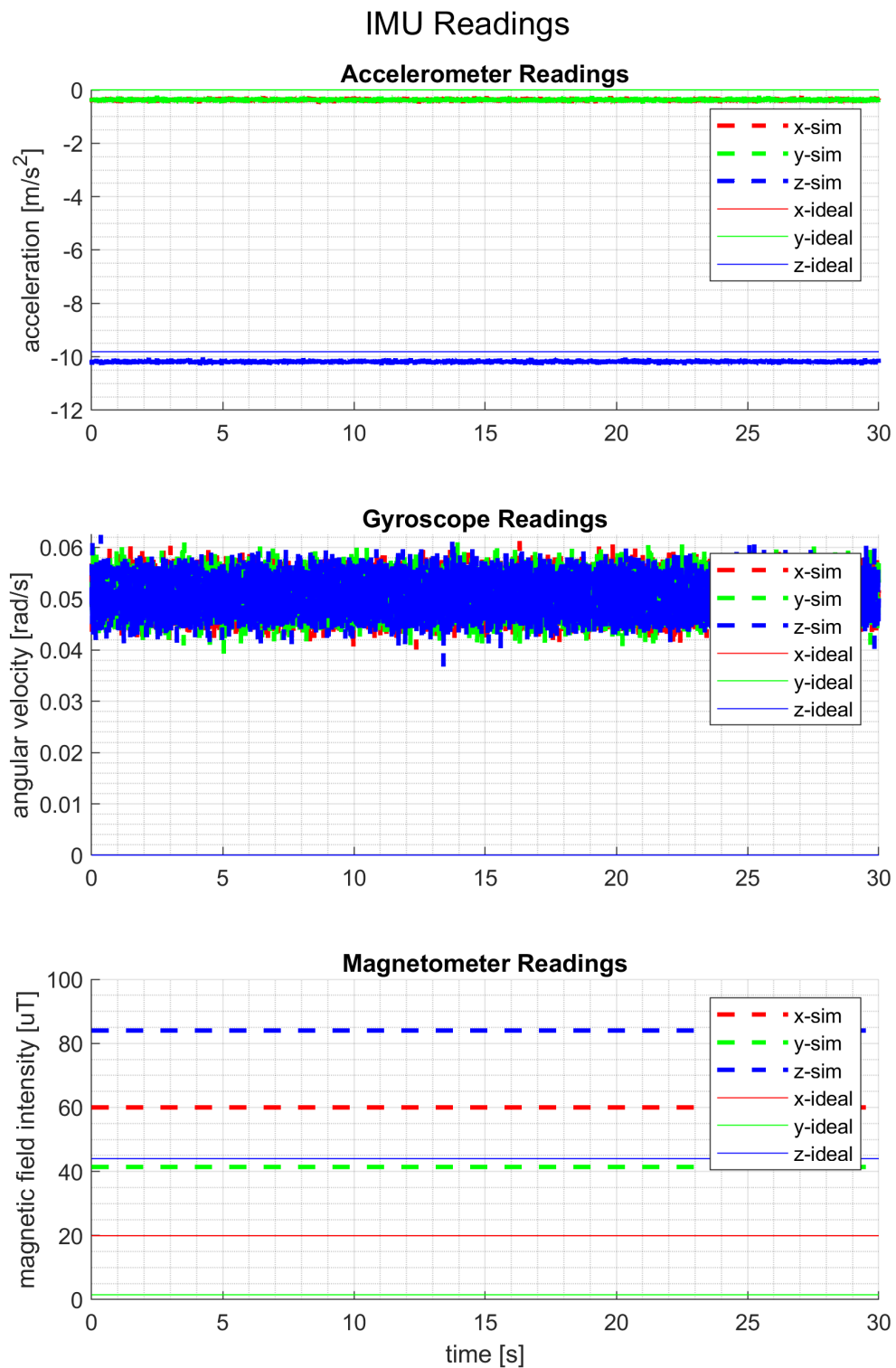


Figure A.27: MMR and ideal IMU simulations, steady state, sensor readings. Deviation of MMR data from an ideal IMU is visible.

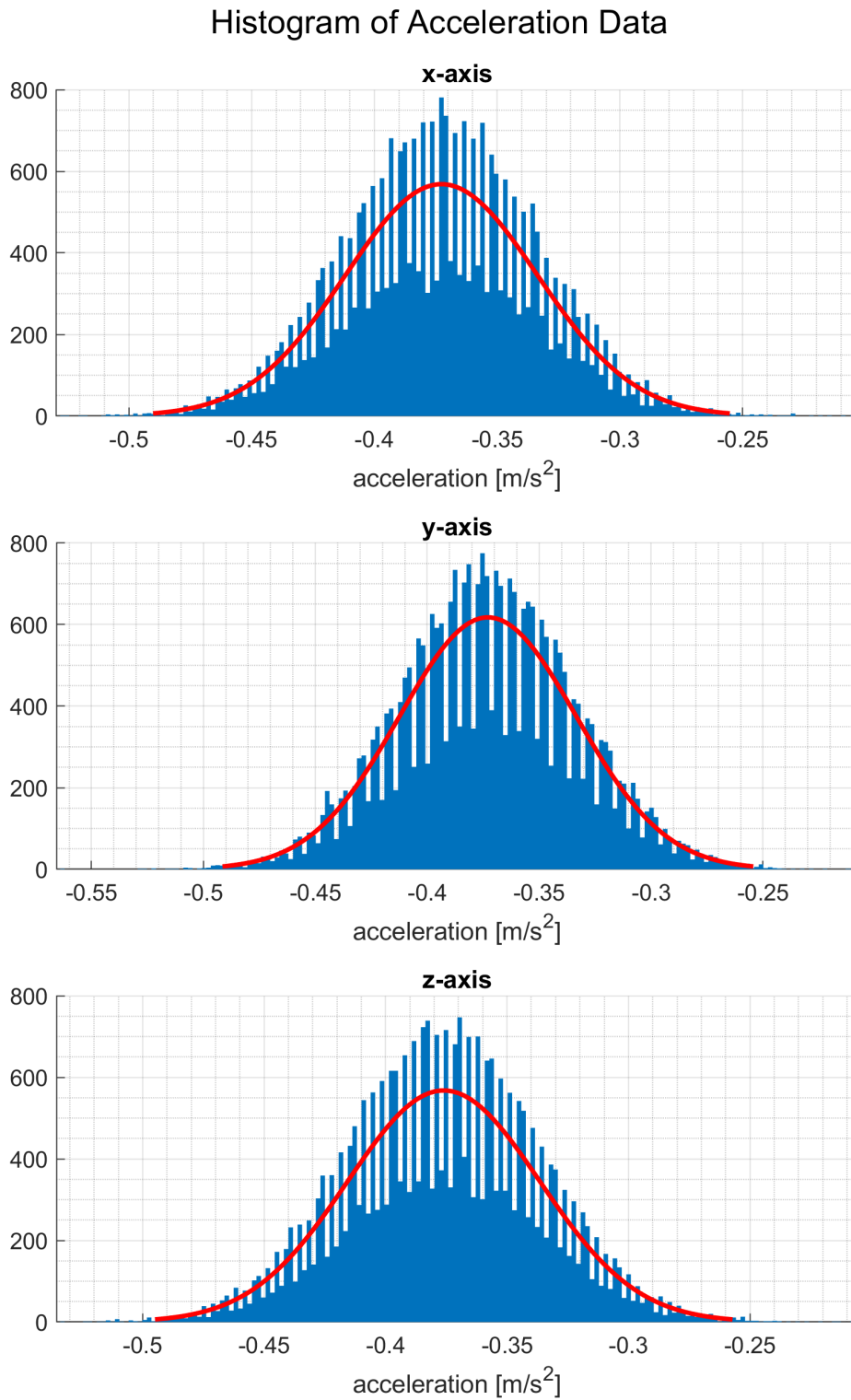


Figure A.28: MMR IMU simulation, steady state, histogram of measured acceleration with Gaussian fit. Offset from ideal zero measurements is shown, noise appears to be Gaussian.

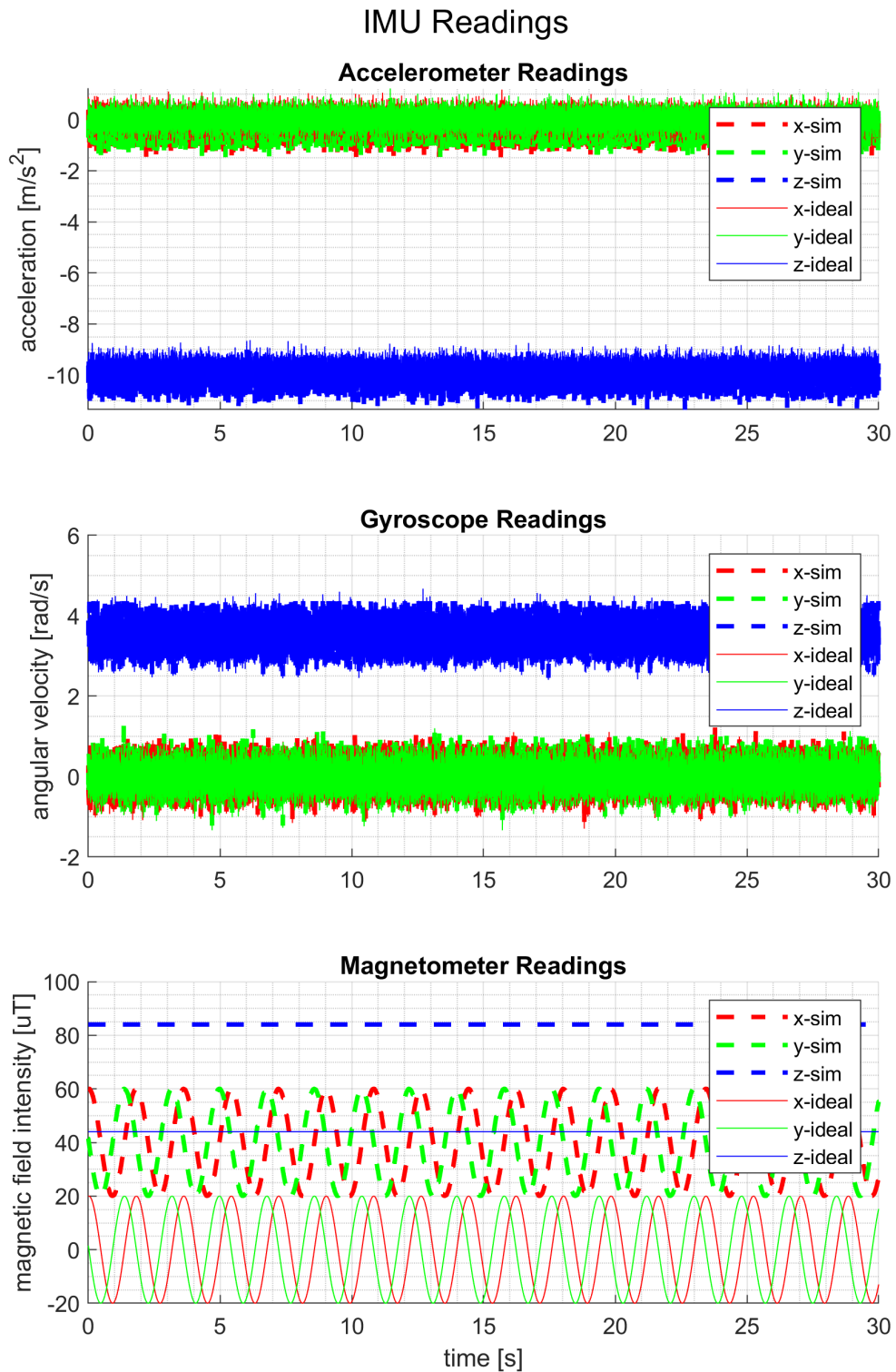


Figure A.29: MMR and ideal IMU simulations, rotation about z-axis, with added Gaussian noise, sensor readings. Deviation of MMR data from an ideal IMU is visible.

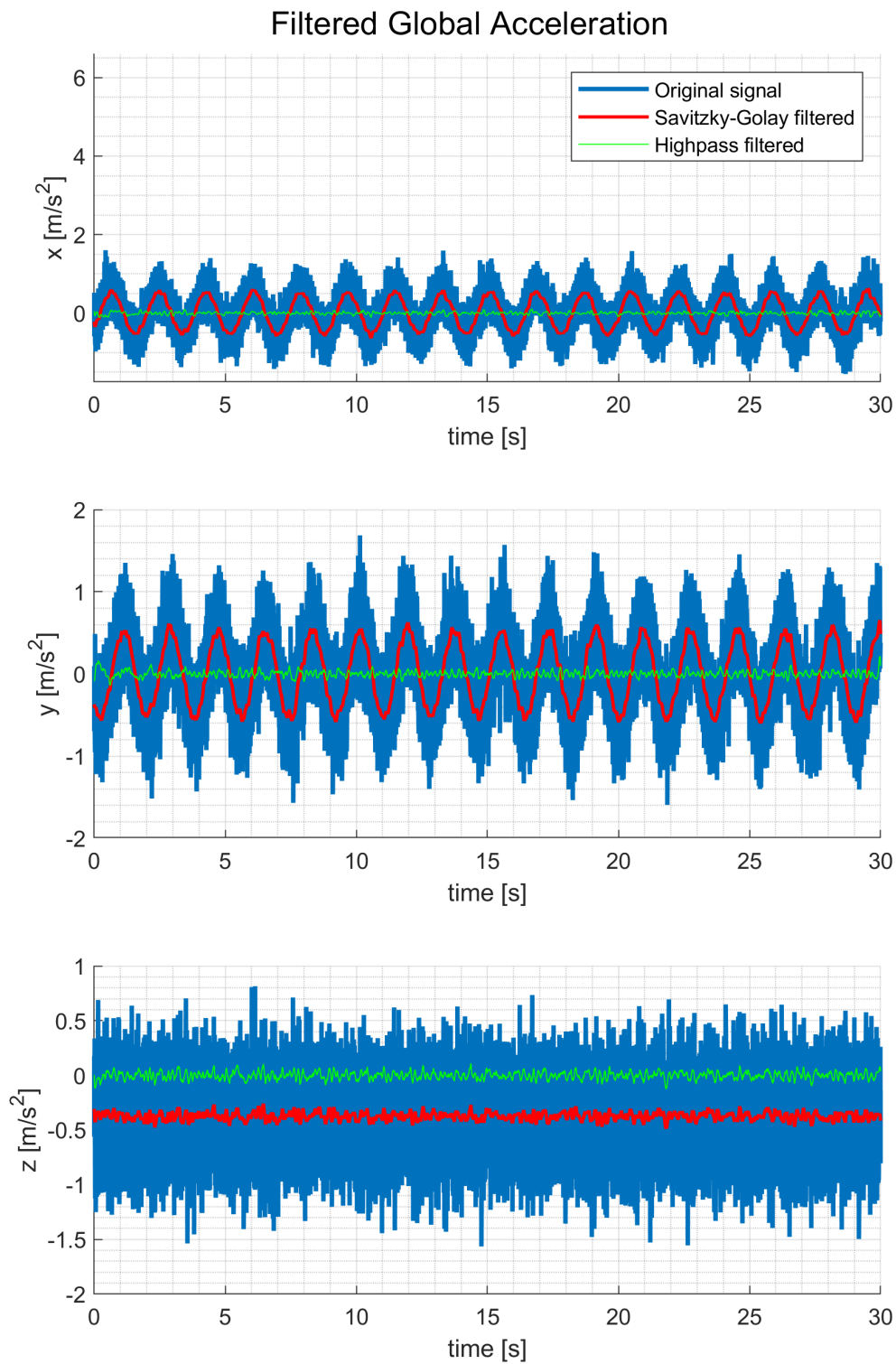


Figure A.30: MMR IMU simulation, rotation about z-axis, with added Gaussian noise, global acceleration. Undesirable oscillation in x- and y-axis is visible, possibly filtered out.

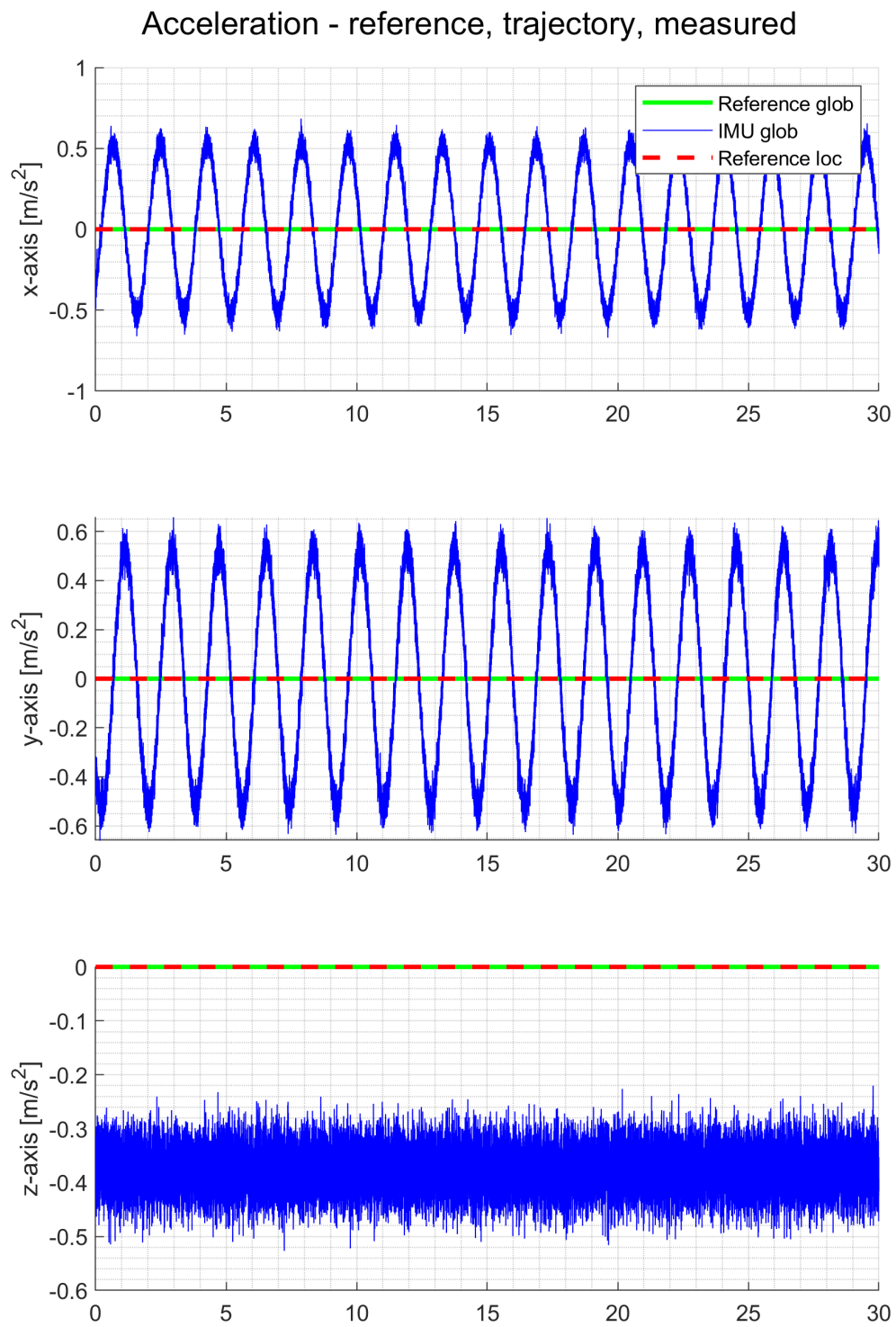


Figure A.31: MMR IMU simulation, rotation about z-axis, with added Gaussian noise, acceleration. Undesirable oscillation in x- and y-axis is visible.

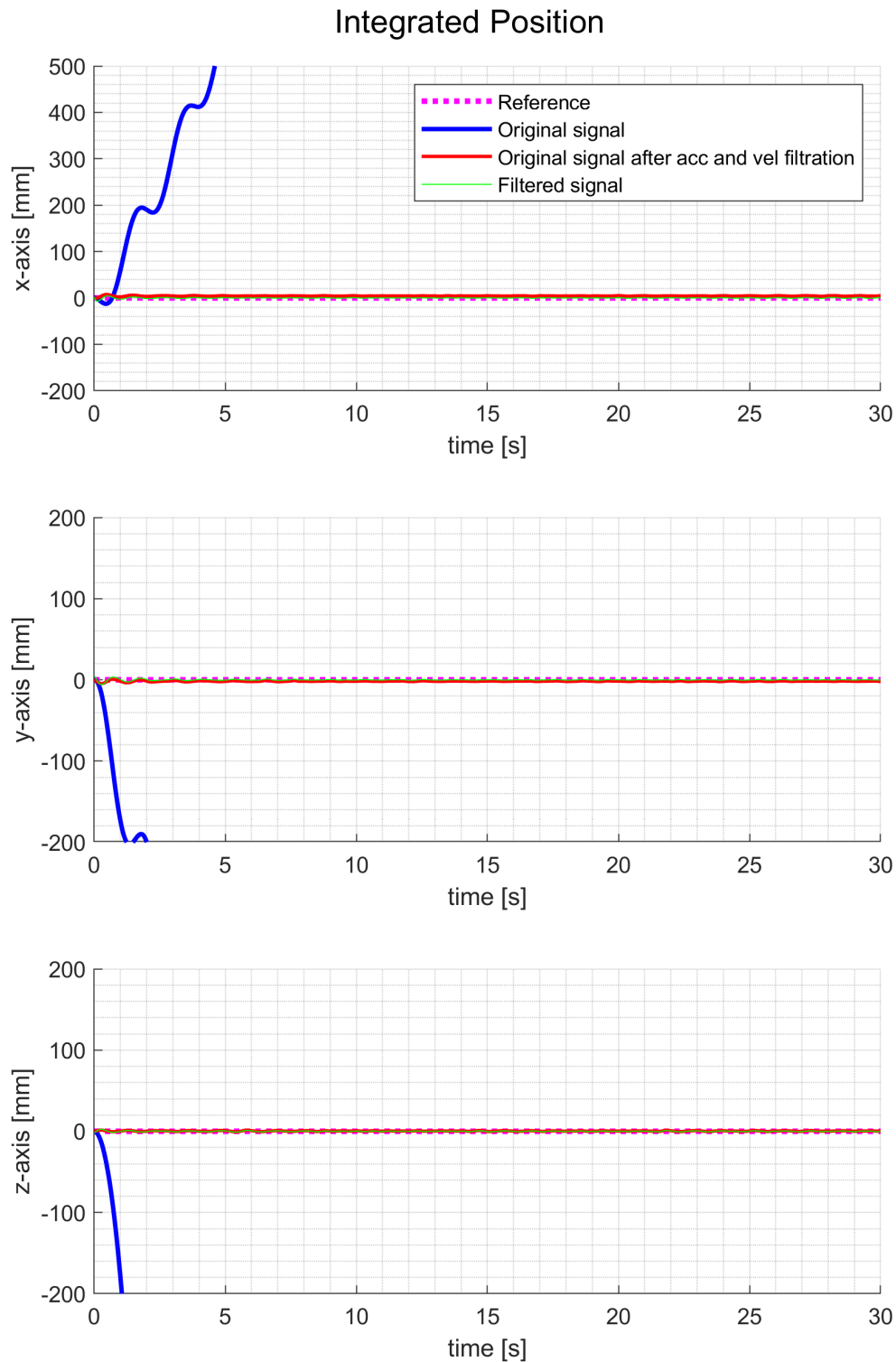


Figure A.32: MMR IMU simulation, rotation about z-axis, with added Gaussian noise, position. Original signal diverges, but coincides with reference when filtered.

