TECHNISCHE UNIVERSITÄT MÜNCHEN
Lehrstuhl für Echtzeitsysteme und Robotik

A FRAMEWORK FOR OPTIMAL DYNAMIC MODELING OF SENSING MODALITIES

Artashes Mkhitaryan

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 05.06.2014 bei der Technische Universität München eingereicht und durch die Fakultät für Informatik am 22.10.2014 angenommen.

Life is what happens to you while you're busy making other plans.

— Allan & Saunders

Dedicated to the loving memory of Anahit Mkhitaryan.

1951 – 2013

ABSTRACT

Sensory substitution is a conversion of one sensory modality to another by means of physical or mathematical transformations. Although the expression "*sensory substitution*" is generally used in context of neuroscience, its application range is much wider. Any direct conversion of one sensory modality to another qualifies as substitution of the former. This, involves such topics as digitization of acceleration, temperature, force, etc. through corresponding sensors as well as emulation of one sensor based on the data from another sensor that has undergone a series of physical and/or mathematical transformations.

Sensory substitution in its broad meaning is one of the major topics of modern natural sciences. Although it has undergone a substantial growth during the course of its modern history, there are many issues that remain open. Due to the fragmented nature of the sensory substitution each discipline dealing with it treats the problems in its narrow scopes. Since sensory substitution is a multidisciplinary topic, this results in issues of choosing the correct interfaces that represent the transition between them. The problem of choosing the correct physical and mathematical transformations to get from one modality to another does not always have a unique solution. Thus, it is hard to estimate which solution is the most optimal. Sometimes the estimated solution is optimal for achieving results with minimum error; however, its complexity is high enough to make it unusable in context of a sensor that provides real time results. Other times the opposite is true, the complexity is low enough to provide real time data; however, the error accumulation amounts to large values that make the further processing of the data unfeasible. Finally, there are cases where multiple solutions are present that provide reasonable results; however, some are better in a certain set of conditions and others are better in other set of conditions. It is hard to build systems that would account for all of them.

In this thesis, we present a framework for optimal registration of different sensory modalities to each other. Our system is capable of finding the most advantageous set of mathematical and physical transformations between the initial modalities and the target modalities. It takes into account the requirements of the problem and can optimize for minimum complexity, minimum error, or strike a balance in optimizing for both in the resulting fusion. Our framework allows the creation of virtual sensors, that may extend the sensing modalities of the current setup. The framework is capable of increasing the reliability of acquired data in multi-sensor systems by being able to asses the amount of accumulated errors. We give two examples of real-world applications of this framework in robotic environments as well as demonstrate how the errors estimated through our framework compare to real world errors.

ZUSAMMENFASSUNG

Sensor Substitution ist die Umwandlung einer Sensormodalität in eine andere durch die Verwendung physikalischer und oder mathematischer Transformationen. Obwohl Sensor Substitution generell im Zusammenhang mit Neurowissenschaften benutzt wird, ist dessen Anwendung nicht auf dieses Gebiet begrenzt, sondern wesentlich breiter einsetzbar. Jede direkte Umwandlung einer Sensormodalität in eine andere kann als Substitution bezeichnet werden. Das beinhaltet Themen wie das Erfassen von Beschleunigung, Temperatur, Kraft, etc. durch die dazugehörigen Sensoren sowie die Emulation eines Sensors basierend auf Daten eines anderen Sensors, die physikalisch und oder mathematisch transformiert worden ist.

Allgemein ist Sensor Substitution einer der wesentlichen Themen der modernen Naturwissenschaften. Obwohl die Sensor Substitution in seiner jüngsten Geschichte ein substantielles Wachstum vollzogen hat, existieren nach wie vor etliche Themenbereiche, die noch nicht behandelt wurden. Aufgrund seiner Fragmentierung behandelt jeder Zweig das Problem in seinem eigenen eingeschränkten Bereich. Da die Sensor Substitution ein fachübergreifendes Thema ist, folgt daraus, dass das Problem die richtige Wahl des Interfaces ist, welches den Übergang zwischen beiden darstellt. Das Problem der richtigen Auswahl der physikalischen und mathematischen Transformation, um von einer Modalität zu einer anderen Modalität zu gelangen, beinhaltet nicht immer nur eine Lösung. Dadurch ist es schwer zu bestimmen, welche die optimale Lösung ist. In manchen Fällen ist die zugrundegelegte Lösung das Optimum, um die geringsten Fehler zu gewährleisten und dennoch ist die gegebene Komplexität zu hoch, um sie im Zusammenhang mit einem Sensor zu nutzen, das Echtzeitergebnisse liefert. Das Gegenteil kann auch der Fall sein: Die Komplexität ist gering genug, um Echtzeitergebnisse bereitzustellen allerdings ist die Anhäufung der Fehler zu hoch, sodass die weitere Bearbeitung der Daten nicht möglich ist. Es gibt auch Fälle in denen mehrere Lösungen vorkommen, die vernünftige Lösungen bereitstellen. Einige Lösungen sind unter bestimmten Bedingungen besser anzuwenden und andere Lösungen unter anderen Bedingungen, daher ist es schwer, Systeme zu bauen, die für den gesamten Lösungsraum darstellbar sind.

In dieser Dissertation wird ein Framework zur optimalen Registrierung von unterschiedlichen Sensormodalitäten präsentiert. Unser System ist in der Lage, die beste Kombination mathematischer und physischer Transformationen zwischen der Initialen- und der Zielkonfiguration zu finden. Es berücksichtigt die Anforderungen des Problems und kann minimale Komplexität und minimale Fehler gewährleisten.

Unser Framework gestattet die Erzeugung virtueller Sensoren, die Erfassungsmodalitäten des aktuellen Falls erweitern können. Das Framework ist imstande, die Zuverlässigkeit der in Multi-Sensor-Systemen erworbenen Daten zu erhöhen, indem die Menge der angehäuften generierten Fehler bewertet werden. Wir geben zwei Beispiele von Real-World-Applikationen dieses Framework in der Robotik und zeigen wie sich Fehler, die durch unser Framework geschätzt werden, mit Real-World Fehler vergleichen lassen.

A rabbit that was running through the forest gets caught by a wolf:

Rabbit: *Please do not eat me today, I have my PhD defense tomorrow. I spent 4 years of my life writing the dissertation, and I do not want to die a M.Sc.*

Wolf: *Whats is your topic?*

Rabbit: *"Superiority of Rabbit over Wolf"*

The wolf, intrigued by the title, lets the rabbit go and decides to attend the defense. Next morning, once the wolf enters the room, a lion jumps out and eats him.

Moral: *It does not matter what your topic is, what matters is who your Professor is.*

ACKNOWLEDGMENTS

First of all, I would like to thank Prof. Drius Burschka for allowing me to work in his chair, providing me with guidance, expertise and support. I would like to express my gratitude to Dr. Konstantinos Dalamagkidis for helping me with paper writing and being a friend and colleague.

Many thanks to my family and particularly to my wife Alina Mihaela Roescu for standing next to me during hard times. I would not be able to complete my work without their love and support.

Last but not least, I would like to express my gratitude to all of my colleagues at Garching-Hochbrück. I enjoyed every second of working with you.

CONTENTS

1 INTRODUCTION 1
  1.1 Sensory Substitution 1
  1.2 Sensory Substitution vs Sensor Data Fusion 3
  1.3 Previous Work in Sensory Substitution 5
  1.4 Contributions 8
  1.5 Structure of the Thesis 9

i THE FRAMEWORK 11

2 OVERVIEW OF FRAMEWORKS 13
  2.1 Overview of Frameworks in Computer Science 13
  2.2 Overview of the Framework for Sensory Substitution 15

3 CONNECTED GRAPH 19
  3.1 Choice of the Modes and Optimization Parameters 19
    3.1.1 Dynamic Mode 19
    3.1.2 Static Mode 20
  3.2 Creation of the Connected Graph 21
    3.2.1 Connection of Nodes 22
  3.3 Description of Nodes 24
    3.3.1 Transformation Blocks 24
    3.3.2 *Start* Nodes 28
    3.3.3 *Goal* Nodes 29
    3.3.4 Error Profiles 29
  3.4 Propagation of Information Through the Graph 31
    3.4.1 Data Propagation 31
    3.4.2 Uncertainty Propagation 34
    3.4.3 Complexity Propagation 35
  3.5 Edges and Weights 41
    3.5.1 Additive Weights 43
    3.5.2 Multiplicative Weights 44

4 SEARCH THROUGH THE GRAPH 47
  4.1 Introduction and Overview 47
  4.2 Dijkstra's Search Algorithm 47
  4.3 Extended Dijkstra's Algorithm 48
  4.4 Path Estimation for Different Optimization Parameters 49
    4.4.1 Optimization for Minimum Error 52

4.4.2 Optimization for Minimum Complexity 52
4.4.3 Optimization for Both Minimum Complexity and Minimum Error 54
4.5 Optimization Modes 54
4.6 Specifications of the Transformation Graph 56

ii APPLICATION EXAMPLES 59

5 SHORT INTRODUCTION TO THE EXPERIMENTAL VALIDATION 61

6 VALIDATION OF THE ERROR ESTIMATION ACCURACY 63
6.1 Introduction and Overview 63
6.2 Transformation Graph Based Analysis 65
6.3 Experimental Validation 65
6.4 Experimental Results 68

7 OPTIMIZATION FOR MINIMAL ERRORS IN DYNAMIC MODE 71
7.1 Problem Statement 71
7.2 The Transformation Graph 71
7.3 Computation of Error Profiles 73
7.3.1 Error Profiles of the "3D Reconstruction" Transformation 74
7.3.2 The Error Profile for Velocity and Acceleration Computation 76
7.3.3 Error Profile of the Cellphone Accelerometer 76
7.4 Final Result 77

8 OPTIMIZATION FOR BOTH MINIMUM ERROR AND COMPLEXITY IN STATIC MODE 79
8.1 Introduction and Overview 79
8.2 Transformation Graph Based Analysis 79
8.3 Physical Implementation 82
8.3.1 Physical Setup 82
8.3.2 3D Reconstruction of the Surface of the Membrane 83
8.3.3 Computation of Force Distribution 87
8.4 Experimental Results 88

9 CONCLUSION AND FUTURE WORK 95
9.1 Conclusion 95
9.2 Future Work 96

BIBLIOGRAPHY 101

LIST OF FIGURES

Figure 1 Paul Bach-y-Rita demonstrating a prototype of a device for substituting vision through the feeling of touch. The image is taken from [50]. 2

Figure 2 Visualization of the process of sensory substitution. Here the red and cyan dots depict the initial and target modalities respectively. The gray dots depict the individual transformations. 2

Figure 3 Challenges of sensory fusion illustrated on the example of stereoscopic reconstruction. In all of the images the light green areas represent the space that is visible to one camera, the dark green area represents the space that both cameras can see. The pink area represents the space where a particular pixel detected by the camera can be located. Note that regular cameras detect the z dimension (depth) of the pixel with infinite uncertainty. The red areas represent the intersection of the spaces from both cameras where the particular pixel can be located. 4

Figure 4 An example of a virtual force sensor constructed using our framework. The latter is realized by an optical camera, which registers the deformations of a plastic membrane. 8

Figure 5 Visualization of the graph based sensory substitution. The transformation nodes are depicted in gray. Initial and target modalities are depicted in red and cyan respectively. 15

Figure 6 Visualization of the classified paths in the graph. Each color represents a single path that was given a certain classification of the traversal quality. 15

Figure 7 Transformation Graph: The edge weights are computed based on the inputs, complexity and errors of the corresponding nodes. An optimal chain is obtained by using extended Dijkstra's search algorithm. 16

Figure 8 Sets of steps for the construction of the graph 17

Figure 9 (A) An example case for usage of the dynamic mode. The task is to increase the reliability of acceleration readings of an already constructed multi-sensor platform. (B) An example case for usage of the Static mode. The task is to construct a 3D surface force sensor, using the set of available initial sensors. 20

Figure 10 Illustration of all the three types of nodes, that are a part of a *transformation graph* . First node on the left is a *Start* node, those have only an output section, thus edges can only point out of them. Second node in the middle is a regular transformation node. Those nodes have both input and output sections, therefore edges can both point into them and out of them. Third node on the right is a *Goal* node. Those nodes have only an input section, hence edges point only into them. 21

Figure 11 Visualization of the edge insertion process. The edges are inserted between corresponding input and output pairs. 22

Figure 12 An illustration of the graph connection process. The nodes of the graph are depicted on the right side. During the first loop over the nodes the indices of the modalities are stored in a 1 dimensional array that contains two vectors: Input and Output. The address of each modality is computed using a hash function after which the index is either pushed into the Input or Output vector depending on the location of the considered modality. Further, a loop is performed over the array, and edges are inserted into the graph based on the indices of Input and Output vectors (Algorithm 2). 23

Figure 13 Different classes of transformation blocks. (A) A transformation with a basic input block. Those are linear transformations that require only one value as its input argument. Here b is the multiplicative weight that is normally set to one for these blocks. (B) A transformation with an iteration dependent input block. Those represent the first class of non linear transformations that require sequential values of the input argument from the previous and current iterations. Here ξ is the multiplicative of the weight that is higher than one due to the non linear nature of the transformation. (C) Transformations with high-order input blocks. Those require multiple consecutive values of the input argument, and do not retain state. Here Ω is the multiplicative weight, usually higher than two. In all of the cases above e is the additive weight. All of the weights depend on the optimization parameters of the problem and will be discussed in Section 3.5. 26

Figure 14 An example of a *Start* node creation, from a given set of initial sensors. 28

Figure 15 Illustration of the information propagation mechanics though the graph. The square boxes represent basic transformations, the eight edge star represents an iteration dependent transformation. The twelve edge star represents a high-order transformation, which for simplicity reasons is of second order. 32

Figure 16 Illustration of the error propagation model through the transformation chain. A) Depicts the transformation chain itself. The latter consists of sequential basic transformations that are all equivalent. Each transformation in the chain implements the "$(A/2) + 1$" simple equation. A complete pseudo-code implementation of the transformation chain is given in Algorithm 3 34