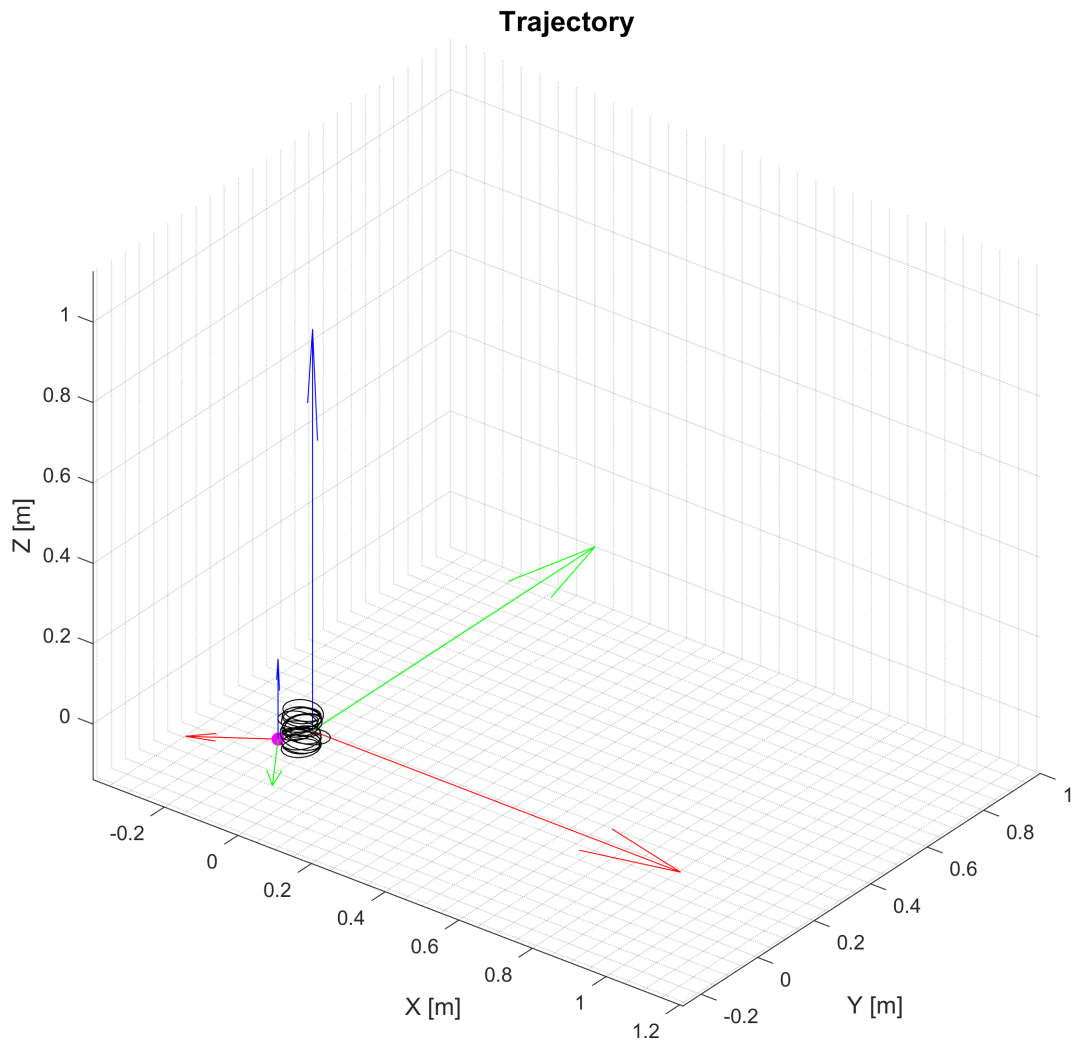


Figure A.33: MMR IMU simulation, rotation about z-axis, with added Gaussian noise, traveled trajectory. Without filtering, motion that was not present is generated due to double integration of errors.

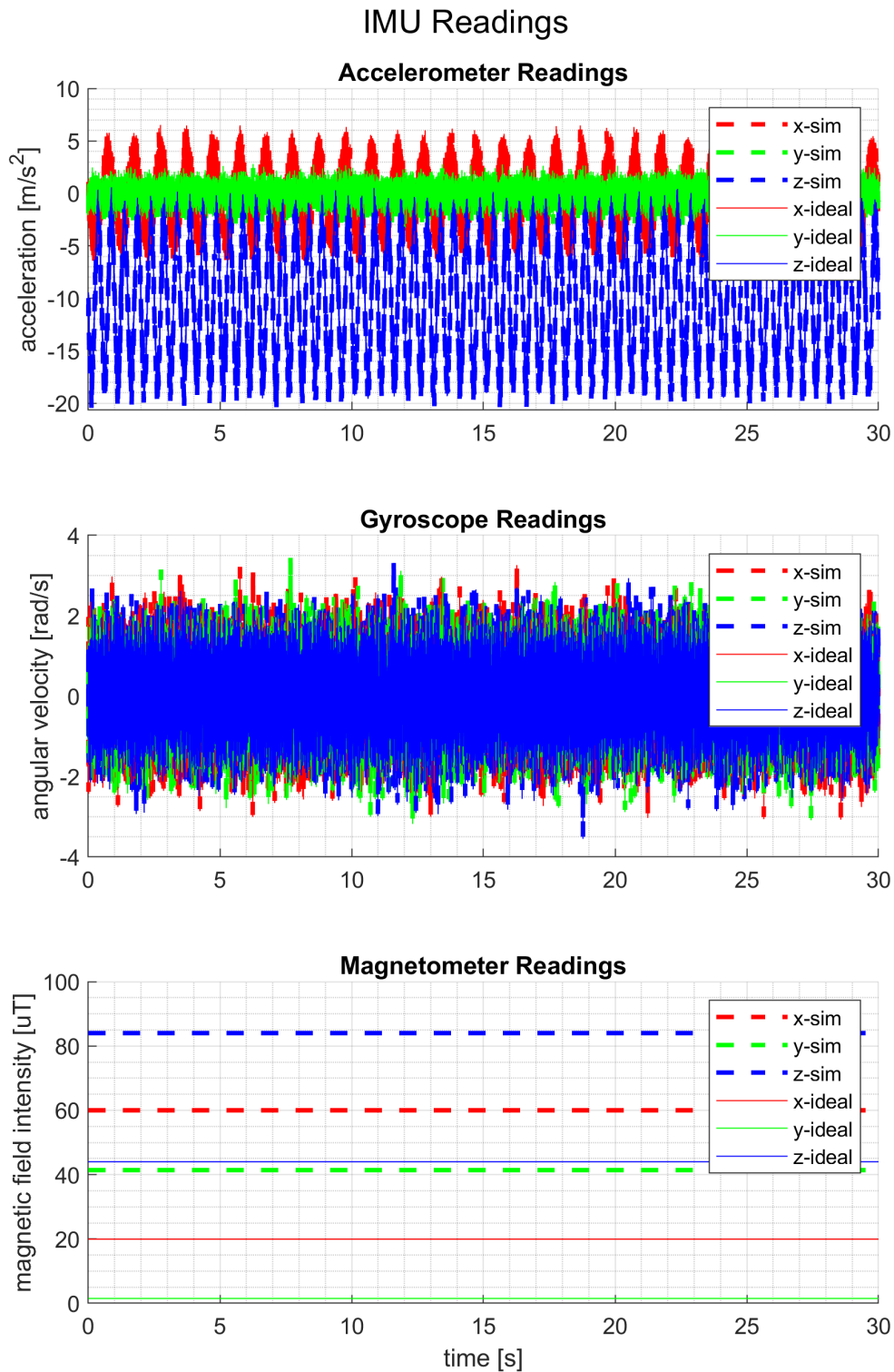


Figure A.34: MMR and ideal IMU simulations, harmonic motion in x- and z-axis, with added Gaussian noise, sensor readings. Deviation of MMR data from an ideal IMU is visible.

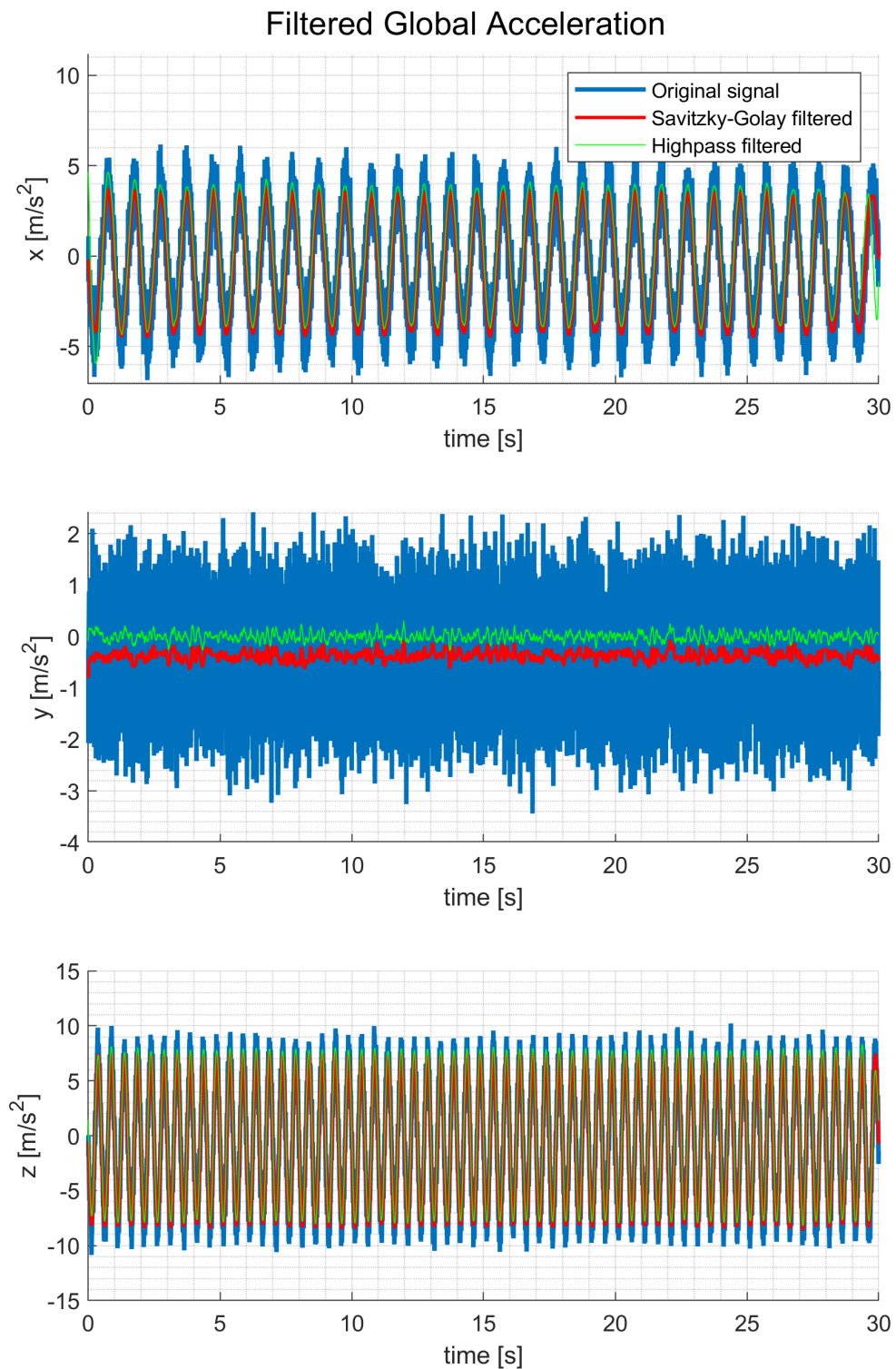


Figure A.35: MMR IMU simulation, harmonic motion in x- and z-axis, with added Gaussian noise, global acceleration. Large oscillation in y-axis is visible, possibly filtered out.

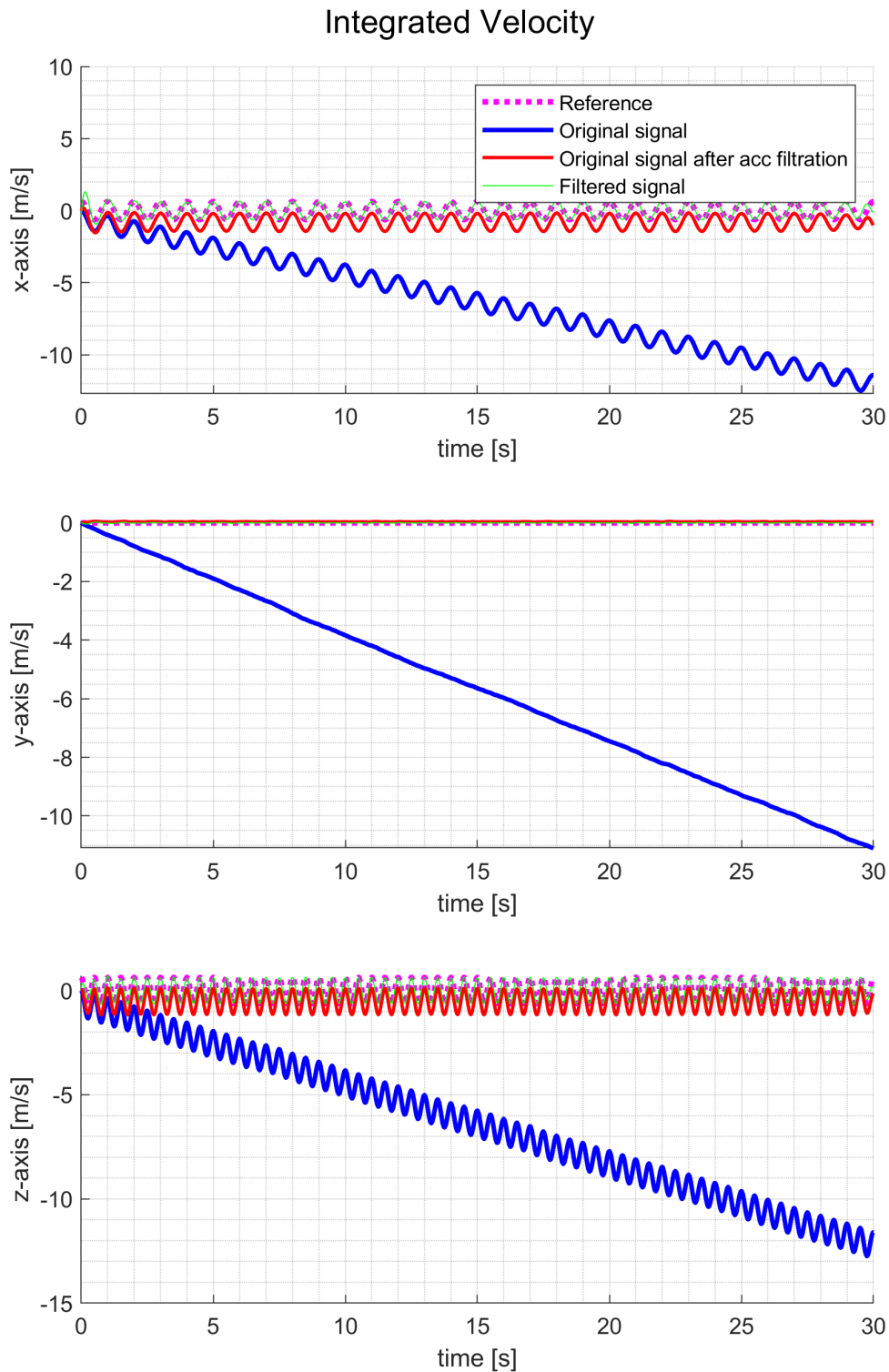


Figure A.36: MMR IMU simulation, harmonic motion in x- and z-axis, with added Gaussian noise, velocity. Original signal diverges, but coincides with reference when fully filtered.

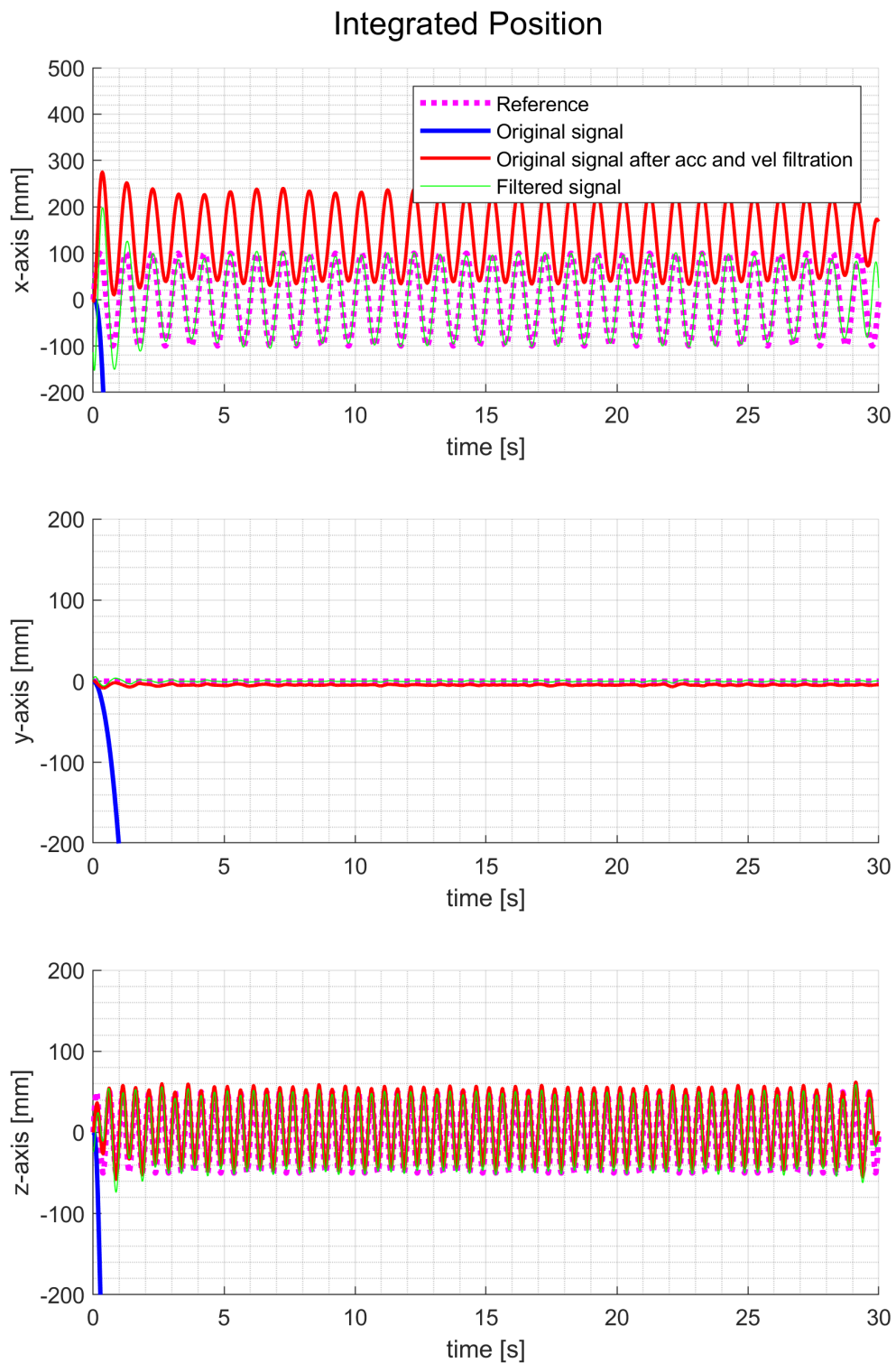


Figure A.37: MMR IMU simulation, harmonic motion in x- and z-axis, with added Gaussian noise, position. Original signal diverges, but coincides with reference when fully filtered.

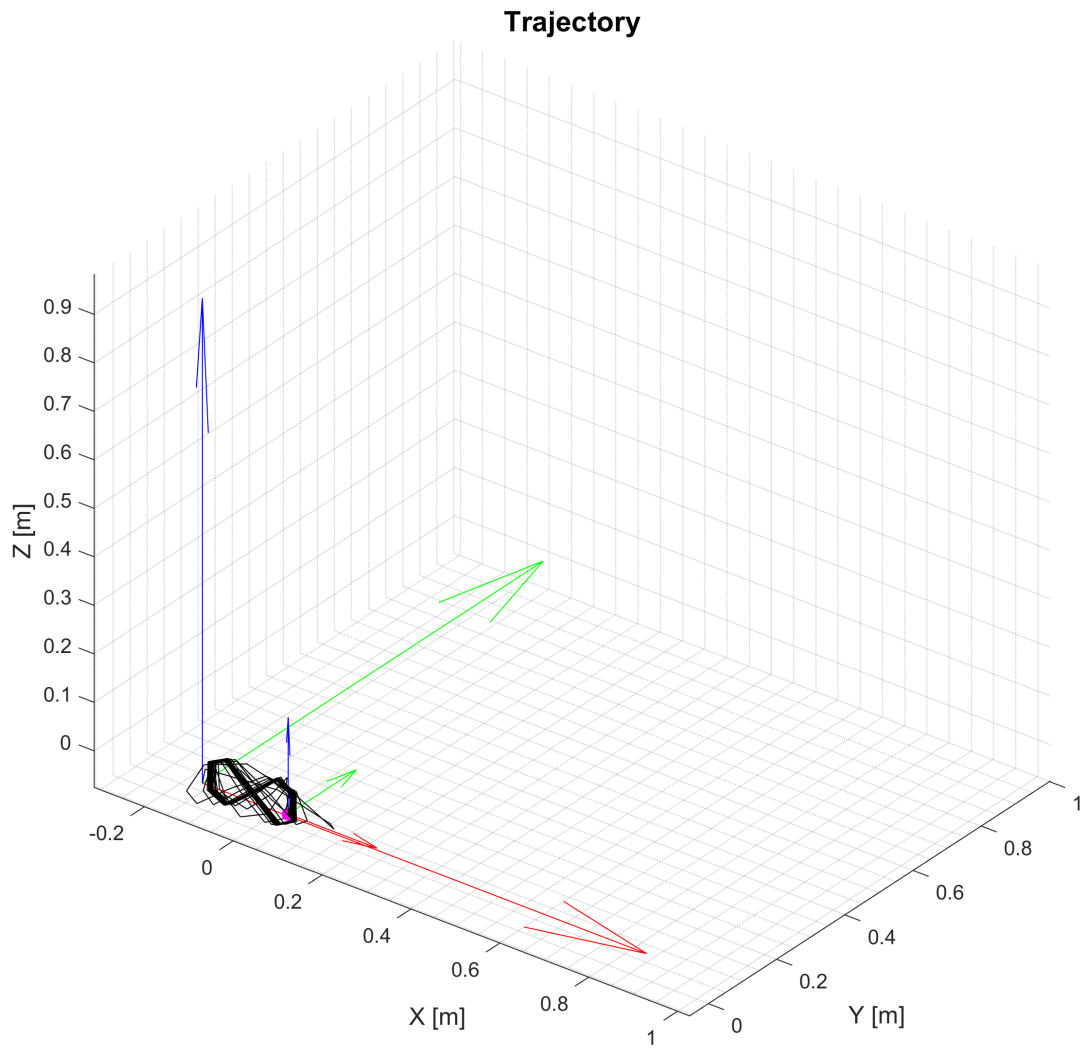


Figure A.38: MMR IMU simulation, harmonic motion in x- and z-axis, with added Gaussian noise, traveled trajectory. With filtering, the trajectory reflects real motion.