Figure 17 Influence of the high-order transformation blocks on the complexity of the transformation chain. Illustration of a sequential case where the initial sensors operate at a comparable frequency to the processor and the transformations take equal amount of time to perform. 37

Figure 18 Influence of the high-order transformation blocks on the complexity of the transformation chain. Illustration of a sequential case where the initial sensors operate at a significantly slower frequency compared to the processor and the transformations take equal amount of time to perform. 39
Figure 19 Influence of the high-order transformation blocks on the complexity of the transformation chain. Illustration of a sequential case where the initial sensors operate at a comparable frequency to the processor and the transformations take different amounts of time to perform. 40
Figure 20 Illustration of the process of assigning additive weights to the edges for the case of error optimization. A generic transformation graph is depicted on the right. The error profiles of the transformation blocks are depicted on the left. 42
Figure 21 4.21(a) A depiction of the first step of the search algorithm, here all of the path distances are assigned to inf expect the *Start* node. 4.21(b) A depiction of an intermediate step of the search algorithm. The step was selected to illustrate the handling of multiple inputs of a transformation. 4.21(c) The final result, after all the distances to the goal have been computed. 50
Figure 22 Effects of weight assignment 51
Figure 23 Optimization for minimum error and complexity 53
Figure 24 A Generic example to help illustrate the work-flow of the framework in different modes. The system consists of two sensors that have one overlaying modality "B". This results into two *Start* nodes that are illustrated here as one for clarity purposes. 55
Figure 25 Illustration of the setup for the experiments conducted to estimate the relation of the real world errors to the computed global path distance. A ball was let to roll freely on the flat surface. The acceleration of the ball was measured for different angles of tilt. 63
Figure 26 The *transformation graph* for estimating the velocity of a rolling ball. The framework is set to operate in dynamic mode with the optimization set to only minimize errors. Some of the transformations introduce no errors therefore their additive values are set to zero in advance. 64

Figure 27 Sketch of the experimental setup. The ball was let free to roll on the ramp, influenced only by gravity. The experiment was repeated with different slopes. 66

Figure 28 Illustration of the experimental results. Here the ball was let free to roll on a surface with a fixed slope. Figure 6.28(a) illustrates velocities of the ball that were computed using different time-steps. Figure 6.28(b) illustrates the resulting relative errors for different time-steps. 67

Figure 29 Illustration of the experimental results. Here the ball was let free to roll multiple times on a surface. For each roll the slope of the surface was altered. Figure 6.29(a) illustrates the estimated velocities for different accelerations. Figure 6.29(b) illustrates the occurring relative errors. 69

Figure 30 Depiction of the transformation graph designed to estimate the best acceleration value of a cellphone. There are two ways to perform. First, by reading the data directly from the accelerometer located on the cellphone. Second, by computing the accelerations based on the images acquired by the camera located on the cellphone. 72

Figure 31 7.31(a) Sketch of the experimental setup for obtaining the error profile. Note that the camera is placed in such a way that the effective distance to the static and dynamic markers remain practically constant. 7.31(b) Illustration of the experiment for computing the [error/target distance] dependency. 7.31(c) Illustration of the experiment for computing the [error/viewing angle] dependency. 73

Figure 32 7.32(a) Illustrates the error dependency of the 3D reconstruction algorithm to the distance between the marker and the camera. 7.32(b) Illustrates the error dependency of the 3D reconstruction algorithm to the angle of view. Here the presented angle is the out-of-plane angle between the cellphone and the surface of the marker. Note that the in-plane rotation angels are not presented since they do not add any errors. 75

Figure 33 The error profile of the accelerometer that is embedded in the cellphone. Here the red line illustrates the standard deviation of the errors that is $1.43 m/s^2$. 76

Figure 34 Final analysis for the two acceleration registration methods. The green dots represent path distances of the data registered by the accelerometer located on the cellphone. The red dots represent the path distances of the accelerations computed from the camera images. 77

Figure 35 7.35(a) Illustration of the registered accelerations. Here the data registered from the industrial accelerometer (XSens) is plotted in blue. The reconstructed accelerations from the cellphone-camera are plotted in red and the accelerations registered from the accelerometer located on the cellphone itself are in green. Note that for lower accelerations the error from the cellphone accelerometer is larger than the error from the camera; however, for large accelerations this changes. To ensure a better observability of the previous statement we provide two zoomed sections 7.35(b), 7.35(c) of the 7.35(a) graph. 78

Figure 36 Figure 8.36(a) illustrates the blueprint of the finger: (1) LED, (2) CCD camera, (3) a valve to regulate the amount of air within the frame, (4) rigid circle markers, (5) rubber skin surface, (6) the airtight frame of the finger, (7) glass, (8) air. Figure 8.36(b) illustrates the constructed prototype of the sensitive fingertip sensor. 80

Figure 37 Transformation graph for a sensitive fingertip sensor. Here the system is set up to sense forces using a regular CCD camera. The graph contains two possible transformation chains. The first chain (marked with cyan dots) contains the set of transformations necessary to sense forces using Hook's law. The second chain (marked with red dots) contains the set of transformations necessary for sensing forces using Newton's second law. 81

Figure 38 Step by step illustration of the membrane surface reconstruction process. The images have been copied from [1] 84

Figure 39 Here C is the camera center, Π is the image plane, a and b are two points on the ellipse, and line ab passes through d. The latter is the projection of circle center D, $AD = DB$. $\rho = 180 - \phi - \omega$ 85

Figure 40 Computation of the force acting on a node. Here $\mathbf{F} = \mathbf{F}_1 + \mathbf{F}_2 + \mathbf{F}_3 + \mathbf{F}_4$ 87

Figure 41 Image 8.41(a) illustrates the locations of estimated (blue dots) and known (red dots) circle centers. Image 8.41(b) illustrates the computed (blue dots) and known (red dots) absolute distances of the circles to the camera center. 88
Figure 42 8.42(a) Illustrates the relation of the computed relative integral force to measured force. Note that it is clearly visible that the relation between them is linear. Thus, the applied Hook's model within the scopes of this experiment holds. 89
Figure 43 Figures 8.43(c), 8.43(d) and 8.43(e) illustrate the captured and processed images from the CCD camera for the three different objects, and Figures 8.43(f), 8.43(g) and 8.43(h) illustrate the respective 3D reconstructions and force distributions. Here the direction of the force vectors are inverted to ensure better visibility. The pictures are taken from [1] 93

LIST OF ALGORITHMS

Algorithm 1 A naive algorithm for connecting the *transformation graph*. The algorithm requires as an input the unconnected nodes. The latter will be referred to as: Graph 22
Algorithm 2 Hash function based algorithm for connecting the *transformation graph*. The algorithm requires as an input the unconnected nodes. The latter will be referred to as: Graph 25
Algorithm 3 Implementation of the transformation chain in Figure 16 A) 35
Algorithm 4 Implementation of the transformation chains in Figure 17 A) 36
Algorithm 5 Dijkstra's search algorithm for finding the shortest path in a connected graph. The algorithm takes as an input a connected graph, the index of the source node and the index of the target node. **Inputs:** Graph, source, target 48