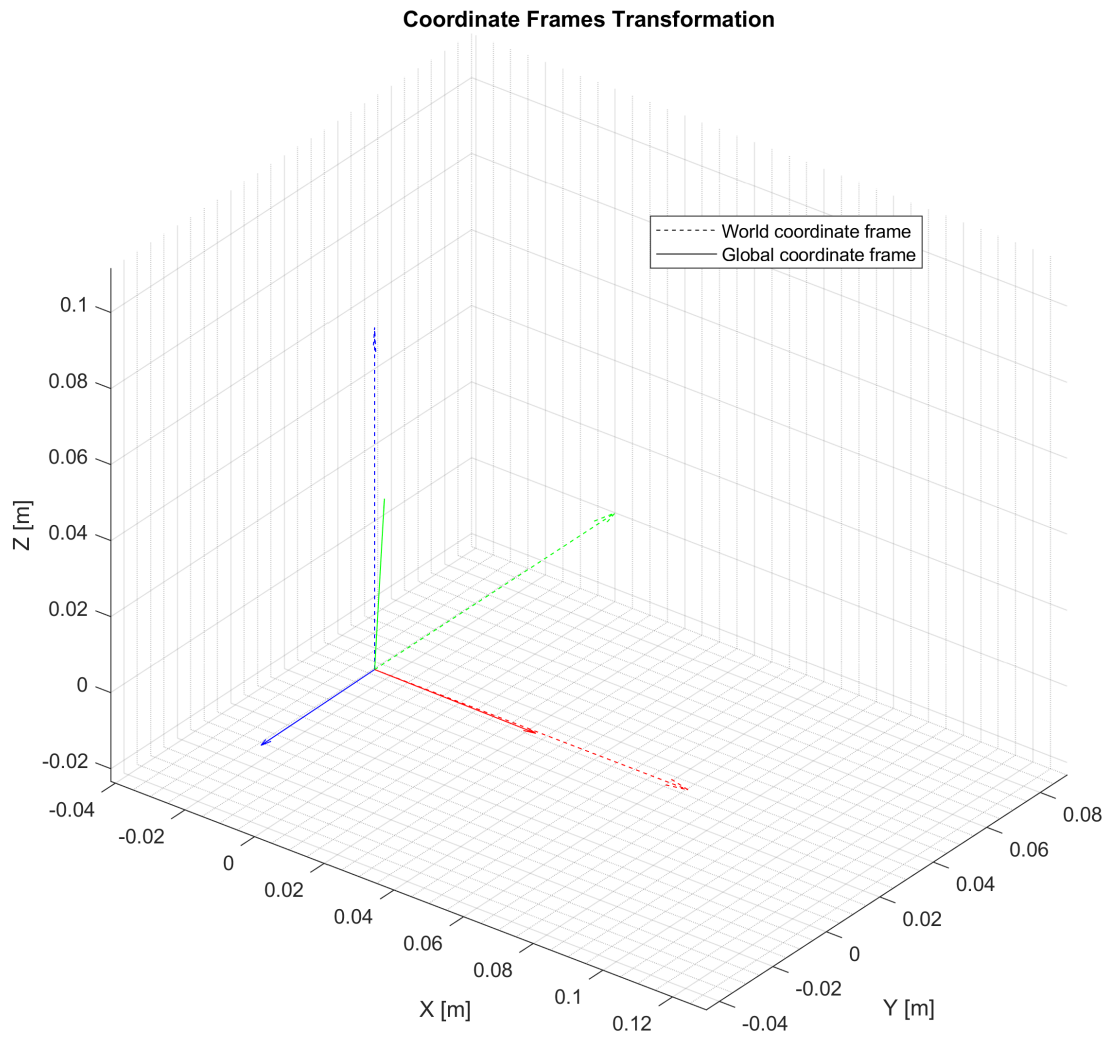


Figure A.39: MMR IMU experiment, transformation of coordinate frames.

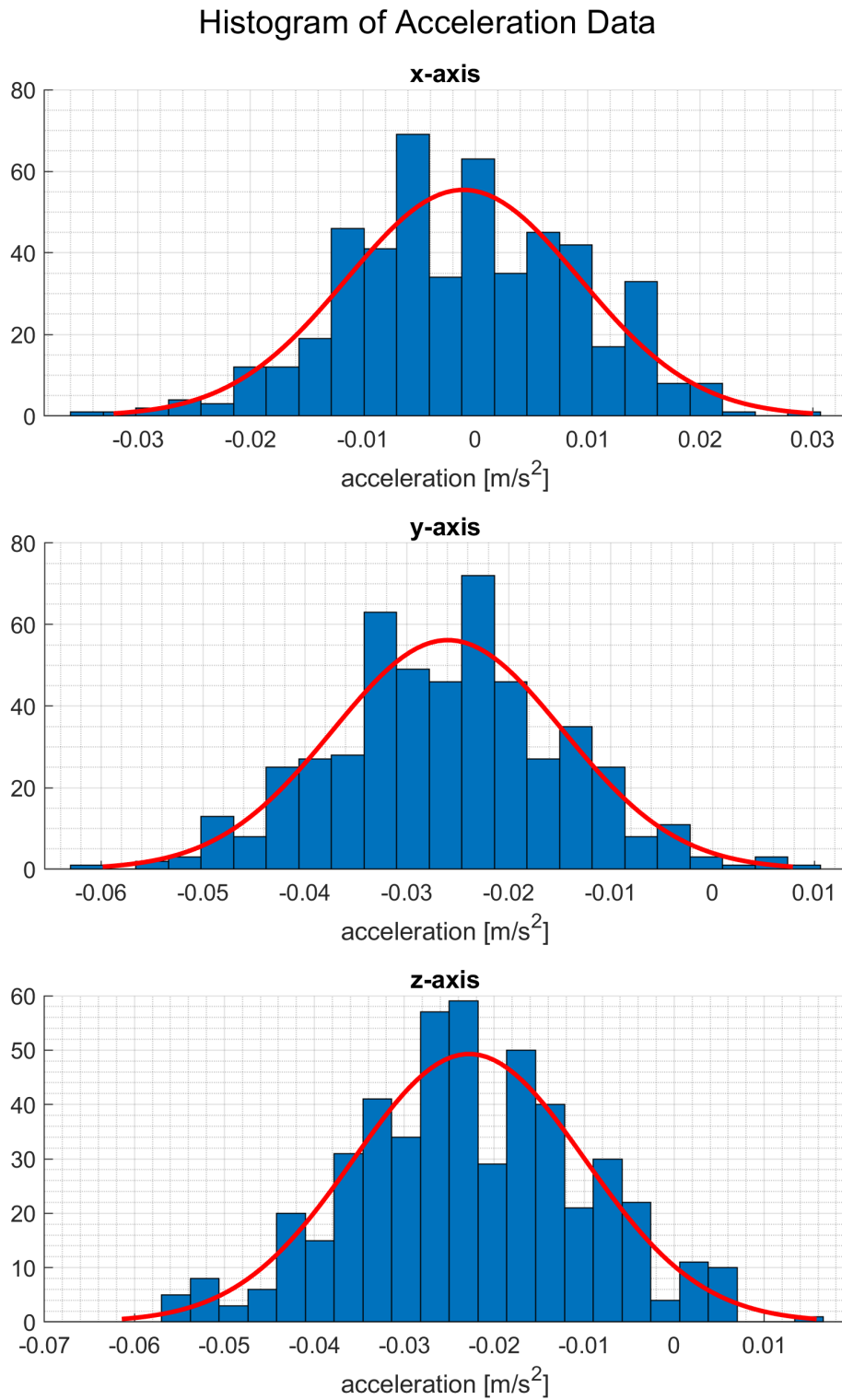


Figure A.40: MMR IMU experiment, steady state, calibrated linear acceleration, histogram with Gaussian fit. Offset from ideal zero measurements is shown, noise appears to be Gaussian.

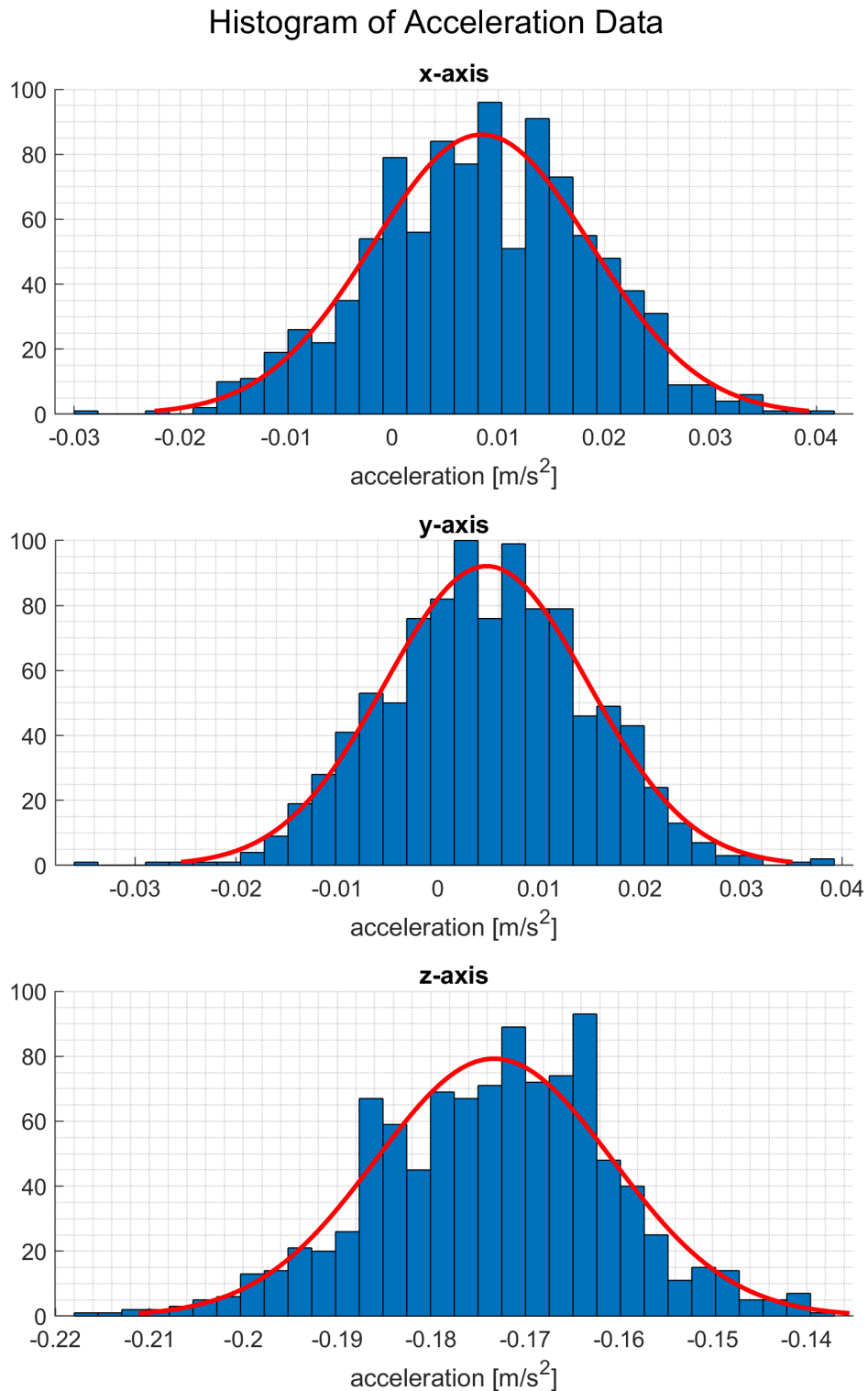


Figure A.41: MMR IMU experiment, steady state, not calibrated linear acceleration, histogram with Gaussian fit. Offset from ideal zero measurements is shown, noise appears to be Gaussian.

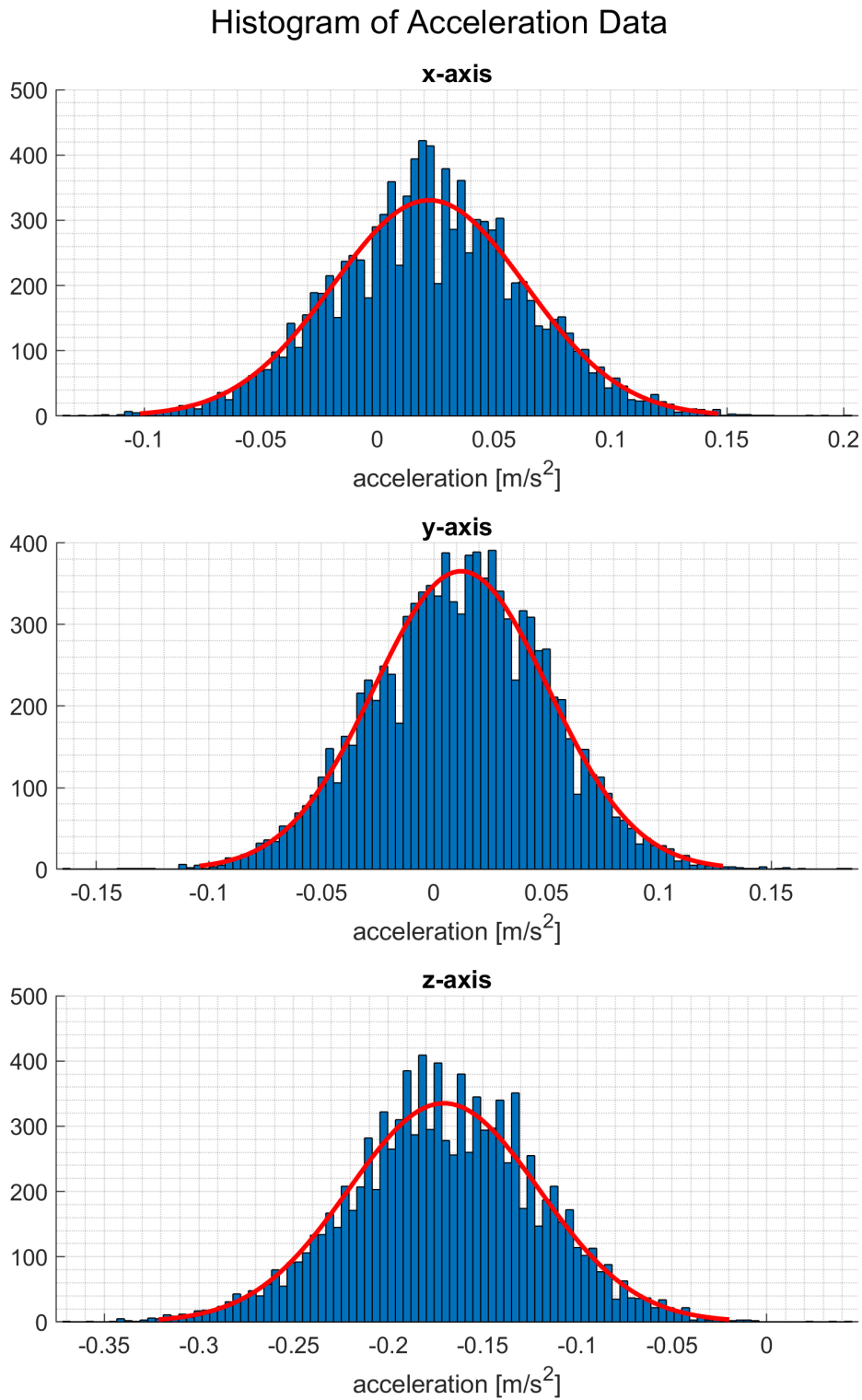


Figure A.42: MMR IMU experiment, steady state, raw acceleration, histogram with Gaussian fit. Offset from ideal zero measurements is shown, noise appears not Gaussian.

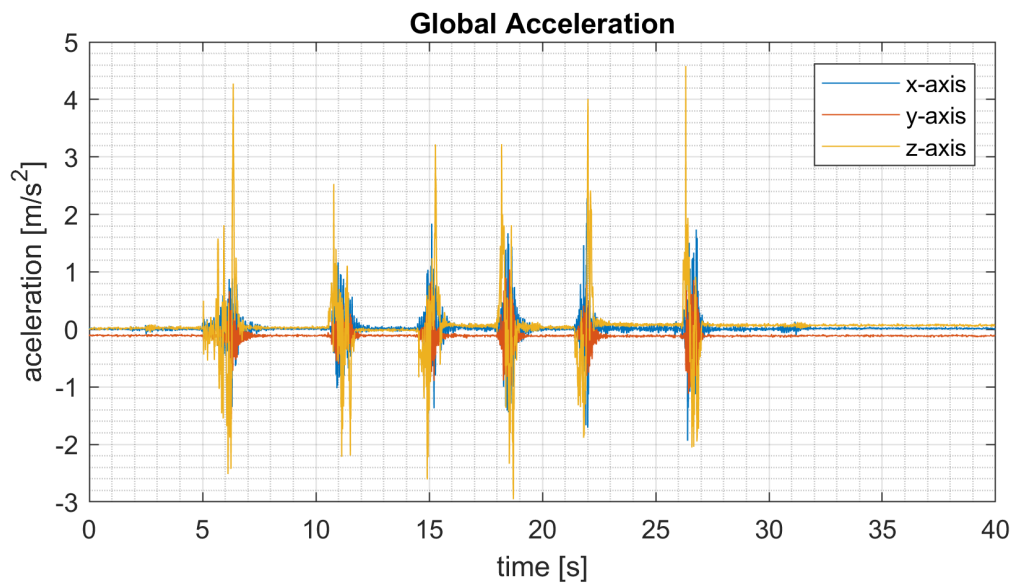


Figure A.43: MMR IMU experiment, repeated translation in z-axis, linear acceleration, global acceleration. The undesirable bias increases with time.

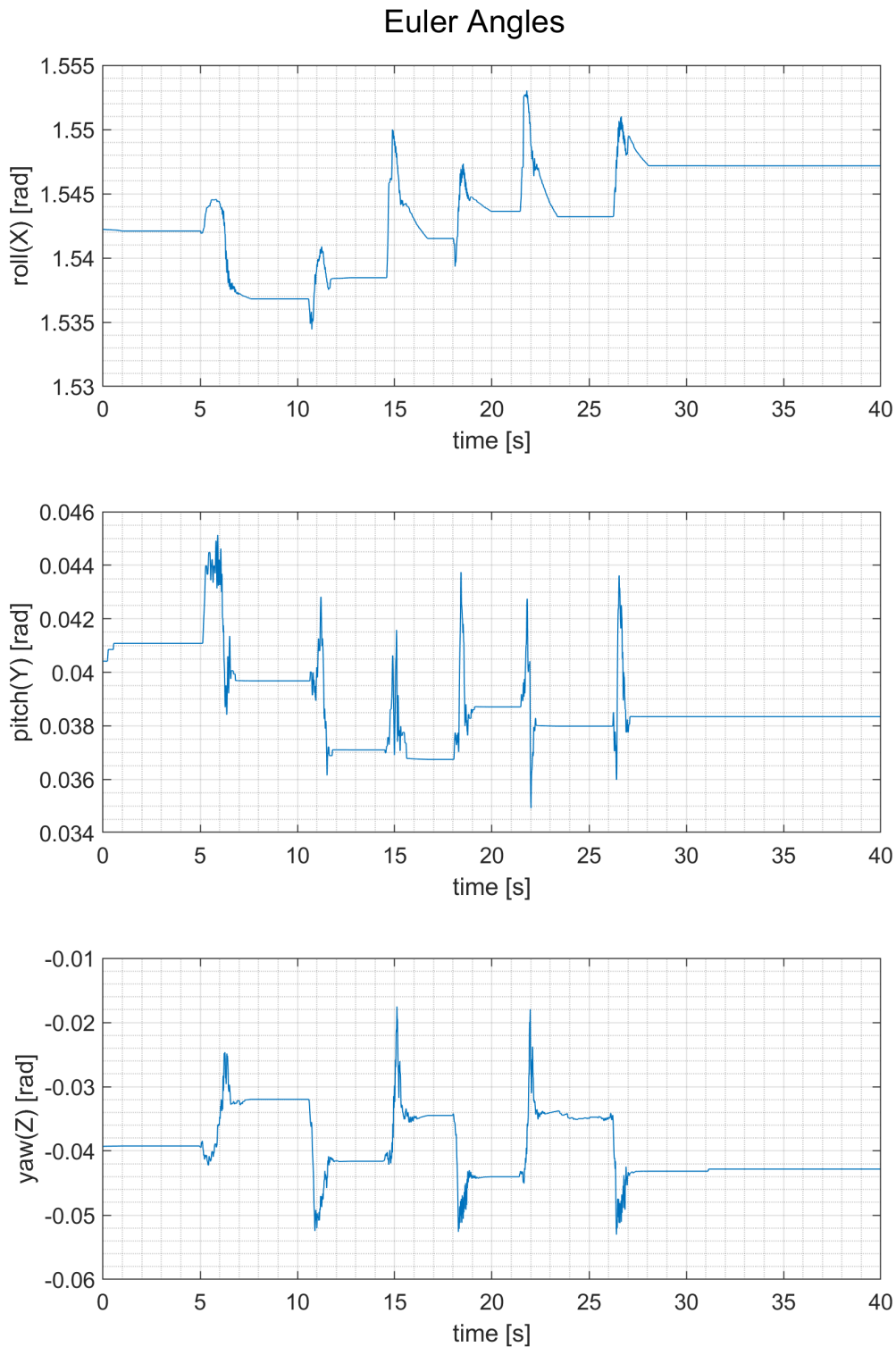


Figure A.44: MMR IMU experiment, repeated translation in z-axis, linear acceleration, Euler angles. Visible drift and deviations are sufficiently small.

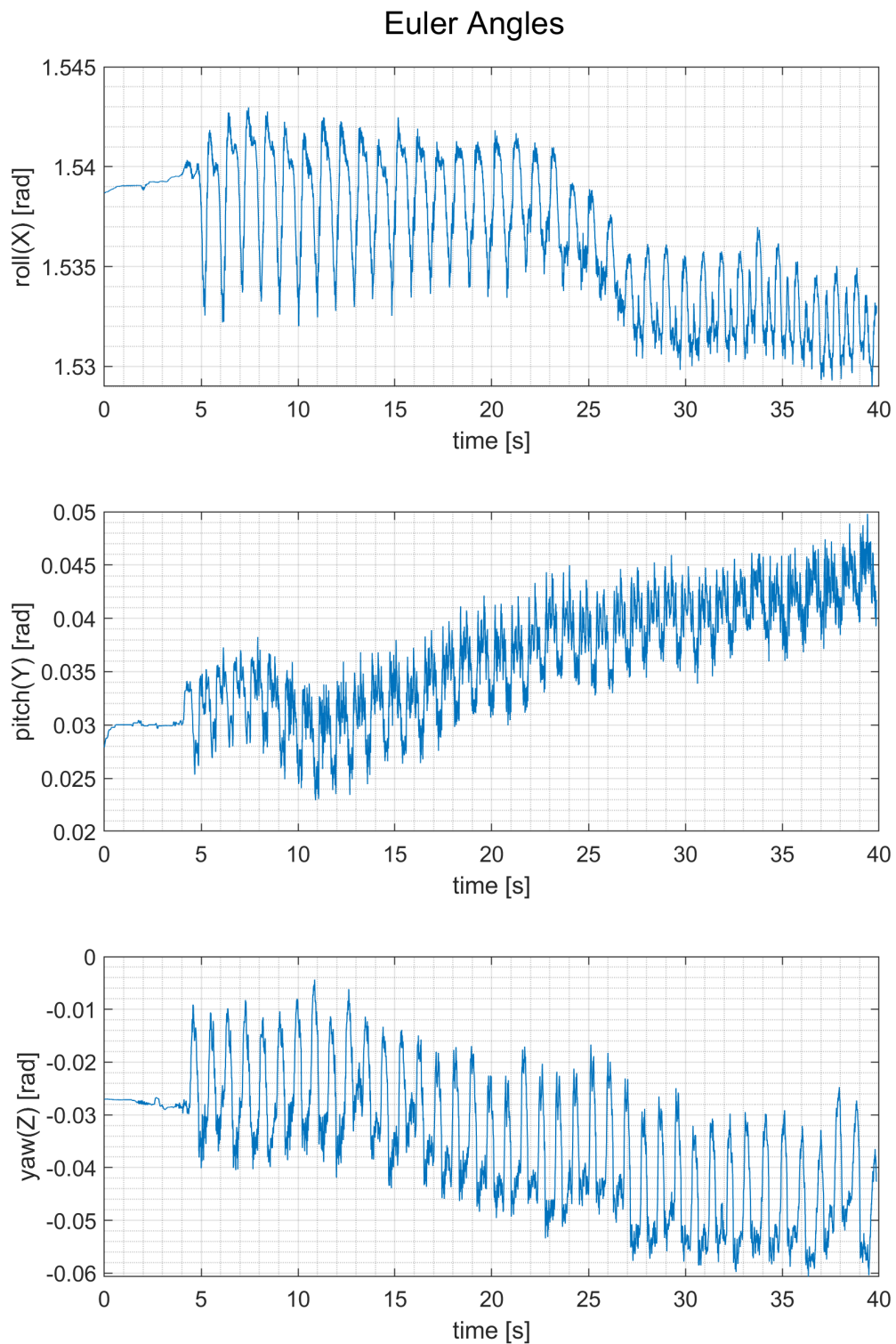


Figure A.45: MMR IMU experiment, harmonic motion in z-axis (fast), linear acceleration, Euler angles. Visible drift and deviations are sufficiently small.

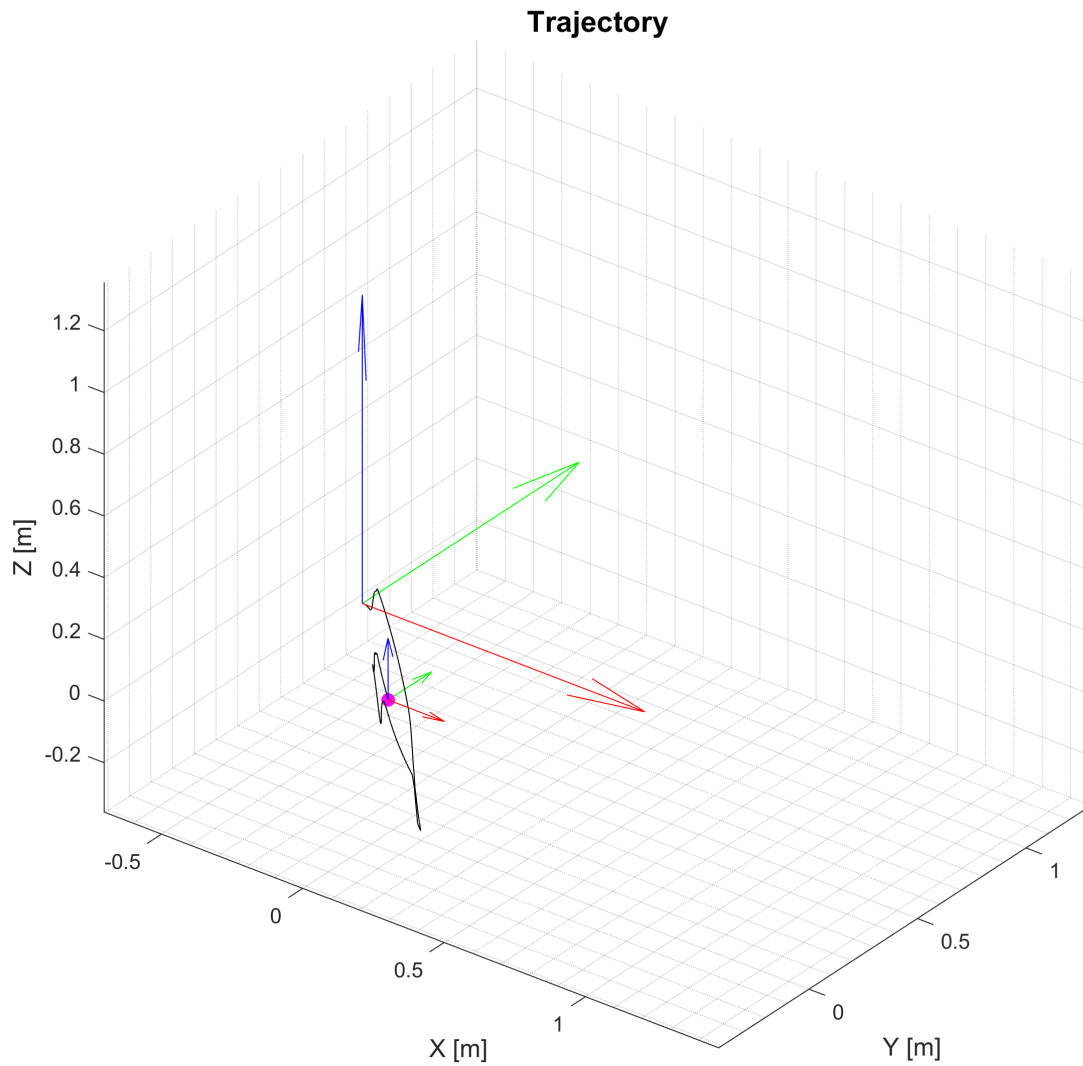


Figure A.46: MMR IMU experiment, repeated translation in z-axis, calibrated linear acceleration, traveled trajectory. Even when filtered, the trajectory does not reflect real motion.

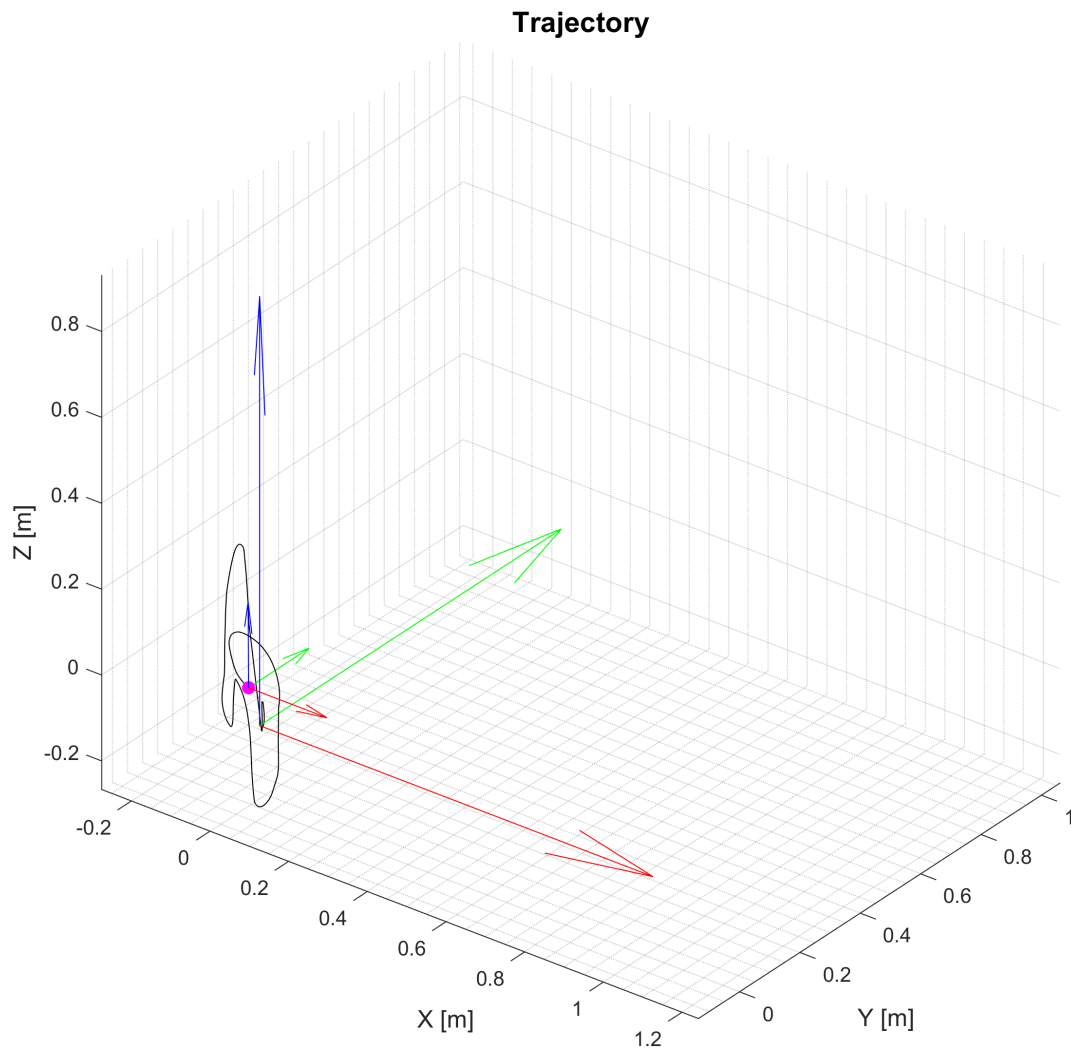


Figure A.47: MMR IMU experiment, repeated translation in z-axis, not calibrated linear acceleration, traveled trajectory. Even when filtered, the trajectory does not reflect real motion.

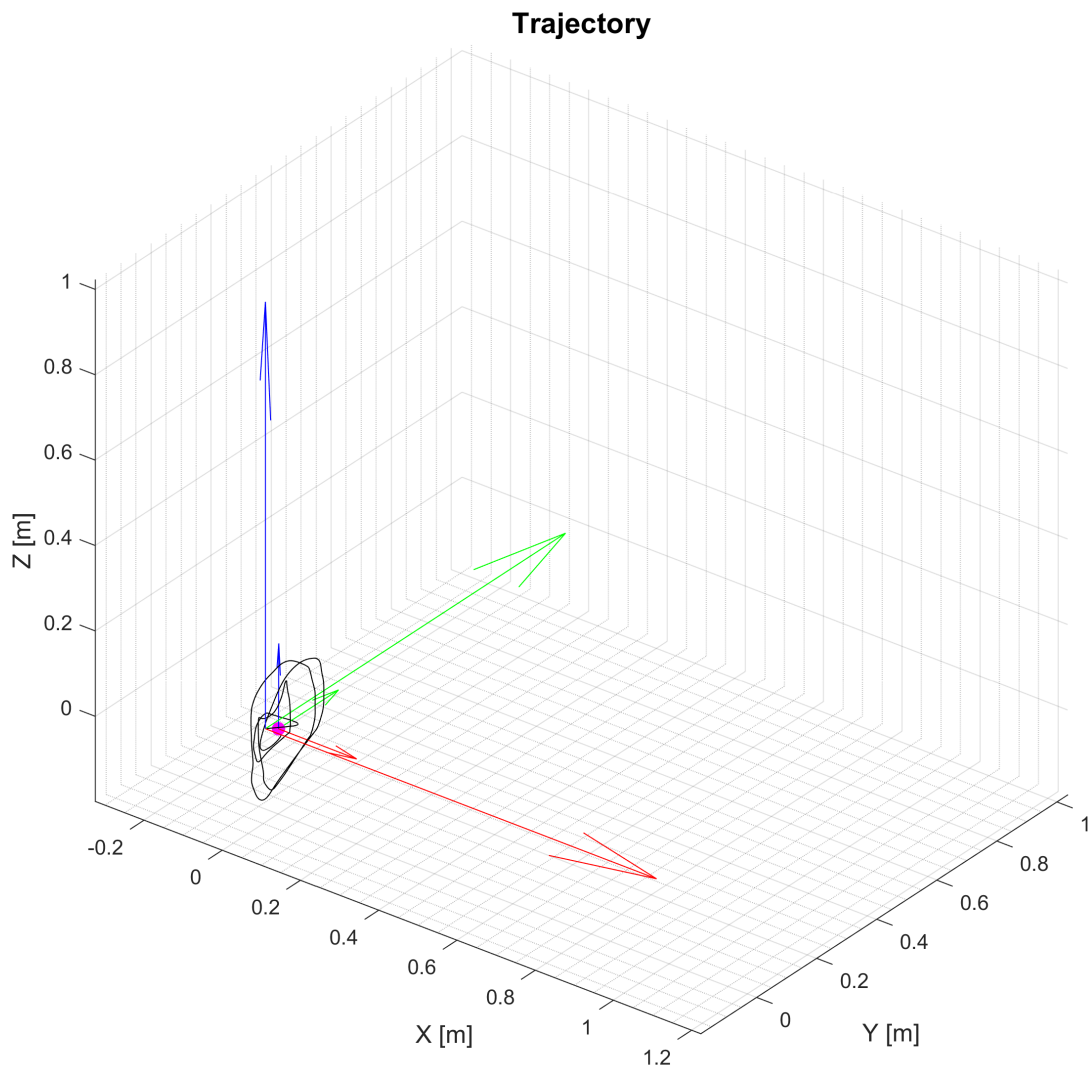


Figure A.48: MMR IMU experiment, repeated translation in z-axis, raw acceleration, traveled trajectory. Even when filtered, the trajectory does not reflect real motion.

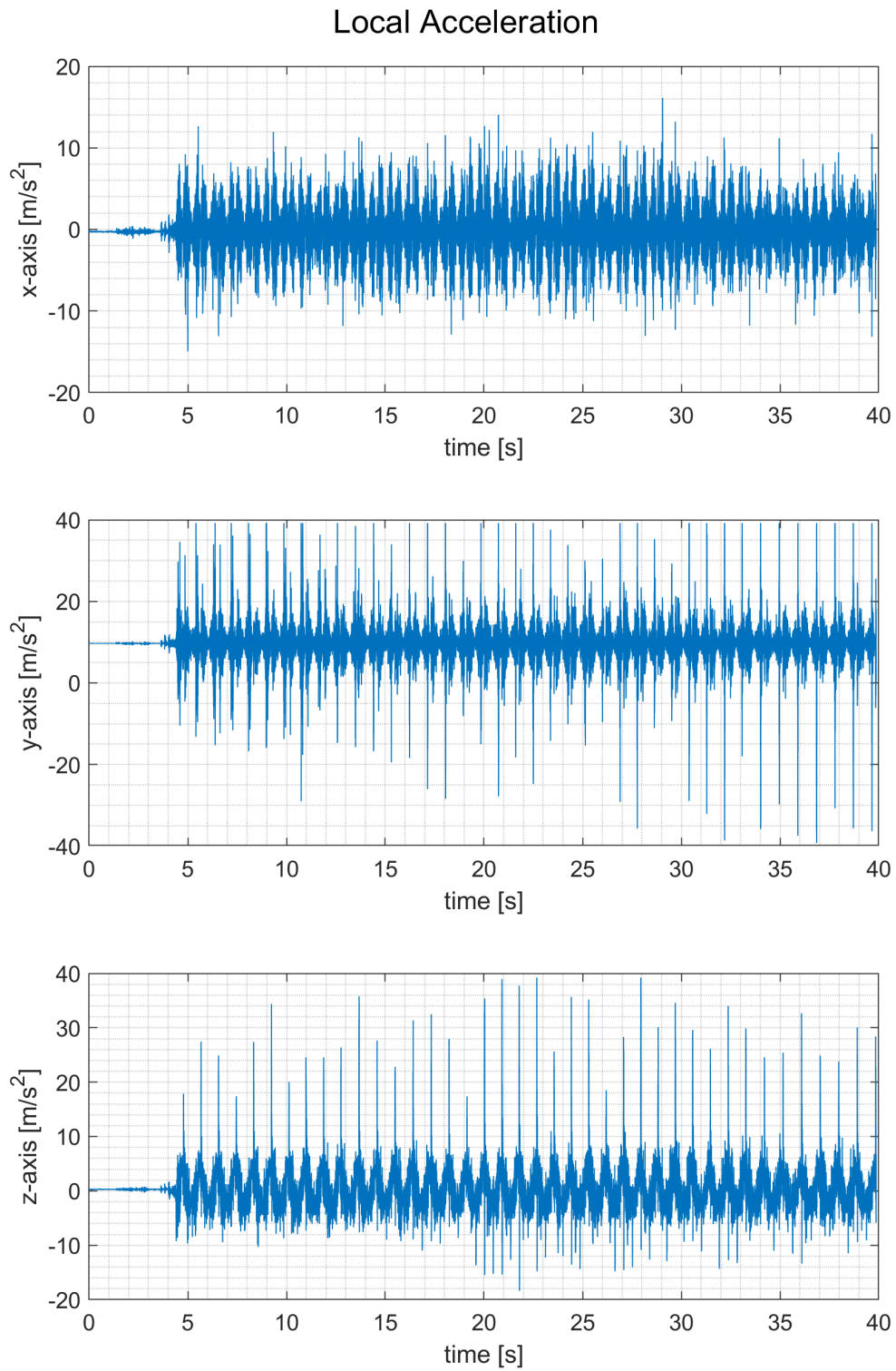


Figure A.49: MMR IMU experiment, harmonic motion in z-axis (slow), raw acceleration, local acceleration.

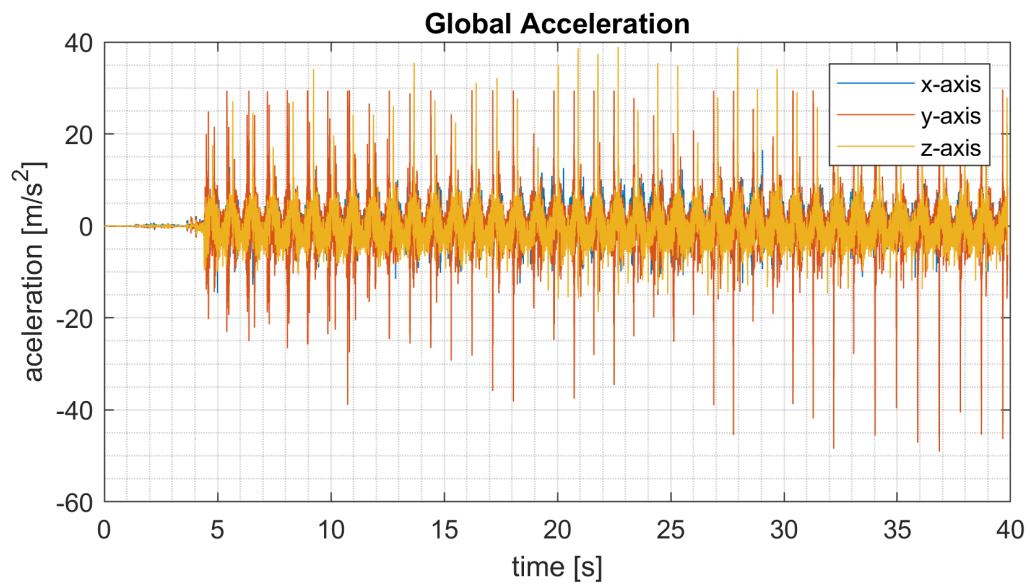


Figure A.50: MMR IMU experiment, harmonic motion in z-axis (slow), raw acceleration, global acceleration.

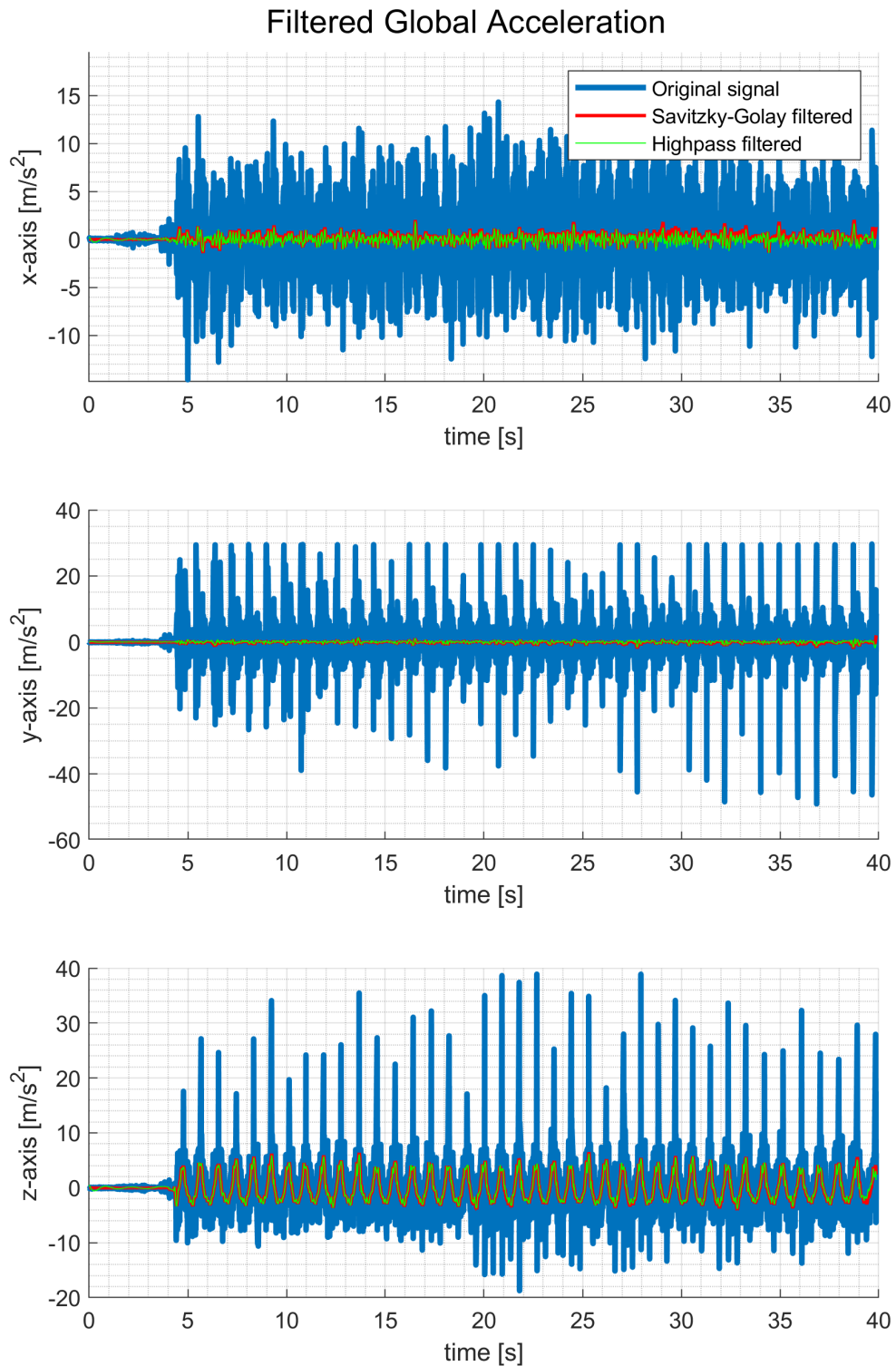


Figure A.51: MMR IMU experiment, harmonic motion in z-axis (slow), raw acceleration, global acceleration. Undesirable oscillation in x- and y-axis is visible, possibly filtered out.