1 INTRODUCTION

1.1 SENSORY SUBSTITUTION

DEFINITION : By definition sensory substitution is a conversion of one sensory modality into another by means of transformations. Where "sensory modality" is defined as a particular way of sensing, such as vision or hearing. The word "modality" shall not be limited to the mentioned examples and can be extended to any mode (state), such as special dimensions, force, velocity, smell, etc. The words "sensory substitution" are mainly used in medical lexicon under the context of restoring a certain defective sensory modality of a handicapped person through another sensory modality. This is realized either biologically i.e. brain plasticity [9], or through some third party devices, that is artificial skin [11], video camera [34] etc (Figure 1). However, the range of applications covered by sensory substitution is much wider than prosthetics. Any conversion of one sensory modality into another qualifies as substitution of the former. Such examples involve digitization of force, acceleration, torque, temperature, etc. using various sensors. Other examples incorporate substitution of one sensor by another though a chain of mathematical and or physical transformations such as force estimation from sequential camera images [1] or distance through time of flight. Due to its wide spread sensory substitution is one of the most important topics in a vast range of natural sciences.

LIMITATIONS : Paul Bach-y-Rita, the man who can be considered the father of modern sensory substitution in context of neuroscience, wrote in his book “Brain Mechanics in Sensory Substitution (1972)” [50]

> “This monograph thus risks becoming outdated in a very short time since the development of refined sensory substitution systems should allow many of the questions raised here to be answered, and some of the conclusions may appear naive to future readers.”

Although sensory substitution in its broad definition has substantially matured during the course of its modern history, it is still far from being perfect. One of the largest problems of sensory substitution is its fragmented nature. Problems are treated on a case by case basis and held strictly within the narrow bounds of the discipline they occur in. The direct effect of this is what is widely known as “Reinventing the

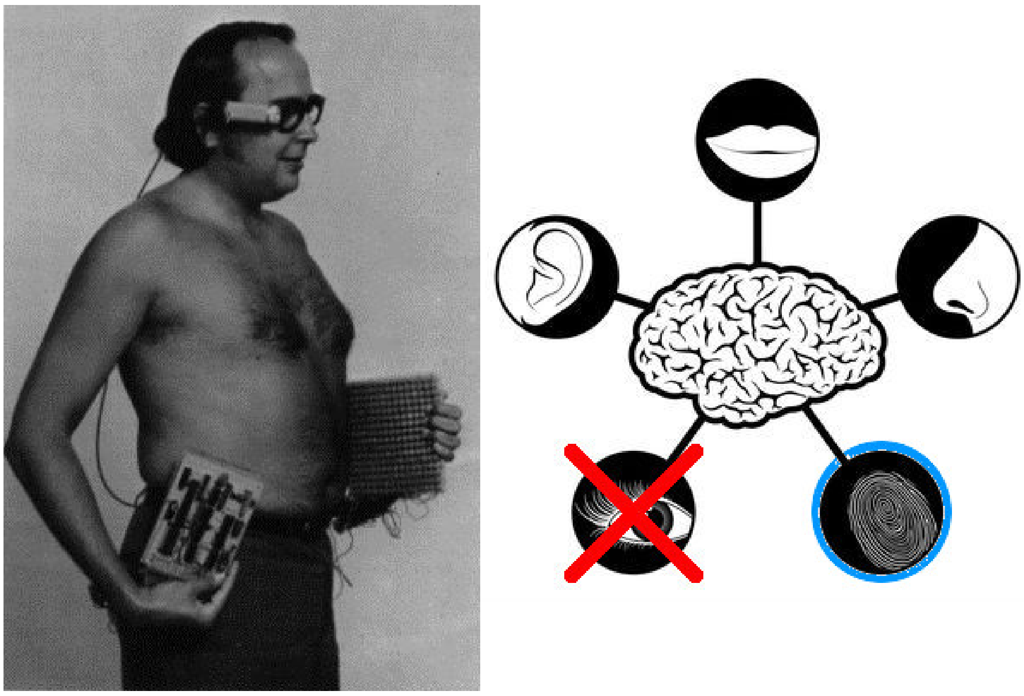


Figure 1: Paul Bach-y-Rita demonstrating a prototype of a device for substituting vision through the feeling of touch. The image is taken from [50].

wheel", since in many cases a major problem of one discipline is already solved in another one. An example of such a situation could serve the reinvention of numerical integration in the medical domain [42] 210 years after the works of Lagrange and Sir. Newton. Another consequence of the fragmented nature is that almost all of the research in sensory substitution can be grouped into categories based on the modality that is being substituted. A general approach, that covers all of the disciplines and does not discriminate between the particular modalities that are being substituted, is missing. Our goal is to provide a framework for sensory substitution that would address the issues of generalization and abstraction of modalities as well as automate the process of the substitution. Based on the definition of sensory substitution this

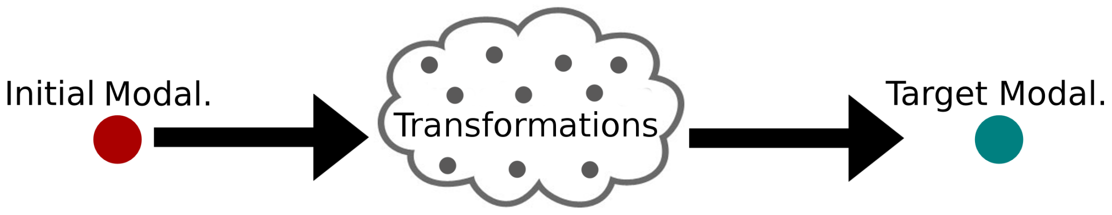


Figure 2: Visualization of the process of sensory substitution. Here the red and cyan dots depict the initial and target modalities respectively. The gray dots depict the individual transformations.

can be realized by solving the problem of finding the correct set of transformations between the initial and target modalities, Figure 2.

Several issues emerge in complex cases where multiple physical and mathematical transformations are required to get from one modality to another. Most of the time more than one possible set of transformations is present, which results in the dilemma of choosing the optimal one. A set of transformations might result in a link between two modalities with small error accumulations, but can have the drawback of having a large complexity, which would render it as useless for constructing a real time sensor. On the other hand the complexity might be fast enough for this purpose, but the error accumulation amount to a large value thus making it unfeasible to extract any useful information from the resulting data. Finally, several equivalent sets of transformations might be present, which depending on the conditions could provide more reliable results than the others. Thus, some sort of a decision making mechanism to always pick the best results is needed.

1.2 SENSORY SUBSTITUTION VS SENSOR DATA FUSION

Sensory Fusion or Sensory Data Fusion is a branch of signal processing that concentrates on the combination of sensor data or data derived from sensor data. The mentioned data is combined in such a way that the resulting data is in some way, shape, or form better than the original data. Note that as a result of its definition, sensor data fusion assumes that there is a way of comparing the initial sensor data to the resulting fused data. The term "better" can be interpreted as more precise, more complete or more reliable.