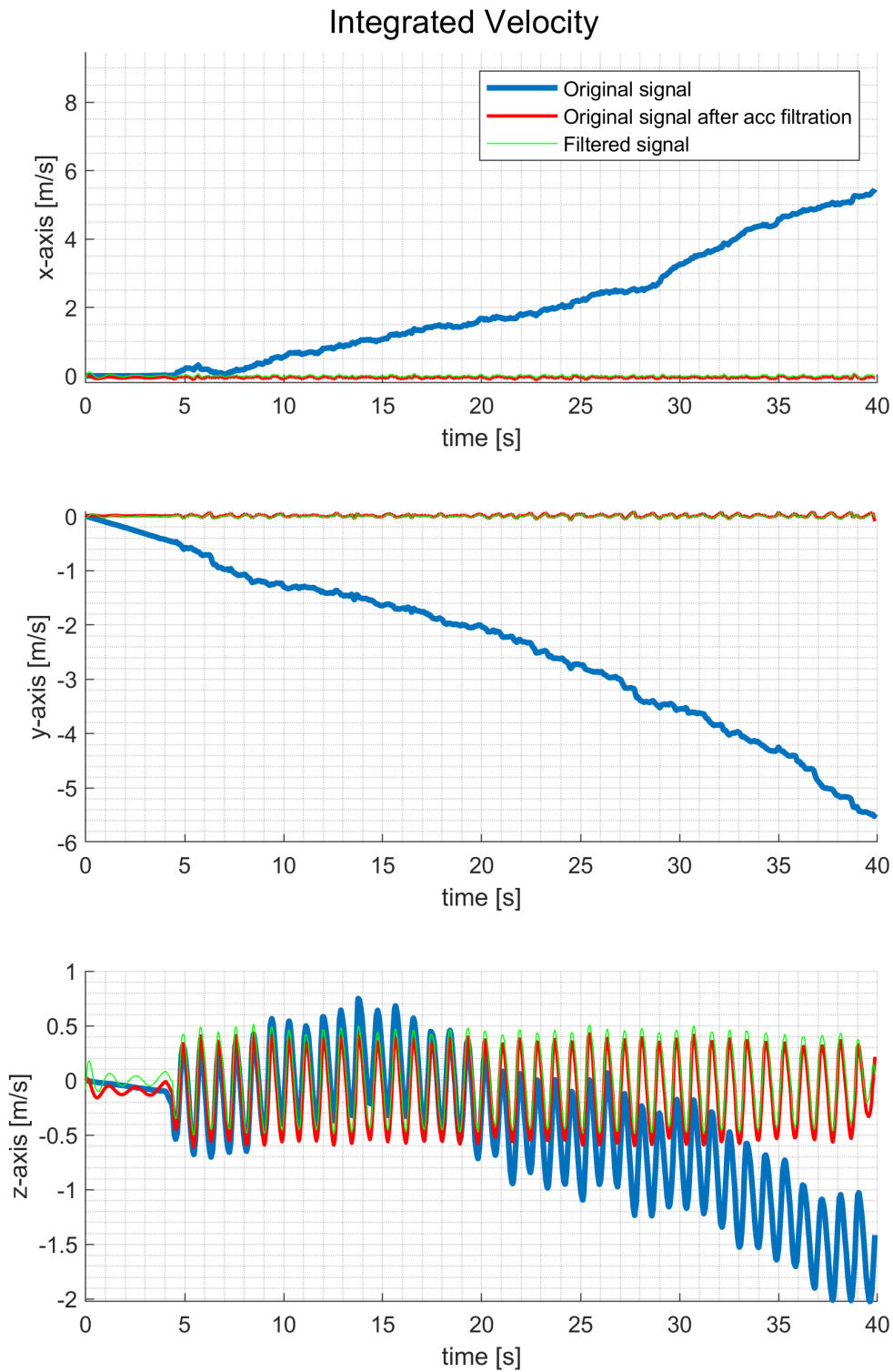


Figure A.52: MMR IMU experiment, harmonic motion in z-axis (slow), raw acceleration, velocity. Original signal diverges, but can be filtered out to show expected values.

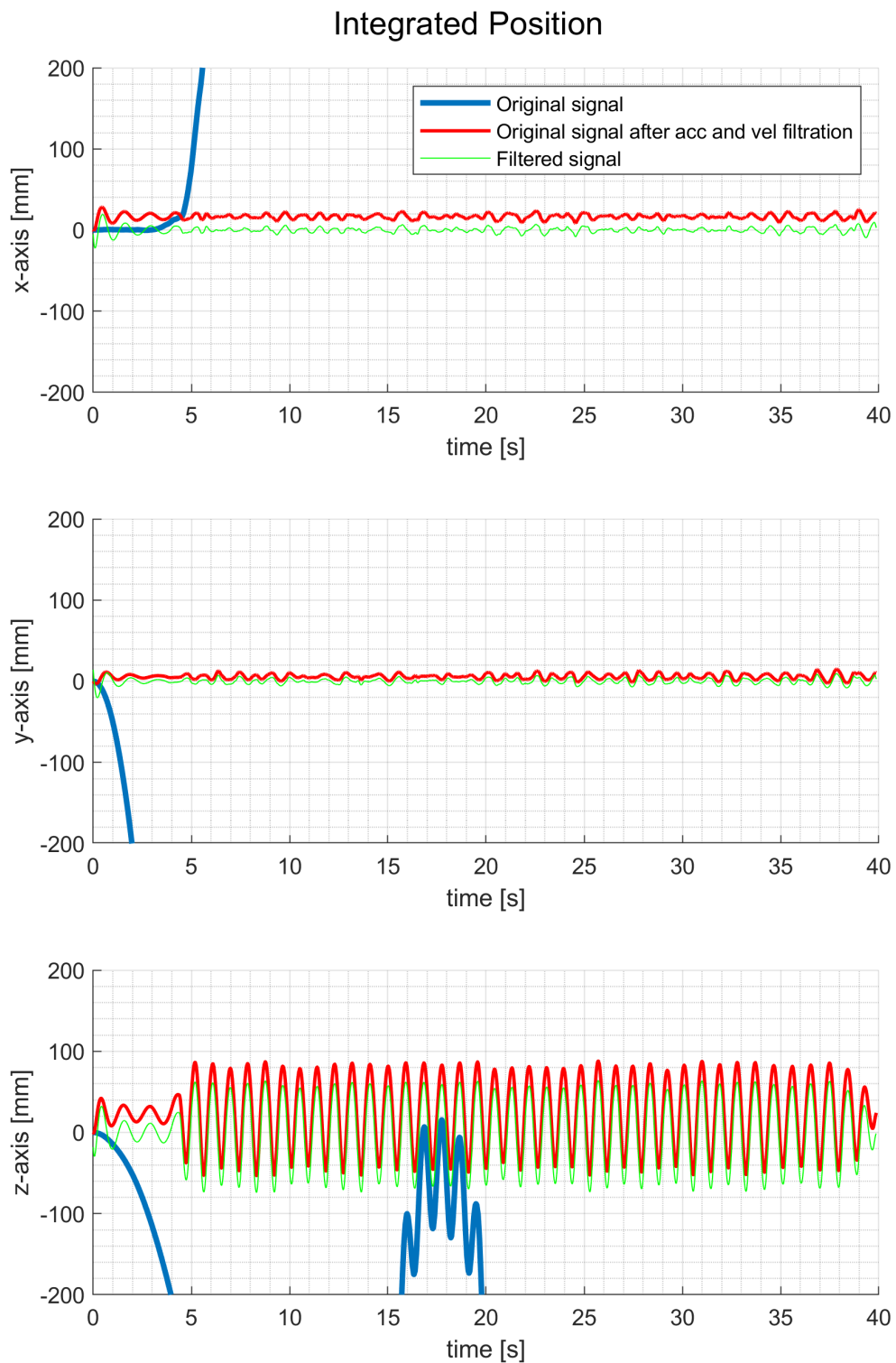


Figure A.53: MMR IMU experiment, harmonic motion in z-axis (slow), raw acceleration, position. Original signal diverges, but can be filtered out to show expected values.

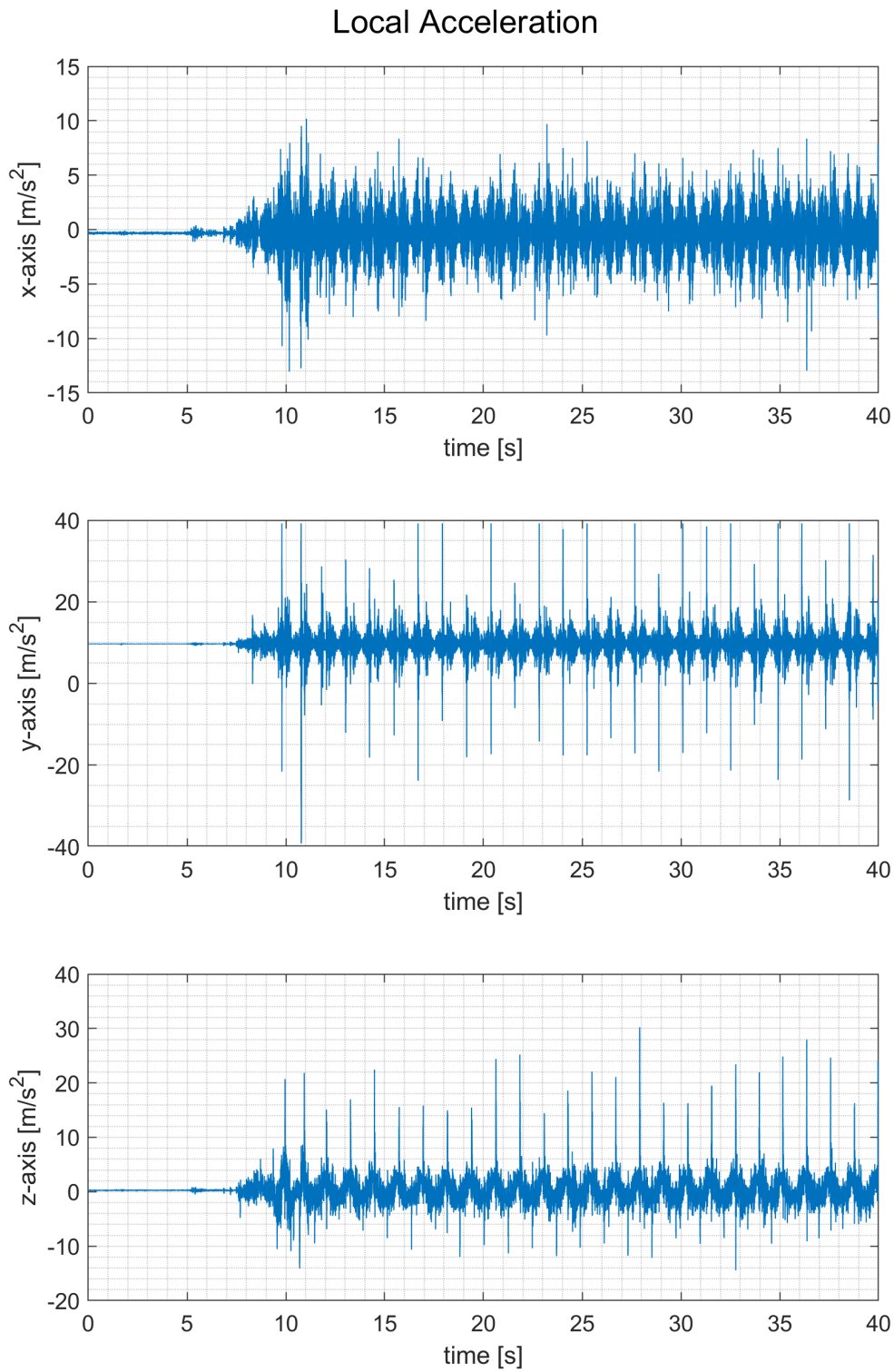


Figure A.54: MMR IMU experiment, harmonic motion in z-axis (fast), raw acceleration, local acceleration.

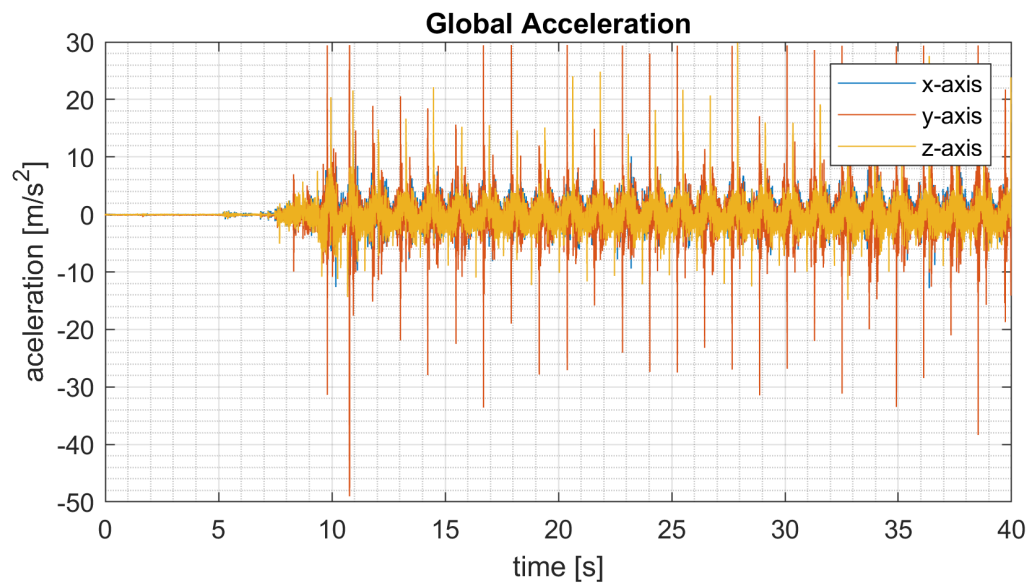


Figure A.55: MMR IMU experiment, harmonic motion in z-axis (fast), raw acceleration, global acceleration.

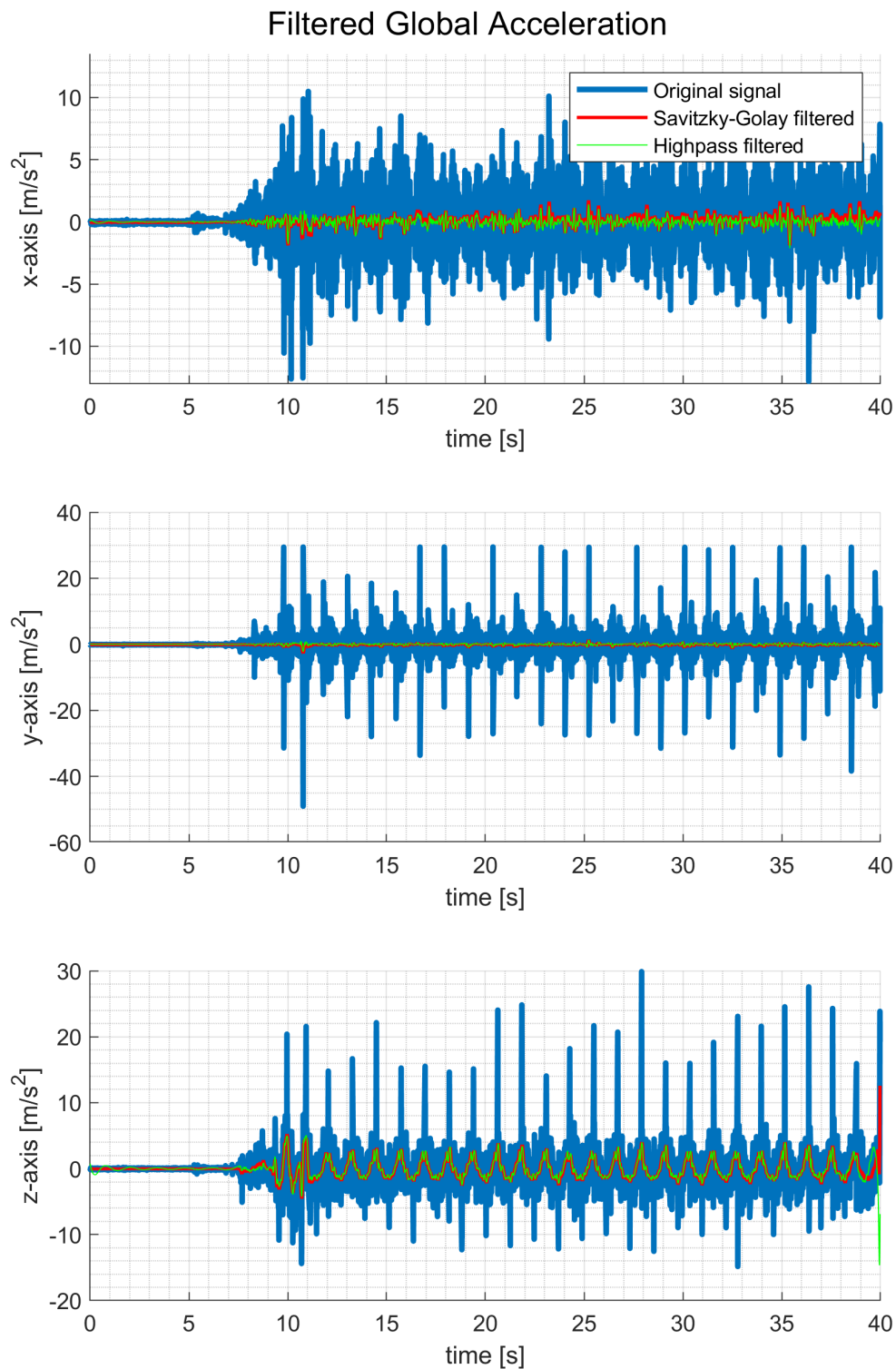


Figure A.56: MMR IMU experiment, harmonic motion in z-axis (fast), raw acceleration, global acceleration. Undesirable oscillation in x- and y-axis is visible, possibly filtered out.

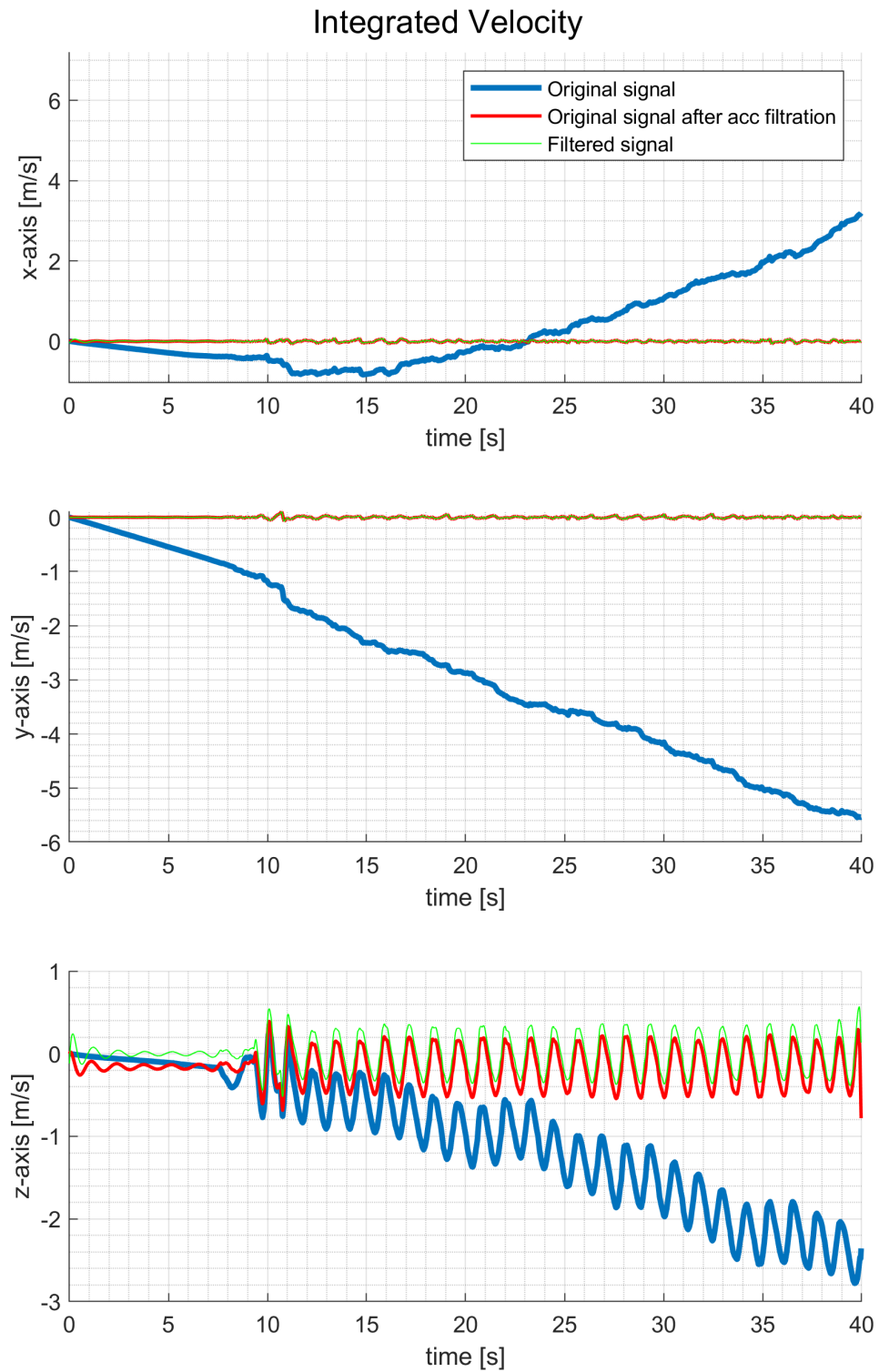


Figure A.57: MMR IMU experiment, harmonic motion in z-axis (fast), raw acceleration, velocity. Original signal diverges, but can be filtered out to show expected values.

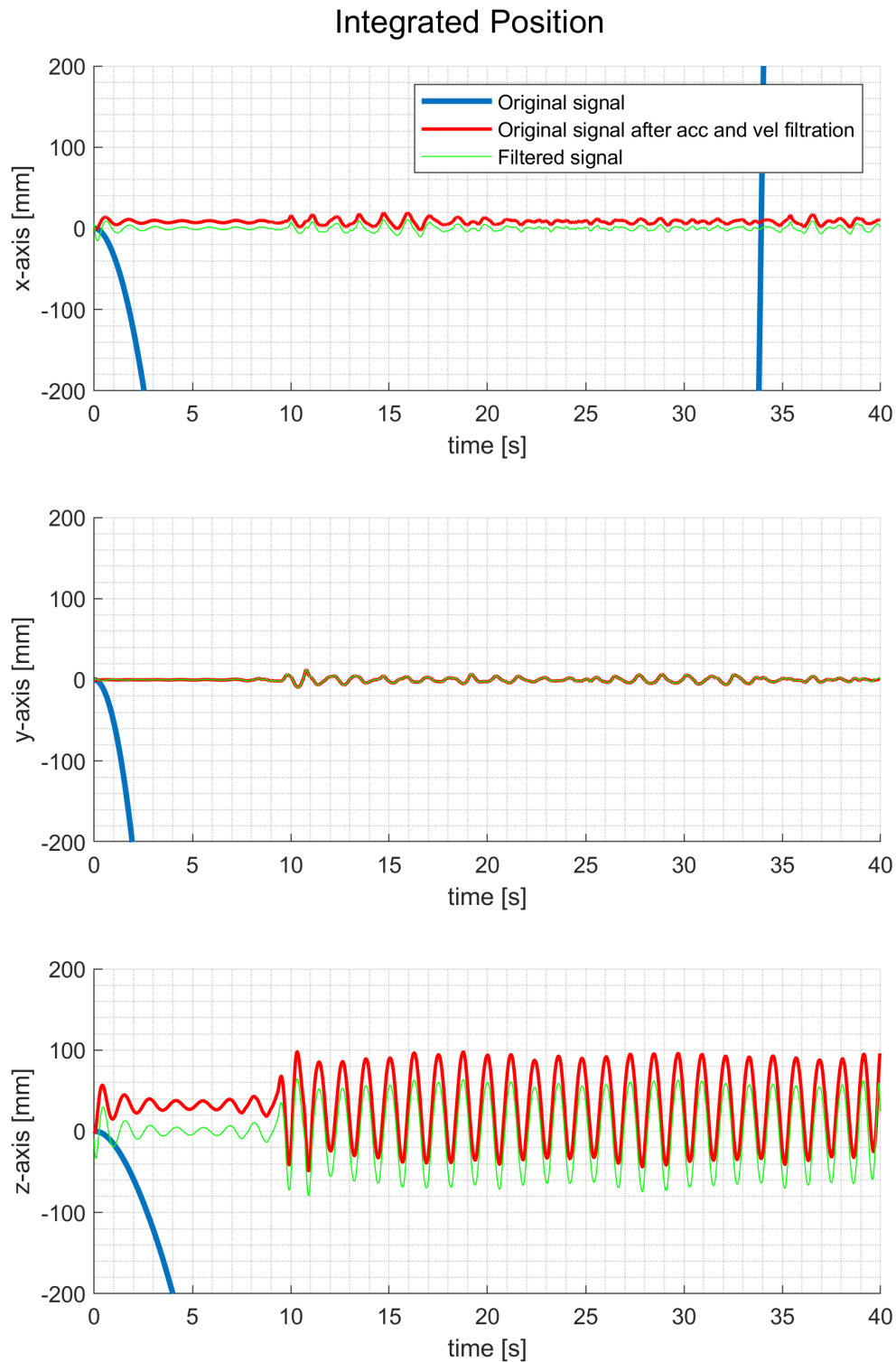


Figure A.58: MMR IMU experiment, harmonic motion in z-axis (fast), raw acceleration, position. Original signal diverges, but can be filtered out to show expected values.

■ A.3 T 265 tracking camera

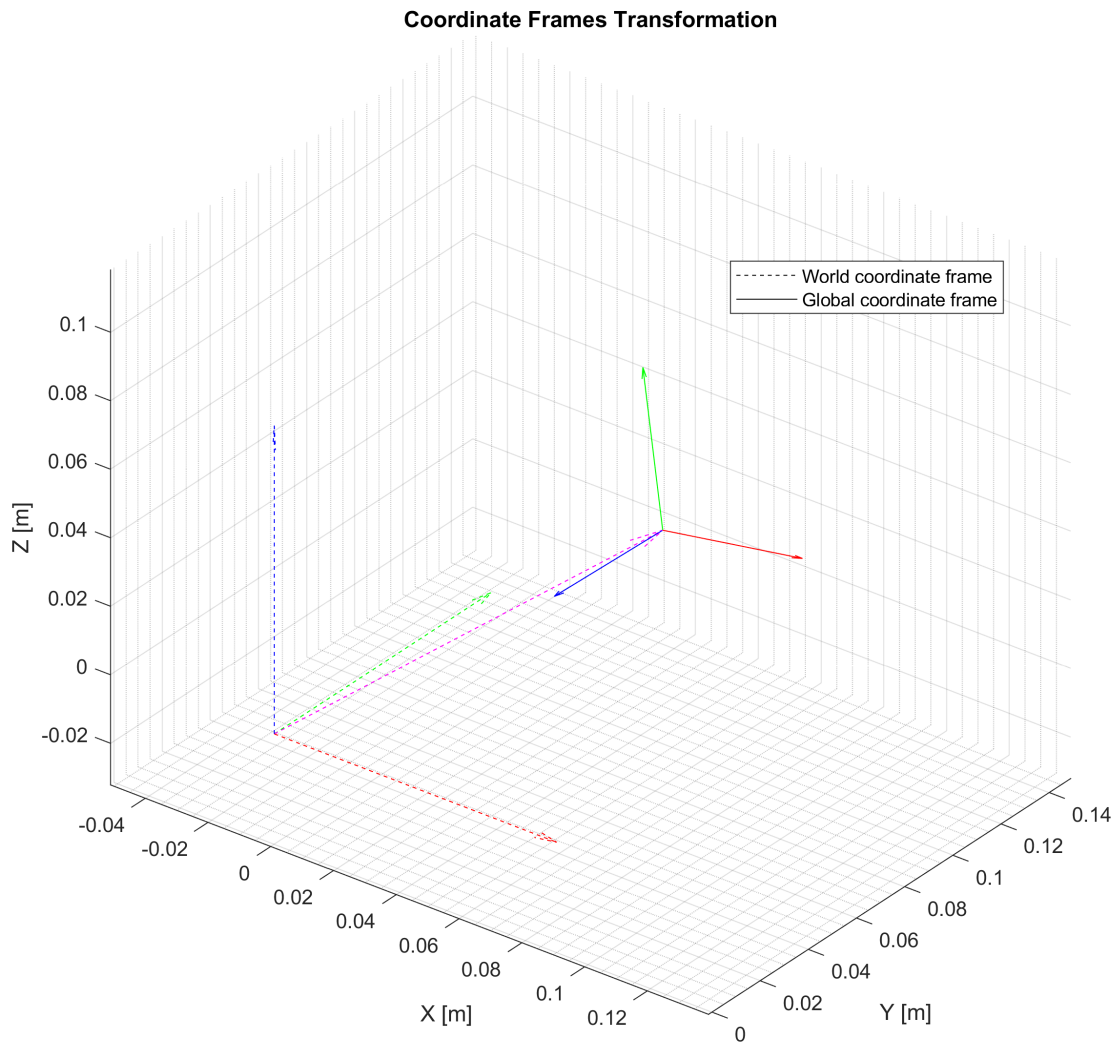


Figure A.59: RealSense T265 experiment, transformation of coordinate frames.

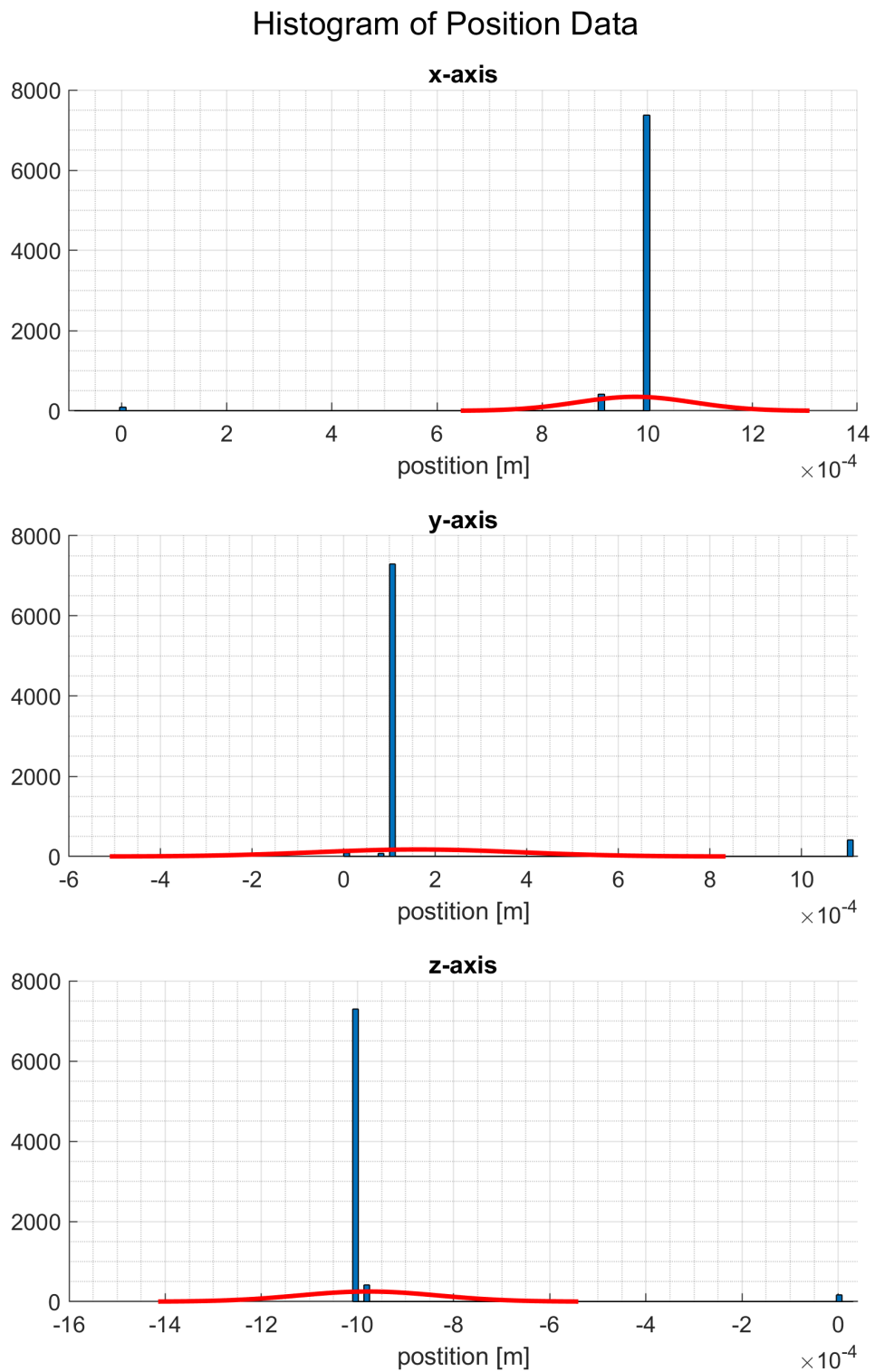


Figure A.60: RealSense T265 experiment, steady state, histogram with Gaussian fit. Offset is visible but sufficiently small, no noise distribution is shown.

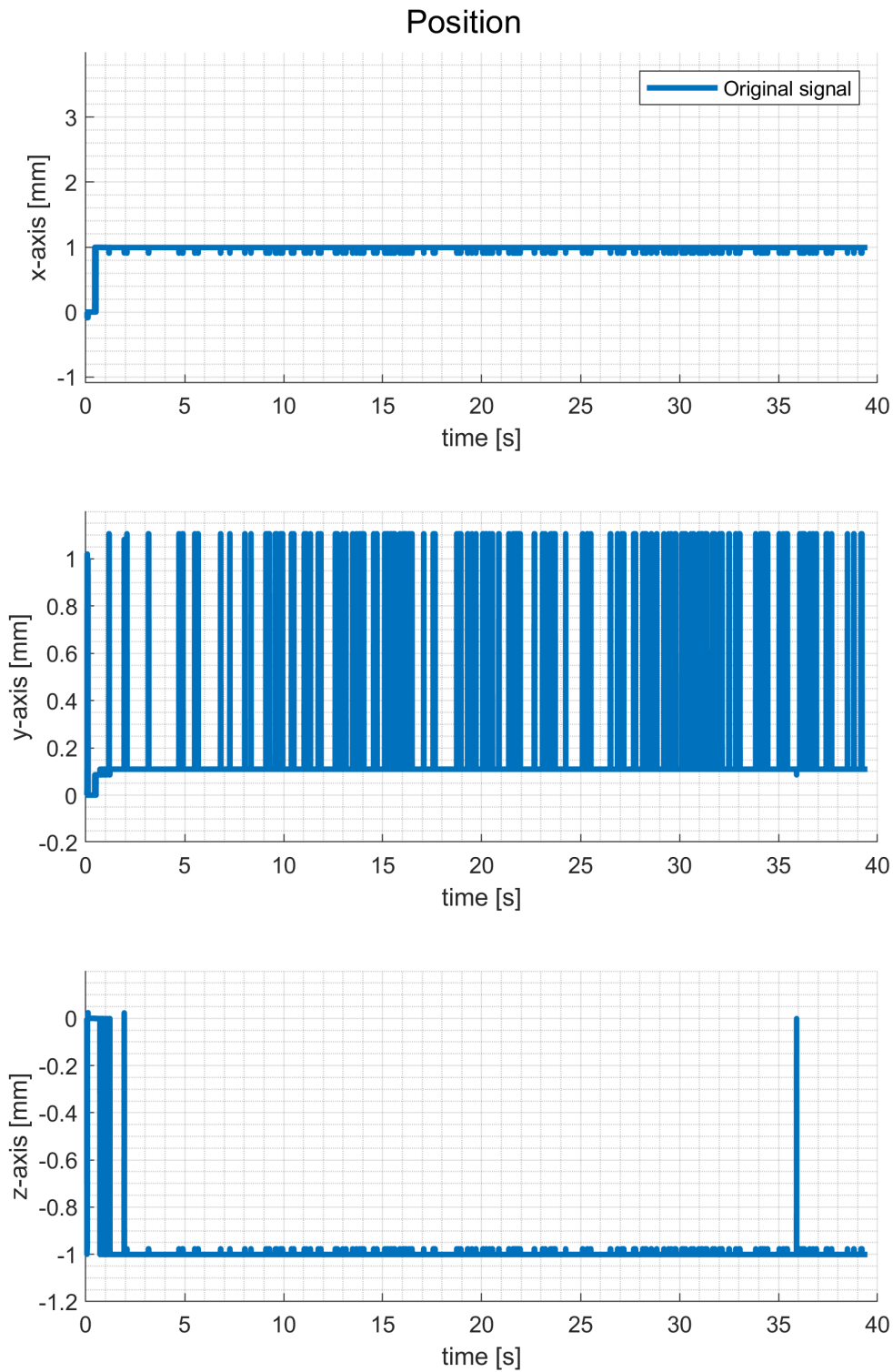


Figure A.61: RealSense T265 experiment, steady state, position. The measured data are stable in sufficient range.

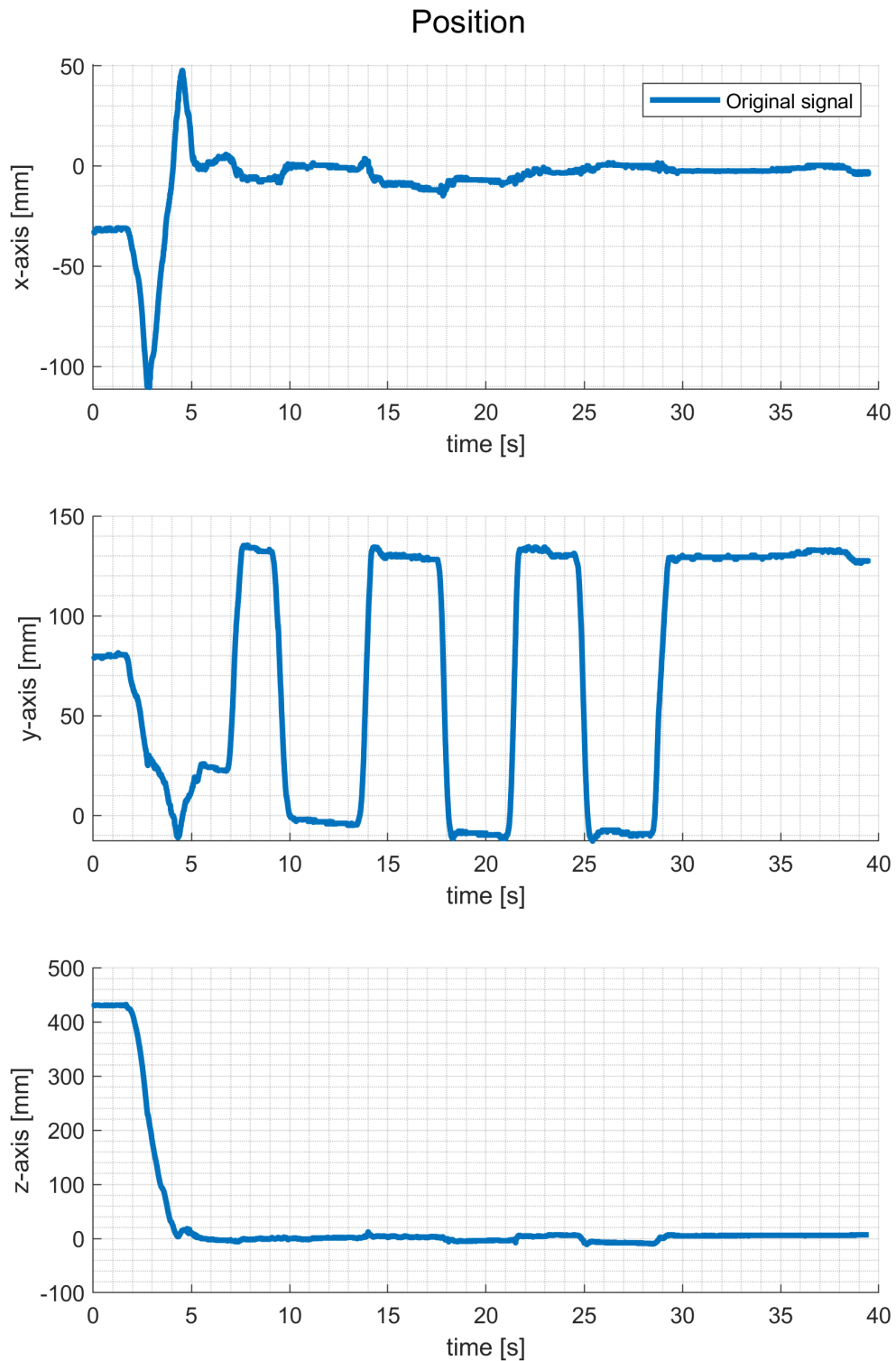


Figure A.62: RealSense T265 experiment, repeated translation in y-axis, position. With coordinate frames initialization part.

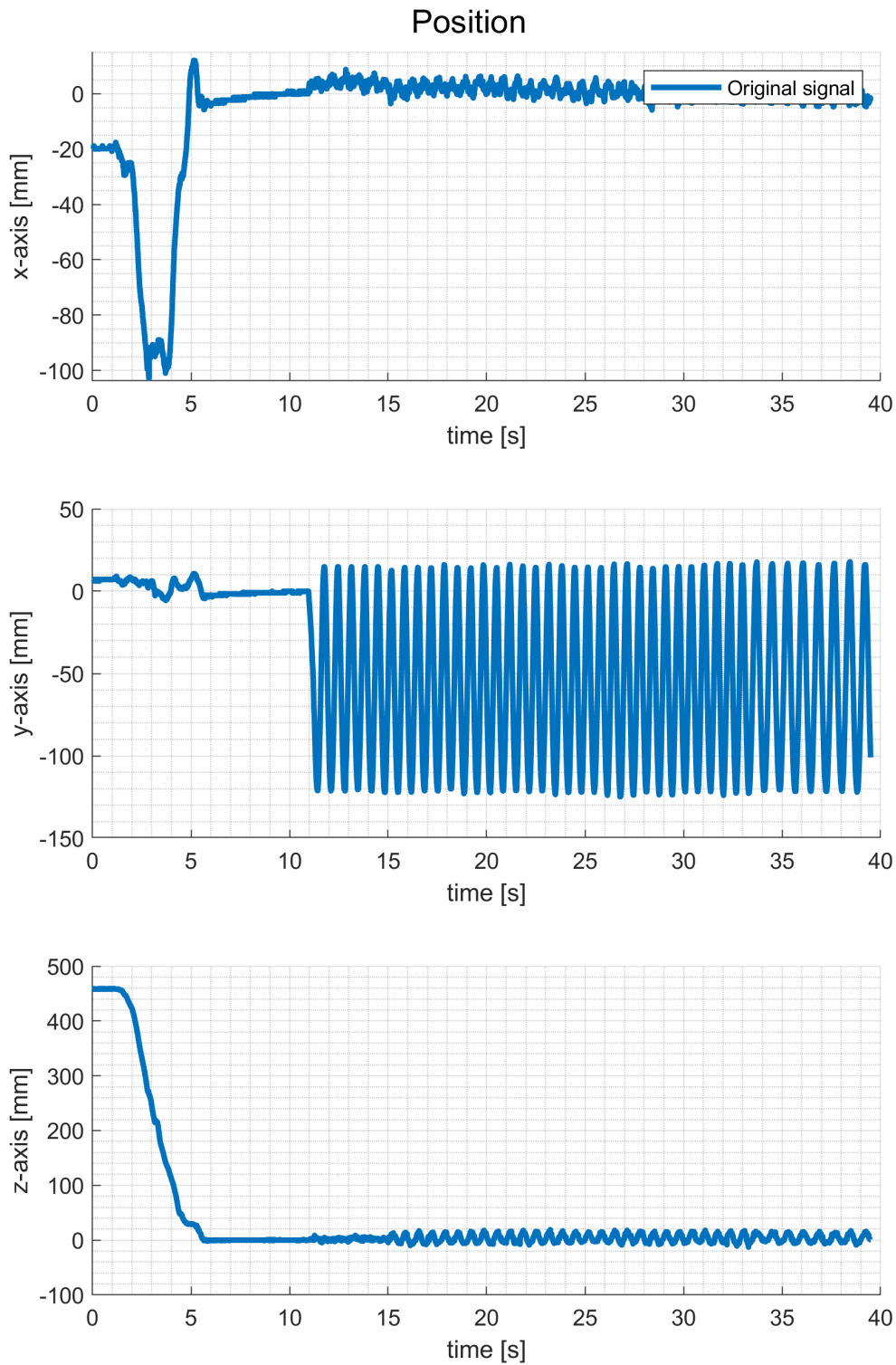


Figure A.63: RealSense T265 experiment, harmonic motion in y-axis (fast), position. With coordinate frames initialization part. Data resemble expected values.

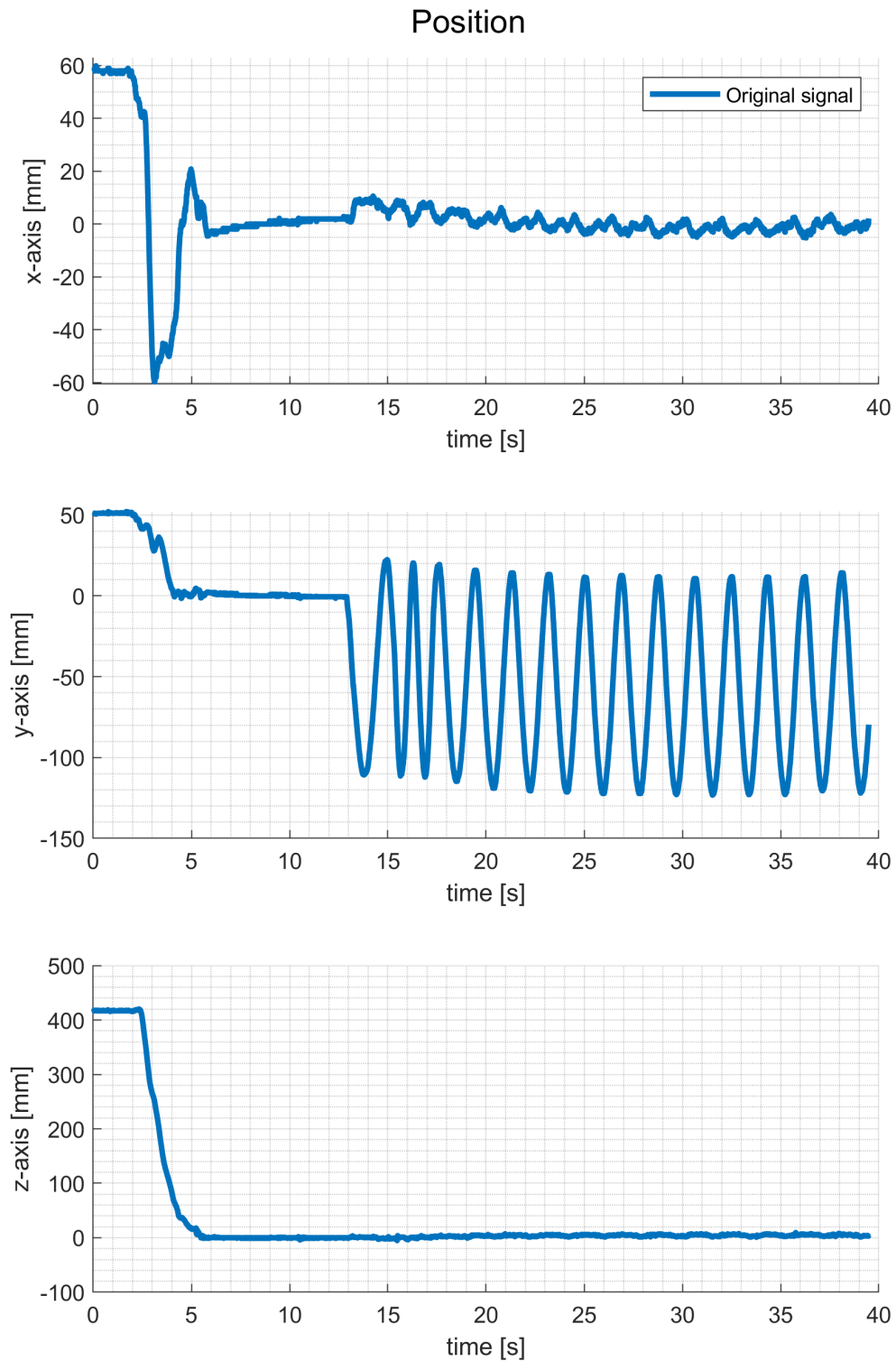


Figure A.64: RealSense T265 experiment, harmonic motion in y-axis (slow), position. With coordinate frames initialization part. Data resemble expected values.