An example of data fusion can be the averaging of the position of the same object detected by three different sensors with known positions in reference to each other. Another example involves the stereo-reconstruction of an object using two cameras located at different viewpoints [30], [43]. In both of the cases we have sensors that are sensing in Cartesian space with some uncertainties depending on the sensing axis. Let us discuss the stereo reconstruction example in detail. Each camera can sense in 3D Cartesian space with a certain uncertainty. This means that the camera determines the position of a pixel as such $p = \{x + \delta x, y + \delta y, z + \delta z\}$. Where $\{x, y, z\}$ represent the true coordinates of the pixel and the $\{\delta x, \delta y, \delta z\}$ represent the uncertainties of the measurement. Note that since a regular camera is a 2D sensor, i.e. it can only determine rays pointing from its center in the direction of the object, the uncertainty $\delta z = \infty$ is always infinite. Normally the goal of stereoscopic reconstruction problems is the reduction of the uncertainty in z direction. Consider Figure 3, the light green areas in all three of the images represent the space that can be sensed by a single camera, the dark green areas represent the space that can be sensed with both cameras at the same time. Further, the pink areas represent the space where a particular pixel can be located when sensing with only one camera, and the red areas represent the space where a particular pixel can be located after the introduction of the second camera. In Figure 3 A) both cameras are placed parallel to each other. Note that such a placement results in the largest space where both cameras can sense. However, this placement does not really help in uncertainty reduction since the space where the pixel can be detected by both cameras is also infinite. In Figure 3 B) both cameras are placed perpendicular to each other. The resulting common space is small; however, the uncertainty of the pixels in this area is the smallest possible for the cameras used.

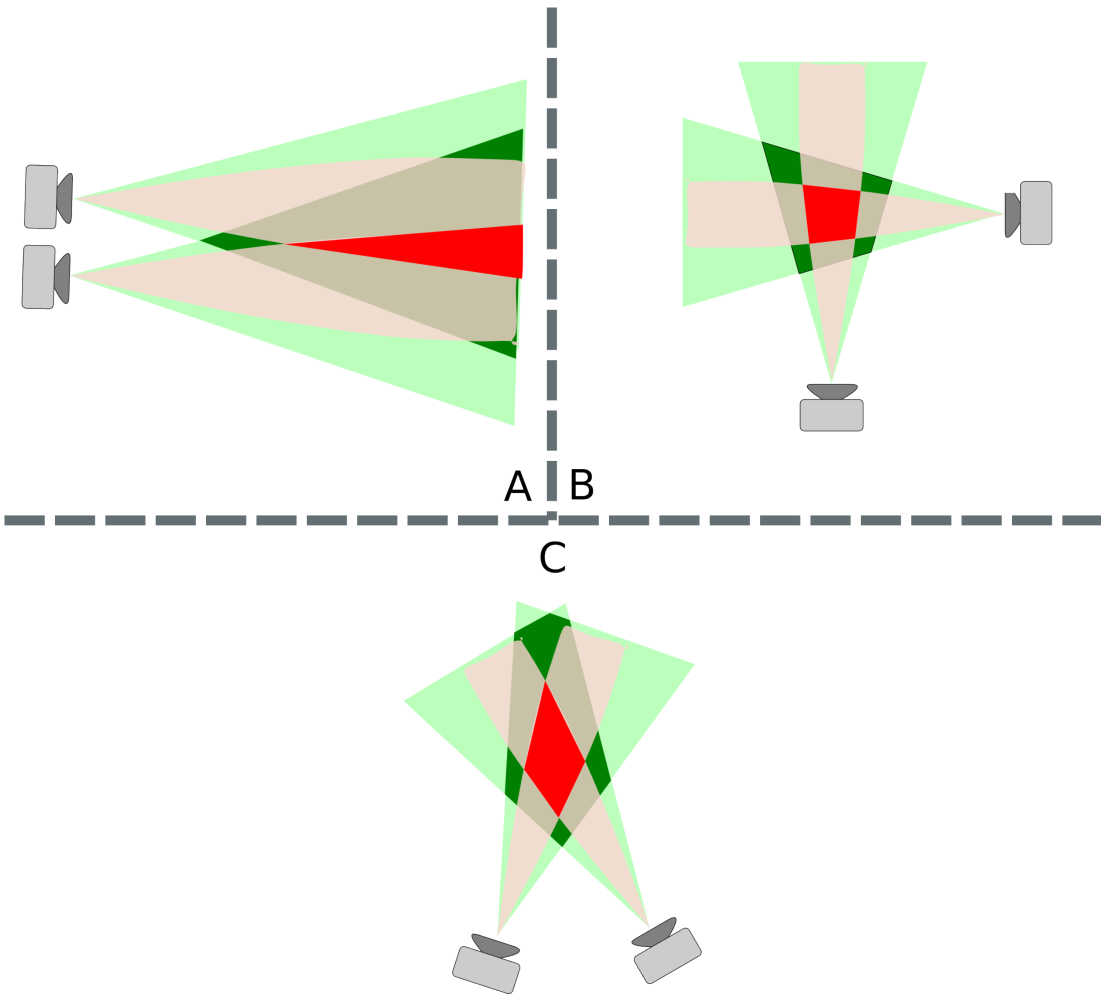


Figure 3: Challenges of sensory fusion illustrated on the example of stereoscopic reconstruction. In all of the images the light green areas represent the space that is visible to one camera, the dark green area represents the space that both cameras can see. The pink area represents the space where a particular pixel detected by the camera can be located. Note that regular cameras detect the z dimension (depth) of the pixel with infinite uncertainty. The red areas represent the intersection of the spaces from both cameras where the particular pixel can be located.

Finally, Figure 3 C) depicts a case where both cameras are placed at an angle in reference to each other. This case is a compromise between the two extremes depicted in A) and B). The resulting uncertainty reduction is in between the first two cases. Finding the optimal placement and configuration of the sensors as well as method for the maximal reduction of the uncertainties in the measurement is the main goal of sensor data fusion. Sensory fusion is not limited to the case described. It can be extended to fusion of data from a set of heterogeneous or homogeneous sensors, soft sensors, and history values of sensor data.

There are multiple mathematical frameworks that are used for sensor data fusion such as the Kalman filter [29], Bayesian networks [37], Dempster-Shafer [13] theory, etc. The Kalman filter performs sensor data fusion based on noisy historic values of the data acquired from the sensor itself. Results of the Kalman filter are more precise than the raw values. Bayesian networks perform data fusion on multiple similar sensors using the belief/probabilistic approach. Dempster-Shafer theory is similar to Bayesian in that it also uses probabilistic/belief based approaches for uncertainty reduction. The inherit nature of sensor data fusion requires that the resulting data be comparable to the original data. Therefore both the data before fusion and after must have the same modality, since comparison of data in different modalities such as meters to kilograms, is not well defined. Sensory Substitution on the other hand concentrates on the transformation of the data modalities from one to the other. Thus, both of the disciplines do not compete with each other but instead complement each other. The results of sensory substitution can be fed into a system dealing with sensor data fusion to increase the reliability of the overall system.

1.3 PREVIOUS WORK IN SENSORY SUBSTITUTION

OVERVIEW : Due to the fragmentations in the topic of sensory substitution it can be easily divided into categories. To retain structure we will initially categorize the subject based on the type of a substituted signal, that is haptic, visual, audio, etc. These will further be divided into subcategories based on the approaches they inherit to tackle the problem. The two largest parts of the mentioned research focus on haptic sensory substitution, aimed at the development of prosthetics, and visual sensory substitution, aimed at the compensation of visual impairment.

SUBSTITUTION OF HAPTIC SENSORY SIGNAL : Haptic sensory substitution can be further divided into subcategories: methods that inherit the electro-mechanical approach and those that adopt the vision based approach. Damian et. al. [12] present an artificial skin for prosthetic limbs, which is designed to help detect slippage. It uses the electro-mechanical approach to compute the 2 dimensional friction forces occurring over its surface. Another example of electro-mechanical force sensing is described in [8]. The authors present a force sensor for teleoperation of remote limbs. The former contains a number of hollow cylinders of different heights that are enclosed in one another eventually forming a conical pyramid. Based on the number of shifted cylinders and the magnitude of the shifts, the authors are able to estimate the perpendicular force vector and the approximate area of the contact. Both of the previous methods only succeed in extracting partial information regarding the contact and forces acting on it. The latter is a result of solving the problem only from the electro-mechanical point of view, which has an advantage of providing reliable readings of data but struggles to operate in higher dimensions and resolutions.

Other disciplines specializing in haptic sensory substitution are concentrating on vision based approaches. In their paper [10] Chorley et. al. present a biologically inspired tactile sensor. The latter consists of a CCD camera which is mounted on the flat side of a half spherical clear silicon, and nodule markers that are mounted inside the spherical surface of the clear silicon. Here the authors propose that the force and contact area shape information be estimated based on the spacial shifts between nodule markers. However, they fail to provide any mathematical, physical or algorithmic suggestions as to how. A slightly different and more successful approach is described in [23], [22], [21]. The authors present a sensitive fingertip sensor that consists of a CCD camera mounted on one side of a clear silicon and two rows of colored spherical markers that are located within the clear silicon. Based on the observed shifts between the two rows of the markers, the authors present an algorithm for three dimensional force distribution reconstruction. The presented force sensor only performs well in estimating force distribution if the magnitudes of the applied forces are relatively high. However, it struggles in dealing with smaller forces, and providing fine detail. The reason behind this is that the clear silicon acts as a low pass filter.

Both of the described disciplines are able to solve only one or two discipline specific aspects needed for a full haptic sensory substitution. The need for a framework that will allow the combination of different aspects into one is apparent.

SUBSTITUTION OF VISUAL SENSORY SIGNAL : Most of the research on substitution of visual sensory signal is aimed at developing devices for the visually impaired. Here too, there are two major disciplines trying to solve the issue: computer vision and neuroscience. However, as in the case for substitution of haptic sensory signal no global solution can be achieved by only considering one side of the equation. Johnson et. al. [31] describe a device for improving the navigation of visually impaired people. The device consists of a stereo pair, a processing unit and a tactor belt. It operates by converting the visual information into vibrations, which are performed by the belt. The authors were successful at obstacle avoidance in sparse environments.

However, the approach failed to provide reliable results in dense environments. Even though a correct setup was used in the scopes of computer vision, that is the stereo setup, this approach struggles with human-machine interface. According to [28],[16], [27],[25],[26] and [24] the human tongue provides a better human-machine interface due to its high sensitivity, which is the direct result of its sensory receptors being closer to the surface.

In [51] another device is described for vision based visual sensory substitution in the context of neuroscience. The authors use a camera mounted on the head of the human user. The latter is connected to a processing unit which converts the images into 144 low-voltage impulses that are sent into the mouth of the user by means of a ribbon cable. As a result, the human operator was able to catch a rolling ball solely based on the input from this device. The authors justify the usage of only one camera by *"Sensory Overload"*. However, one can argue that the introduction of the second camera in combination with stereo reconstruction and object recognition could significantly reduce the amount of information provided to the tongue of the user.

BIOLOGY AND NATURE : Many papers in biology discuss the mechanics behind the sensory substitution for humans and animals who have lost their vision or never had it. These mechanics can serve as a strong motivation for sensory substitution in other disciplines. In his paper, Rauschecker [20] performs experiments on auditory localization with cats that were blind since birth and sighted cats. The results have shown that as a compensation to visual impairment, the blind cats were better at auditory navigation. In his work Windsor [41] describes the navigation mechanism used by the Blind Mexican cave fish (Astyanax fasciatus). Here the fish uses reflections of waves caused by its motions to successfully avoid obstacles and form an idea regarding the surrounding terrain.

Other cases of sensory substitution in nature can also serve as a strong basis for sensor development and substitution in man made devices. In [36] the authors describe the logistics behind auditory localization performed by snakes. Snakes lack a tympanic membrane and the external ear openings, but are equipped with a perfectly functioning inner ear. They can locate the prey based on the vibrations of the sand by placing their jaw on it, which allows the transfer of vibrations to the inner ear. Farnosch [19] et. al. describe in their work the model by which a frog detects its prey. Using many of its lateral organs it can not only determine the direction and the nature of the motion occurring in the distance but can also distinguish between two different sources of motions. In [7] the authors describe the mechanics by which the snakes equipped with a poor infrared sensor determine their prey.

1.4 CONTRIBUTIONS

UNIFICATION OF INTERFACES BETWEEN DIFFERENT DISCIPLINES : Although current cognitive neuroscience research treats Sensory Substitution as visual input substitution by either acoustic or tactile modalities, the goal of this thesis is to provide a framework that generalizes this concept to any sensing modality. While there are currently many approaches that specialize on substitution of individual classes of sensory signals, a general method that unifies all the types into one framework is missing. An introduction of such a framework will contribute to the reduction in fragmentation of sensory substitution. A direct result of such unification will lead to closing the knowledge gaps in individual disciplines by providing the interface and transformations between them. Our framework helps to avoid situations such as in the case described above for Visual sensory signal substitution, where computer vision specialists were able to succeed in executing the computer vision part, but struggled in human machine interface, and vice versa.

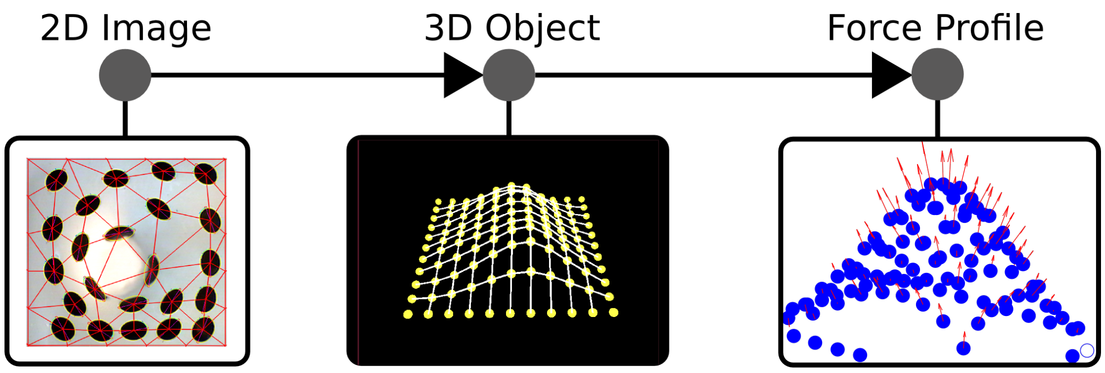


Figure 4: An example of a virtual force sensor constructed using our framework. The latter is realized by an optical camera, which registers the deformations of a plastic membrane.