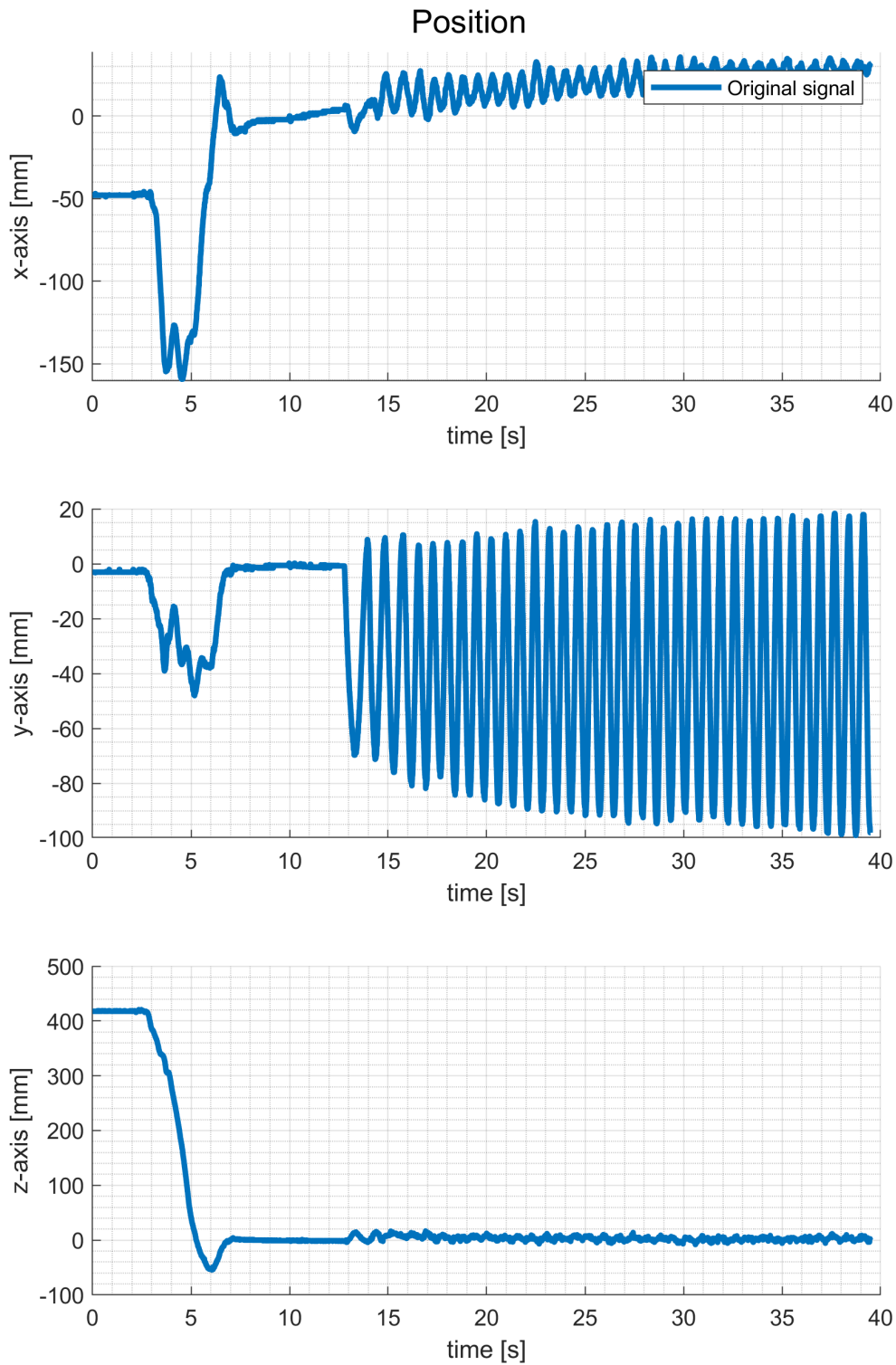


Figure A.65: RealSense T265 experiment, harmonic motion in y-axis (fast), position. With coordinate frames initialization part. Data resemble expected values, but with an observable drift.

■ **A.4 MMR IMU sensor and T265 tracking camera**

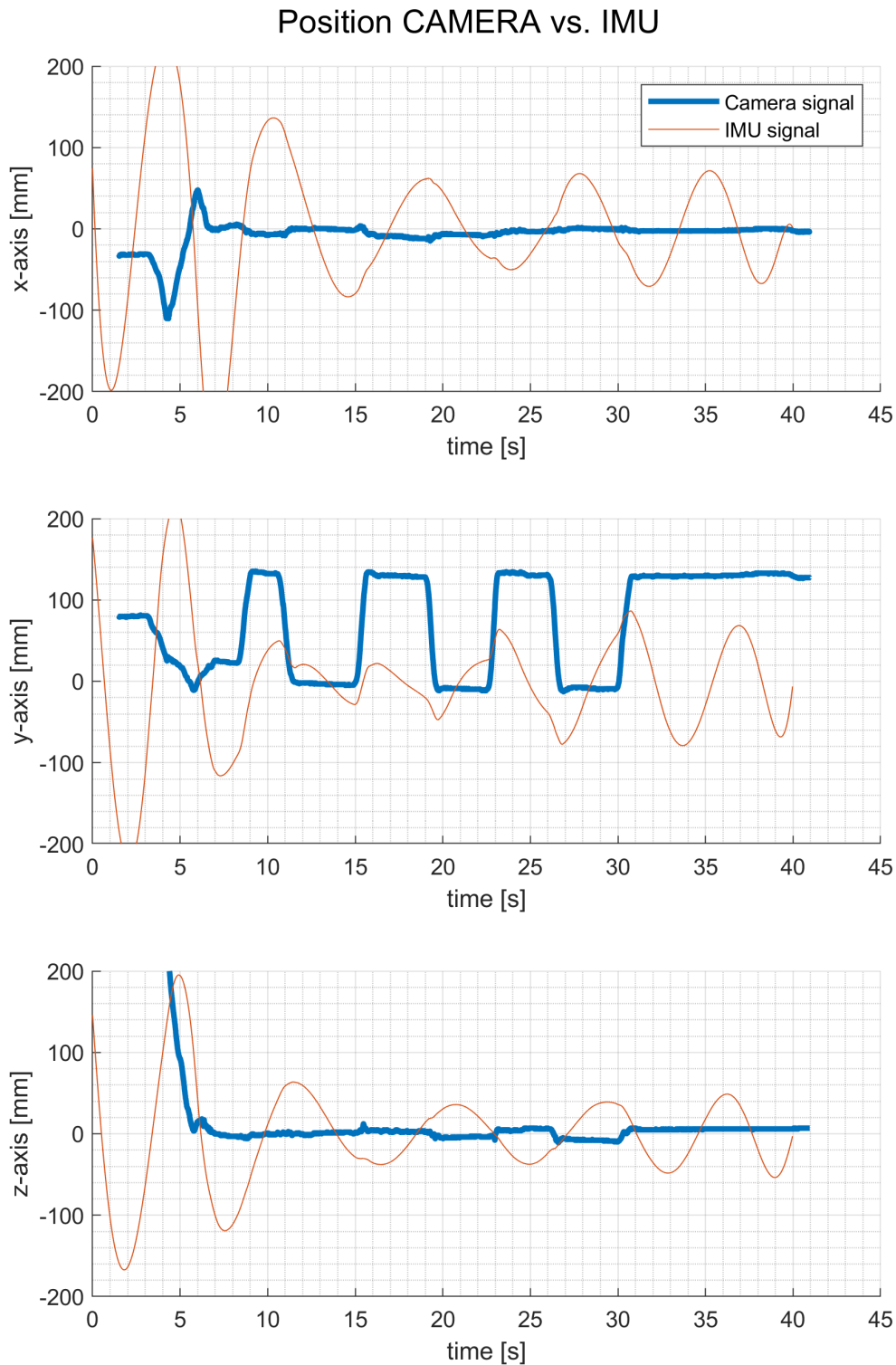


Figure A.66: MMR IMU and RealSense T265 experiment, repeated translation in y-axis, position. T265 shows expected data, MMR follows them only inaccurately.

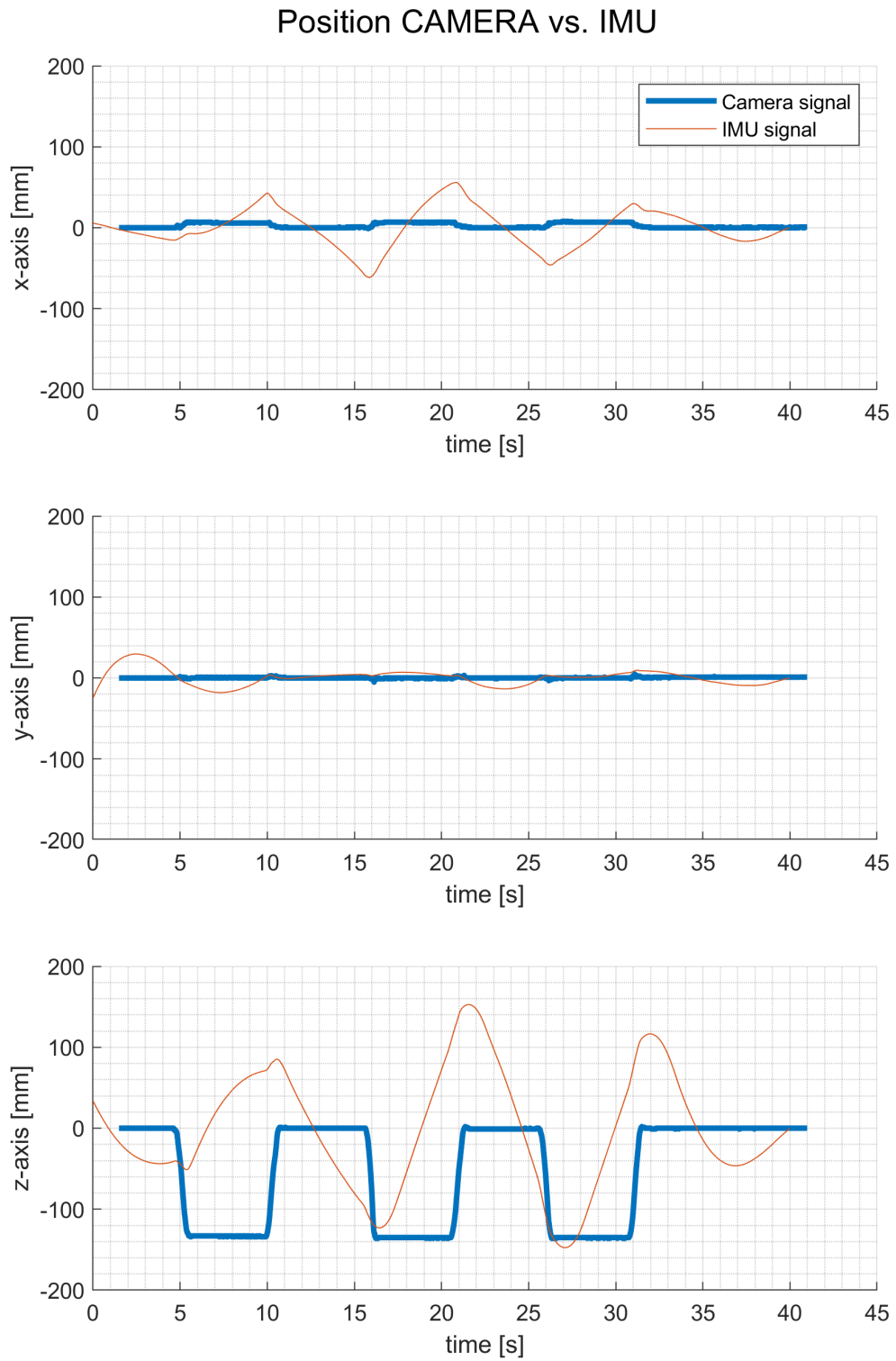


Figure A.67: MMR IMU and RealSense T265 experiment, translation in z-axis, position. T265 shows expected data, MMR follows them only inaccurately.

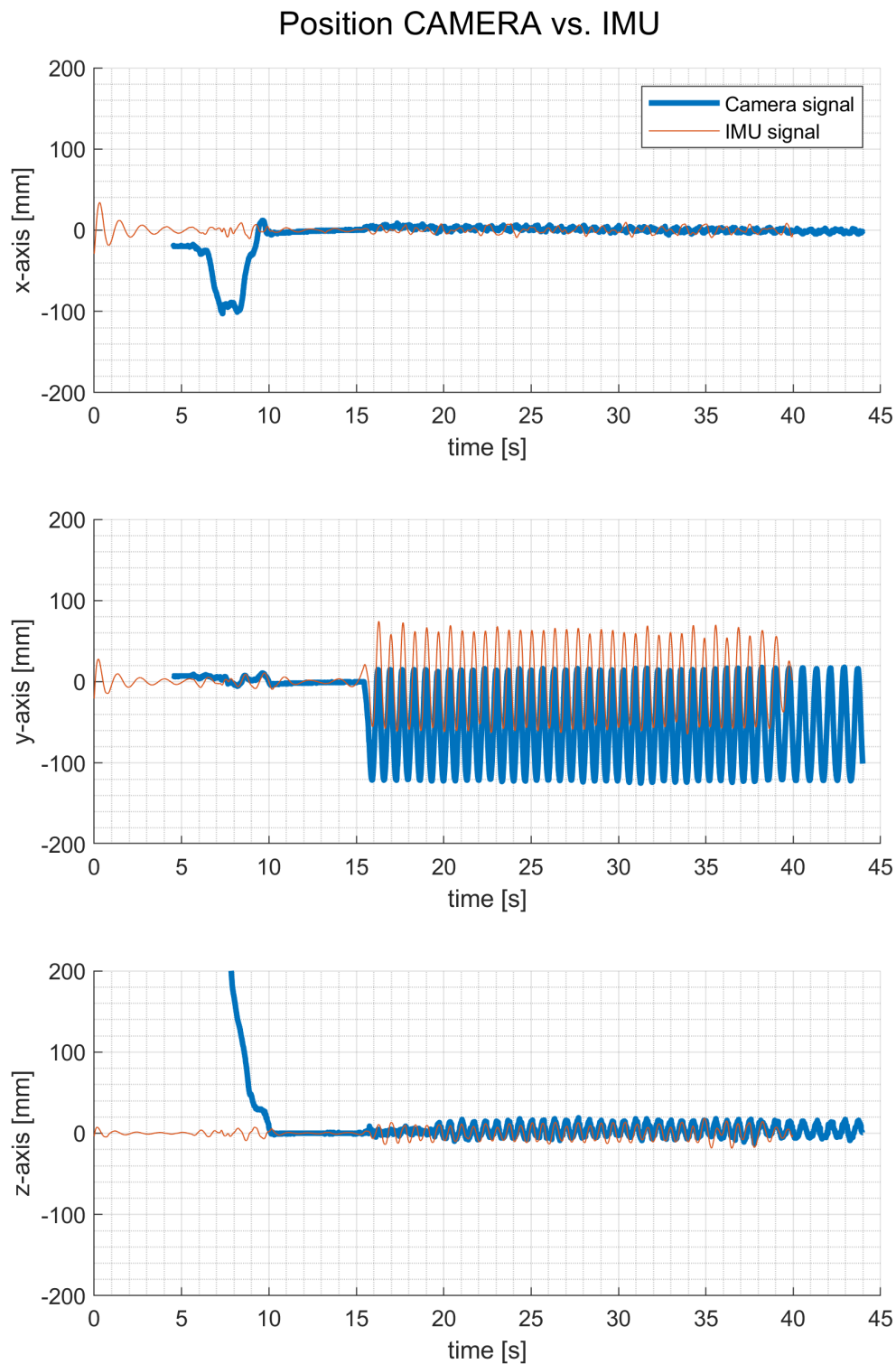


Figure A.68: MMR IMU and RealSense T265 experiment, harmonic motion in y-axis (fast), position. T265 shows expected data, MMR follows them only inaccurately, with offset.

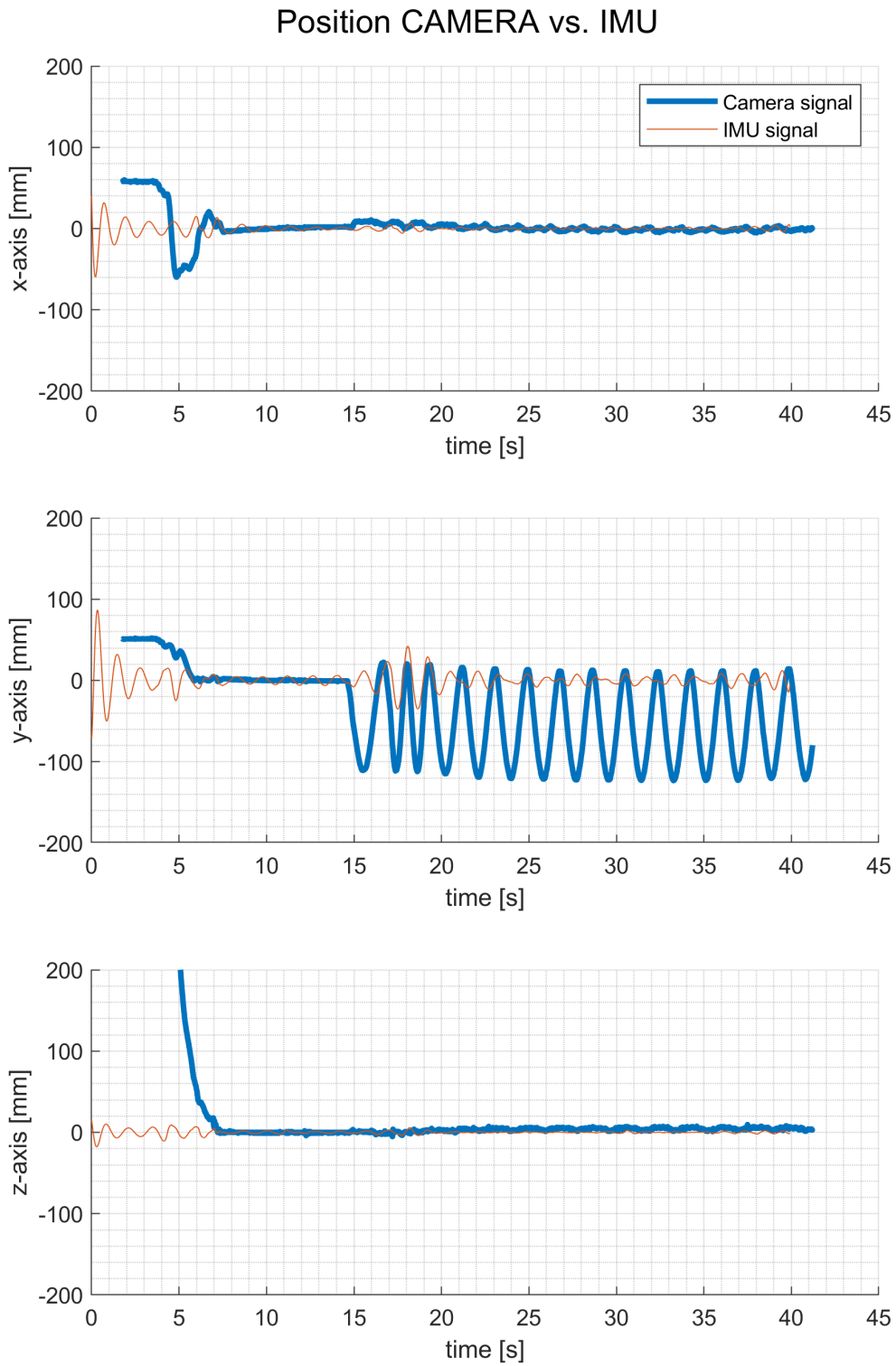


Figure A.69: MMR IMU and RealSense T265 experiment, harmonic motion in y-axis (slow), position. T265 shows expected data, MMR follows them only inaccurately, with offset.

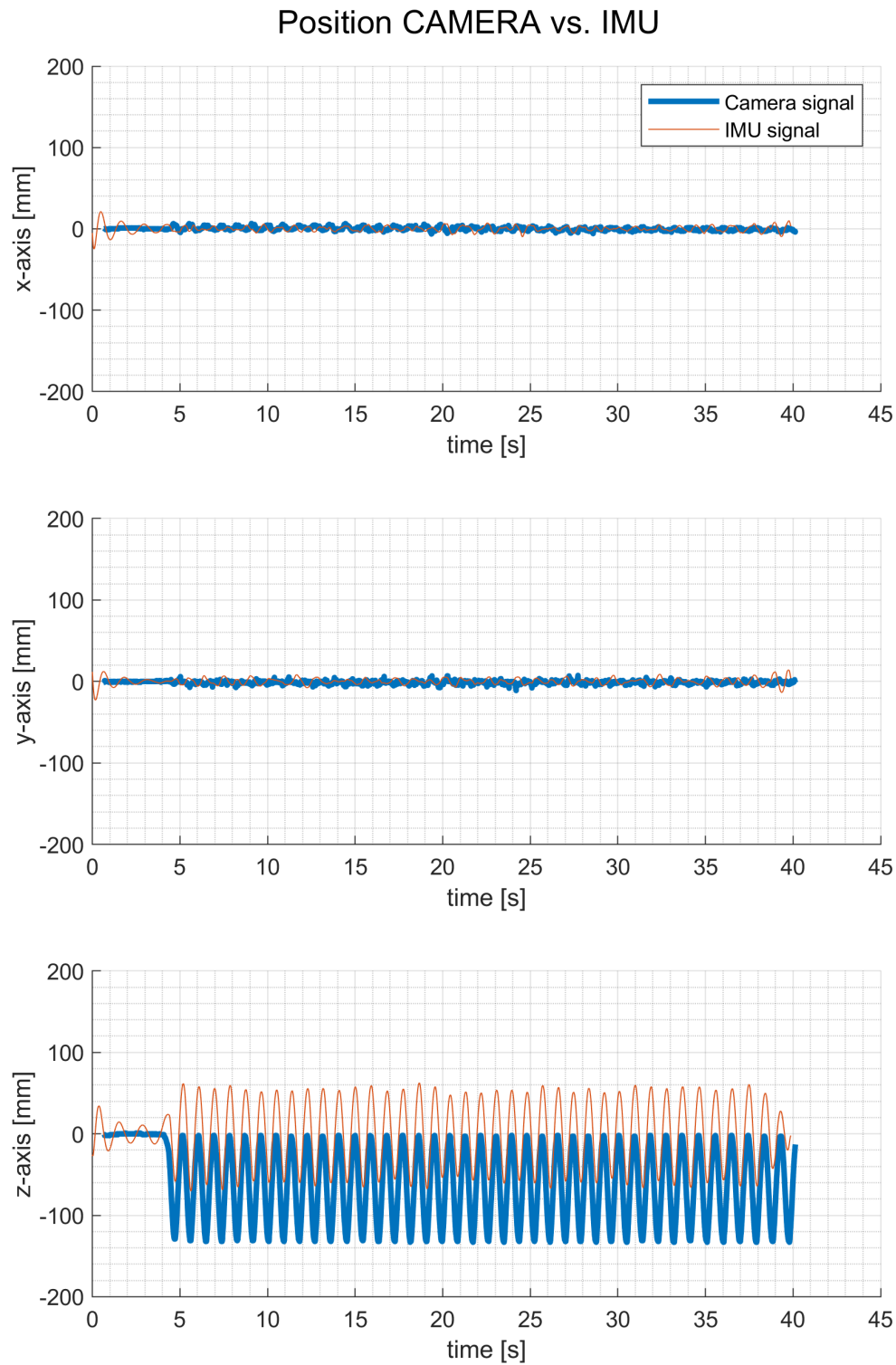


Figure A.70: MMR IMU and RealSense T265 experiment, harmonic motion in z-axis (fast), position. T265 shows expected data, MMR follows them only inaccurately, with offset.