COMPUTATION OF THE OPTIMAL CHAIN OF TRANSFORMATIONS BETWEEN THE INPUT AND OUTPUT MODALITIES : Construction of virtual sensors from a set of initial physical sensors is an integral part of our framework. It estimates the optimal chain of transformations from the sensing domain of the original sensor(s) to the desired sensing modality using a set of physical laws and mathematical operators. The process of chain construction can be tuned to optimize for either minimum error, minimum complexity, or to strike a balance between minimum error and complexity. It takes into account not only the modality(ies) that the initial sensor(s) operate(s) in but also its (their) operating range and expected errors. The chain of transformations is established by applying the extended weighted Dijkstra's [14] search algorithm on a connected graph, the nodes of which are the available transformations. The latter ensures that the constructed chain is the optimal one. In situations where the initial operating range of the original sensors changes, our framework is capable of dynamically reconfiguring the graph to the current conditions.

An example of this is illustrated in Figure 4 where a virtual force sensor is created by means of an optical camera observing the deformations of an elastic membrane. This example will be discussed in detail in Chapter 8.

DYNAMIC OPTIMIZATION BETWEEN MULTIPLE TRANSFORMATION CHAINS FOR MULTI-SENSOR PLATFORMS : Recently there is a trend of equipping various mobile platforms with a number of sensors that operate in different modalities. Modern vehicles are equipped with LiDAR, stereo cameras, ultrasound systems, accelerometers, GPS, rotary encoders, etc. Similarly smart-phones are fitted with a monocular camera, gyroscope, GPS, accelerometer, microphone and much more. Our approach allows the fusion of sensory data from different modalities. It not only increases the reliability of the acquired data by adding redundancy to the system, but also widens the spectrum of sensible modalities using virtual sensors. It allows a dynamic readjustment of the chain of transformations between the initial and target modalities based on the current values of the perceived arguments.

1.5 STRUCTURE OF THE THESIS

The content of this thesis is divided into two parts. Part i concentrates on describing the theoretical background and the framework itself. Part ii is the experimental part, where different aspects of Part i are evaluated.

PART I : Begins with the overview of the framework in Chapter 2, where a brief introduction to the main steps of the framework as well as its operational modes and optimization parameters are presented. This is followed by Chapter 3, where the reader is presented with a detailed description of each individual aspect of the framework, which involves the construction of the transformation graph with a strong focus on its individual parts, the different operational modes, and the optimization parameters. Finally Chapter 4 begins by describing the process of estimation of the optimal chain followed by the effects that the optimization parameters have on the computation of the chain, and concludes with the illustration of differences of the operation modes.

PART II : Begins with an example designed to illustrate how the error estimates obtained using our framework compare to real world errors in Chapter 6. We demonstrate how our framework operates in a multi-sensor environment. Where it has a task to increase the reliability of a particular sensor reading in Chapter 7. We conclude this part by showing an example case for constructing a new sensor using the framework in Chapter 8.

We provide conclusions and talk about the future work in Chapter 9

Part I

THE FRAMEWORK

2 OVERVIEW OF FRAMEWORKS

2.1 OVERVIEW OF FRAMEWORKS IN COMPUTER SCIENCE

In computer science there are multiple ways of implementing a framework. These involve data-flow networks such as the ones described in [32] and [35] or functional libraries such as [47] and [48] as well as object-oriented frameworks such as [39]. Each of these come with positive and negative attributes depending on the problem being solved. In this section we will briefly describe each of them, talk about their strengths and weaknesses as well as bring some examples of situations where they are successfully used.

DATA-FLOW FRAMEWORKS : Such frameworks are used for modeling systems that deal with transmitting information between different static points. They are generally capable of retaining state and are good in constructing automated systems. The main goal of such systems is finding the optimal set from a large cluster of choices as well as modeling the flow of information through pivot processing units. These systems tend to concentrate on the global picture of the problem, thus they are not as flexible for low level adjustments and are very application specific. Examples of such nets involve Petri nets [38] and Neural networks [33]. Petri nets are largely used for modeling distributed systems. They were initially implemented for describing chemical processes. The integrated binary logic in Petri networks makes them a perfect candidate for describing a network of linked transformations. However, they lack in mechanisms for automatic network construction and classification of individual subnetworks. Neural networks are widely used in machine learning and are a family of statistical learning algorithms. Such networks are great for the classification of the data quality. However, they do not provide any tools for modeling transformations and interfaces between them.

FUNCTIONAL LIBRARIES : Such frameworks are usually abstract sets of functions that are implemented to perform small mathematical or logical tasks optimally. They allow low level adjustments, are highly configurable and reusable. Functional libraries are widely used as additions and extensions to existing programming languages. Examples of functional libraries include pymunk [49] and XVision [46]. Pymunk is a physics library for python. It is ideal for modeling simple physical transformations. However, it lacks mechanics for defining transitions from one transformation to another, or for dealing with multiple networks of transformations. XVision is a well known C library for computer vision. It is ideally suited for performing image processing tasks, and thus can serve well in the implementation of related transformations. In both cases functional libraries are simply not suited for defining interfaces between different transformations as well as evaluating the different solutions of the problem or automatically assembling sets of candidate solutions.

OBJECT-ORIENTED FRAMEWORKS : Such frameworks are ideal for modeling different real-world or abstract objects. They retain state and define interfaces of communication with the object in question. Examples of such frameworks involve the Open Dynamics Engine (ODE) [45] or Eigen [44]. ODE is a physics engine that models an open world with well defined physics and rigid objects located within it. As one would expect, the interactions between the objects are well defined and optimally implemented. However, due to its strict interface and functionality definition, it is practically not possible to re-purpose the engine to any other task that does not involve rigid body dynamics. The same could be said about Eigen, which is an object-oriented framework designed to optimally implement a 2 dimensional matrix with all the corresponding operations and interfaces.

REQUIRED FUNCTIONALITY FOR THE FRAMEWORK : As mentioned in Chapter 1.1 the main goal of our framework is to establish an optimal chain of physical and mathematical transformations between the initial sensing modalities of the available sensors and desired target modalities. Note that the usage of the word "sensor" does not discriminate between human sensory organs (eyes, skin, etc) and digital sensors. There is a number of issues that need to be solved and defined by the framework to be able to solve the problem. Since most of the time more than one transformation is required to get from initial to target modalities, transformations need to have a well defined and standardized interfaces between each other. The latter will allow them to communicate information between each other. Often more than one way of achieving a solution exists. The framework must define a metric for classifying each of the ways of obtaining a solution. It should also provide a way of configuring the classification criteria. Further the framework should allow the flexibility of reclassification of the solutions, in case some of the configuration parameters change. All of this can be achieved by using a data-flow framework based on such mathematical structures as graphs [17]. By representing the transformations as graph nodes, the main problem of finding a set of transformation between the initial and target modalities can be reduced to graph construction and traversal, Figure 5.