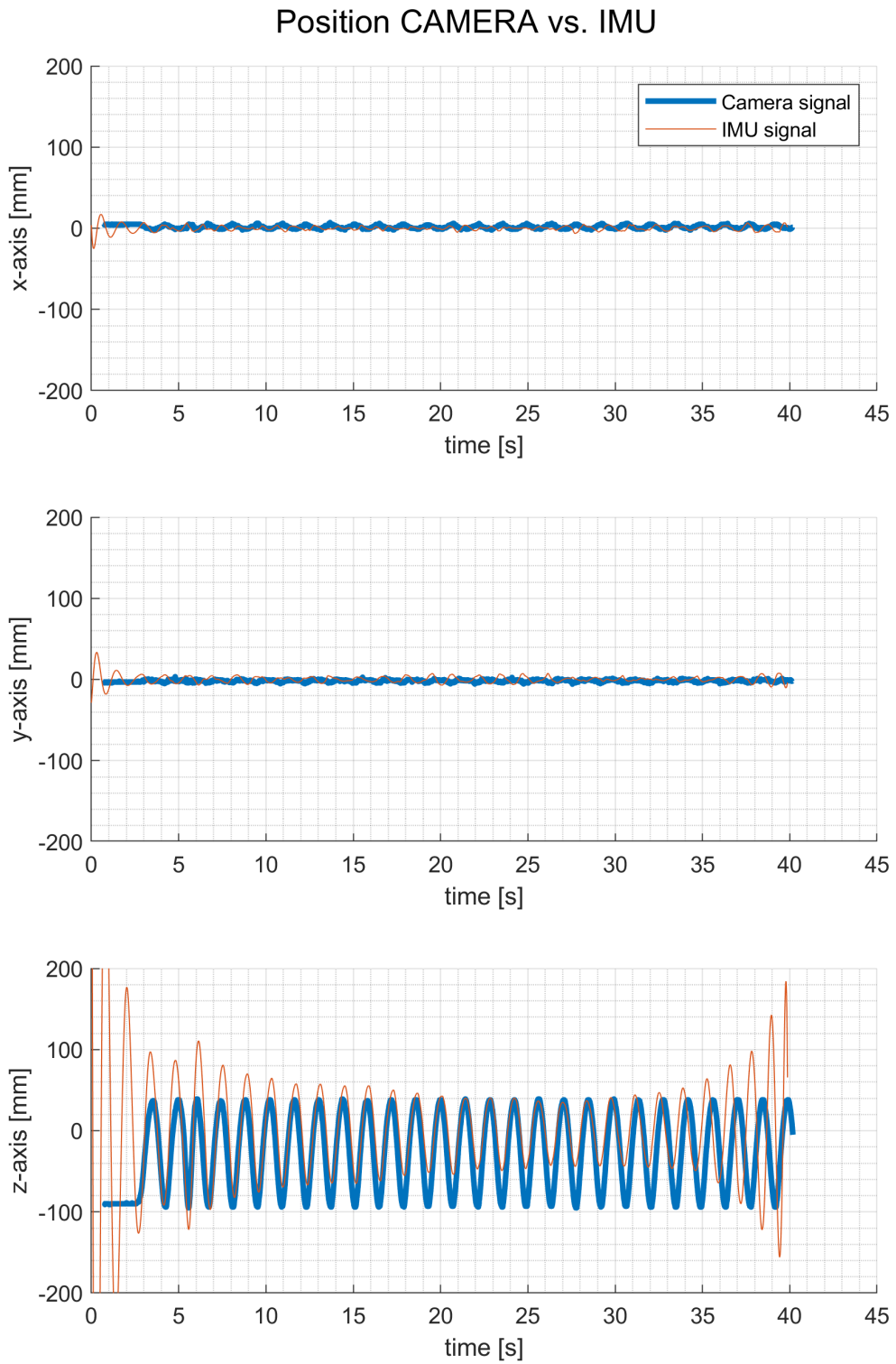


Figure A.71: MMR IMU and RealSense T265 experiment, harmonic motion in z-axis (slow), position. T265 shows expected data, MMR follows them only inaccurately, with offset.



Appendix B

Robot simulation results

■ B.1 Simulation data

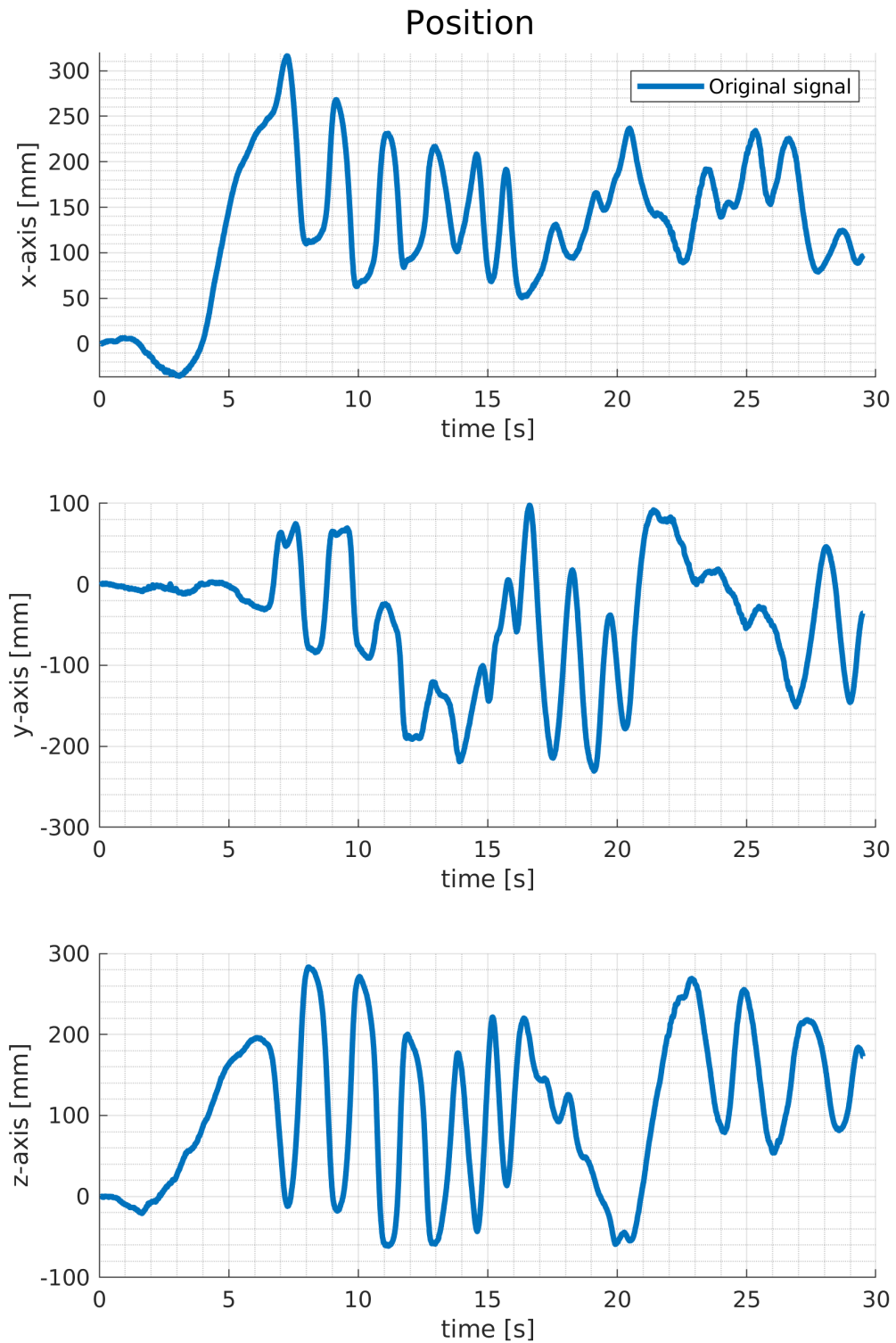


Figure B.1: Trajectory 1 sampled by tracking camera T265 held in hand, random motion imitating painting, 1D.

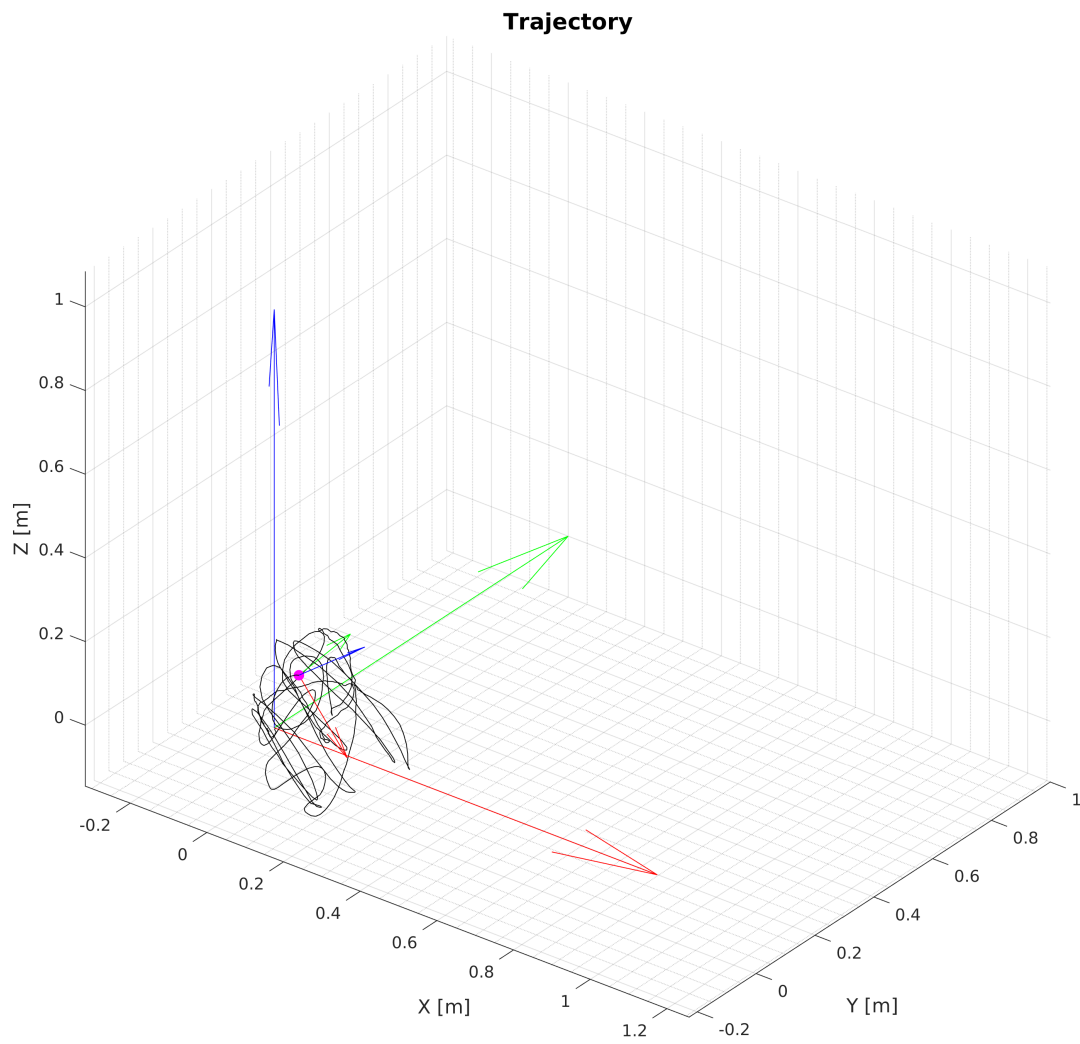


Figure B.2: Trajectory 1 sampled by tracking camera T265 held in hand, random motion imitating painting, 3D.

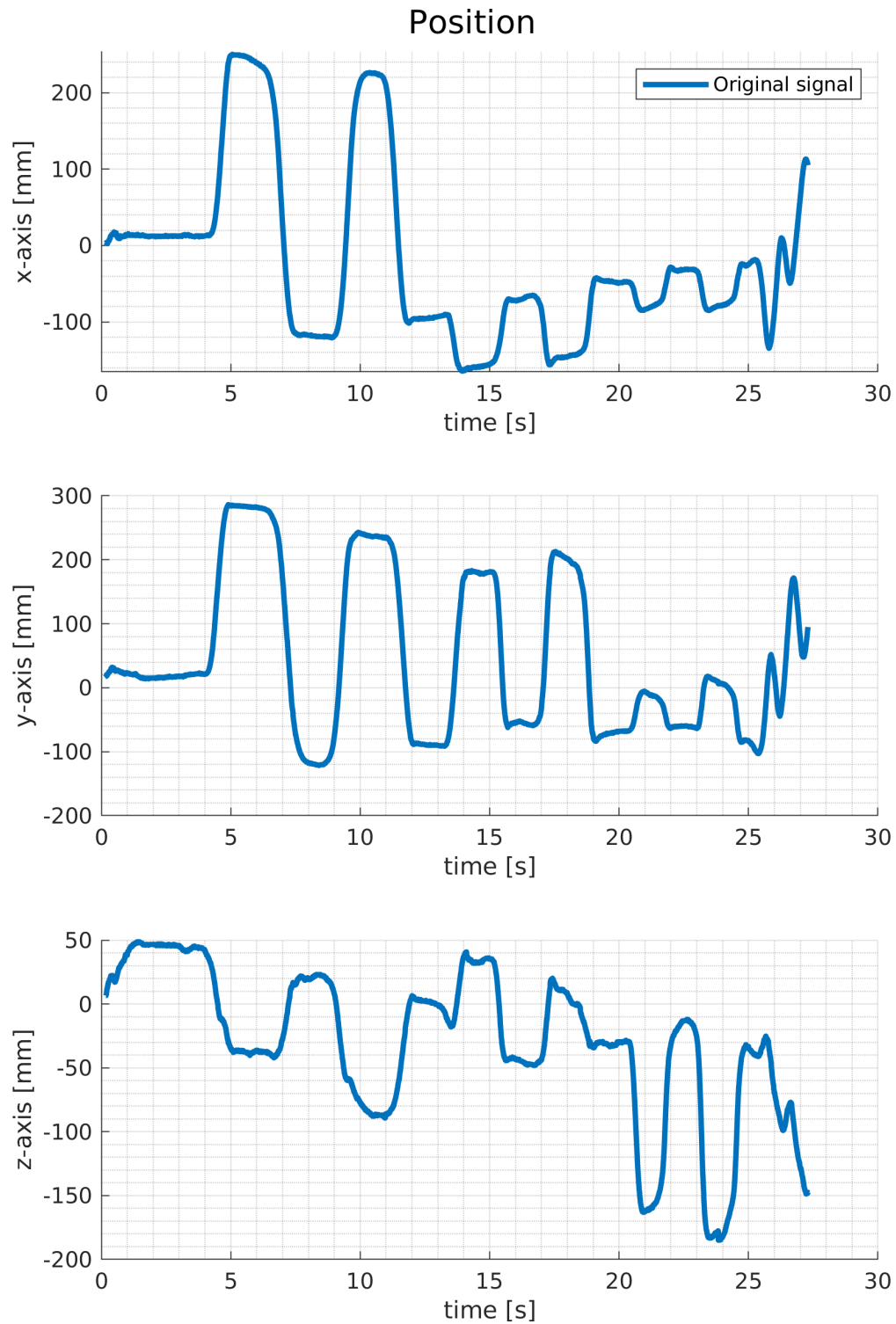


Figure B.3: Trajectory 2 sampled by tracking camera T265 held in hand, random motion imitating painting, 1D.

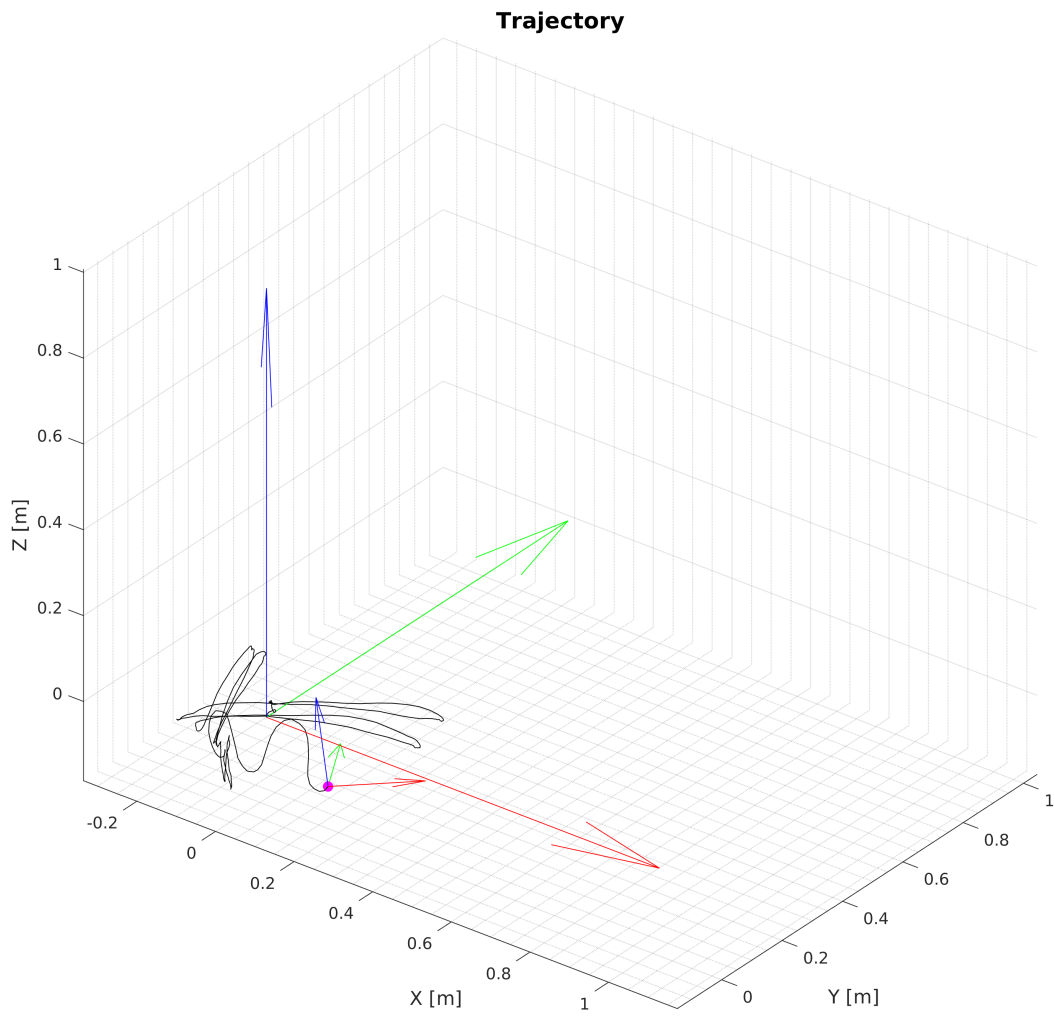


Figure B.4: Trajectory 2 sampled by tracking camera T265 held in hand, random motion imitating painting, 3D.

■ B.2 PyBullet

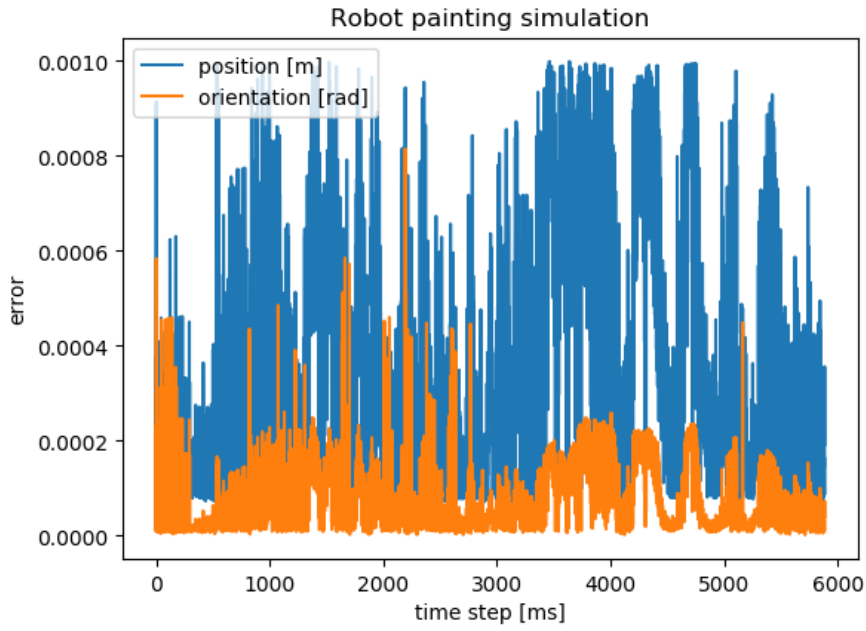


Figure B.5: Error of solutions of inverse kinematics while performing trajectory 1 by KUKA in PyBullet simulation.

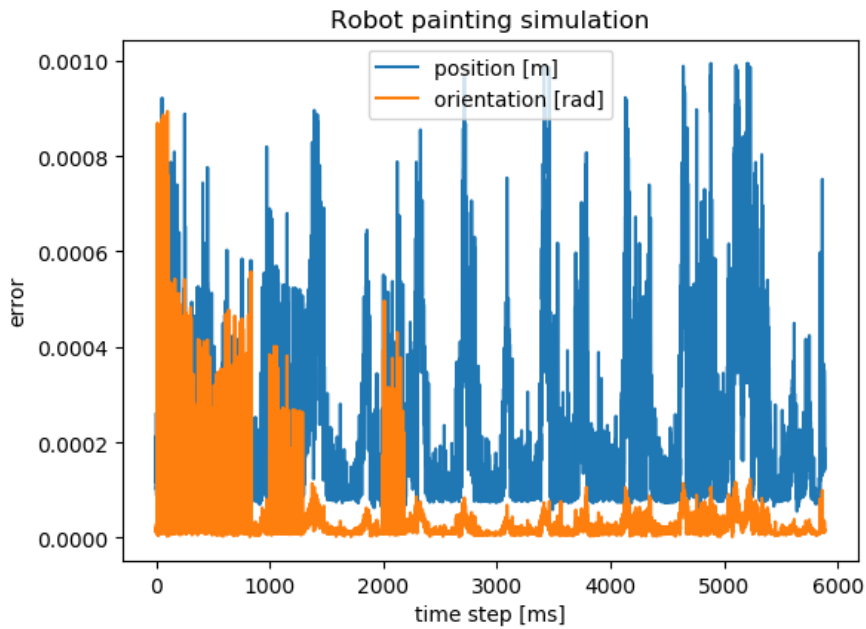


Figure B.6: Error of solutions of inverse kinematics while performing trajectory 2 by KUKA in PyBullet simulation.

Videos of robotic simulation performing trajectory 1 and 2 are available in form of an electronic appendix.







B.3 ROS - Descartes

Videos of robotic simulation performing trajectory 1 and 2 are available in form of an electronic appendix.



Appendix C

Source code

-  **C.1 Laser Trackers post-processing**
-  **C.2 MMR IMU sensor control and post-processing**
-  **C.3 T265 tracking camera post-processing**
-  **C.4 PyBullet KUKA robot simulation**
-  **C.5 ROS - Descartes ABB robot simulation**
-  **C.6 PLC control program**

All source code is available in form of an electronic appendix or on GitHub at <https://github.com/megi-mejdrechova/DP-Robo-Painter>.