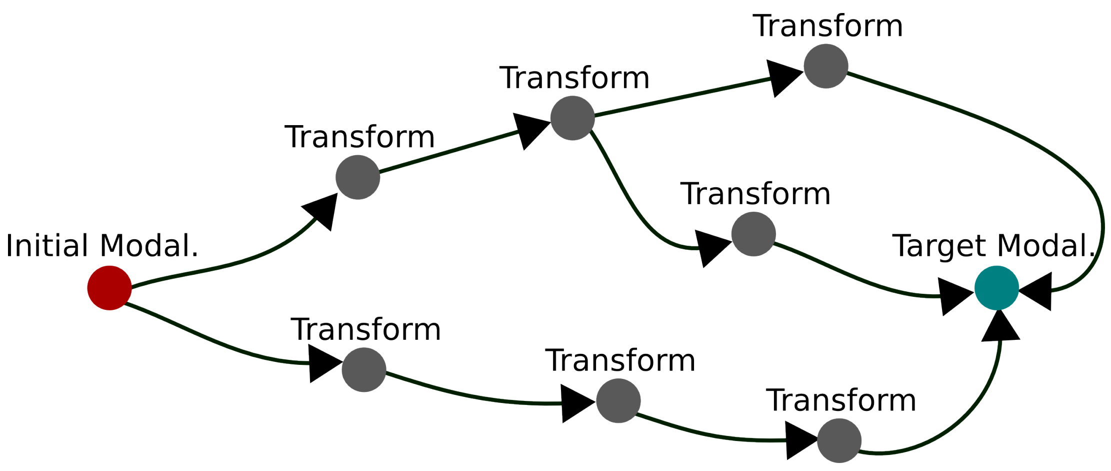


Figure 5: Visualization of the graph based sensory substitution. The transformation nodes are depicted in gray. Initial and target modalities are depicted in red and cyan respectively.

Each traversal sequence (path) that results in getting from the initial node to the target node represents a solution to the problem of sensory substitution. Further the classification of the solutions can be achieved by the computation of the path distances, Figure 6. Paths that have a different path distance are colored in individual colors. The red and cyan paths share common segments which are the first two segments of the cyan path.

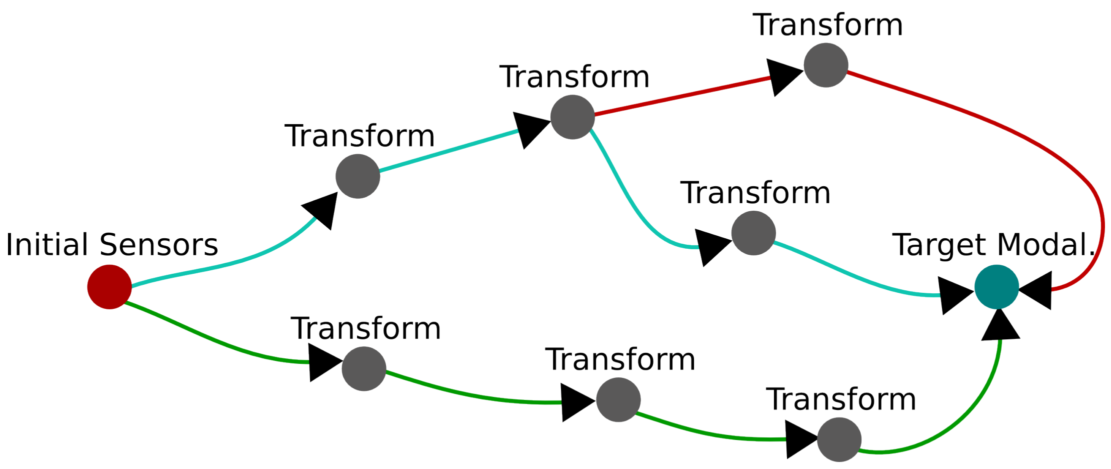


Figure 6: Visualization of the classified paths in the graph. Each color represents a single path that was given a certain classification of the traversal quality.

2.2 OVERVIEW OF THE FRAMEWORK FOR SENSORY SUBSTITUTION

In this section we provide a short introduction to the framework for sensory substitution. It is represented as a connected graph, which includes the afore mentioned physical and mathematical transformations as its nodes. The transformation nodes are provided externally as a *bag of transformations* where each individual element is a transformation from one modality to another: e.g. Newton's second law, Hook's law, etc. The *bag of transformations* can be obtained from the knowledge base of the considered area. Further the graph contains some special nodes i.e. *Start* and *Goal*, that represent the modalities of the initial sensors and the desired sensing modalities accordingly. We call this graph a *transformation graph*. The links between the nodes of the graph are represented by edges that are equipped with weights.

The framework is flexible and can be calibrated to optimize for such parameters as errors, complexity or both of them combined. This is done by manipulating the reasoning process for weight assignment.

OPERATION MODES : Our framework can operate in two modes: dynamic or static. The mode is determined based on the problem that needs to be solved. Dynamic mode is used for systems that are already constructed and all of the initial sensors are set. In this mode for every iteration our framework uses a chain of transformations to actively establish a link between the modalities in which the initial sensors operate and the target modalities. Static mode is used for planning a construction of new sensors. This mode aims to estimate a static optimal link between the available and desired sensing modalities which is established only once. The main difference between the two modes is in the computation of weights based on which the optimal transformations chain is chosen. In static mode the weights are statically set and are based upon the operating ranges with corresponding expected values of the initial sensors. On the other hand in dynamic mode all the weights are recomputed during each iteration based on the current value of the argument that is being processed.

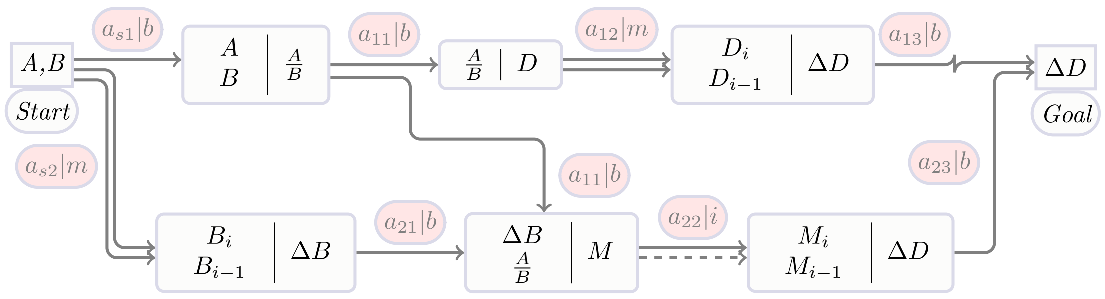


Figure 7: Transformation Graph: The edge weights are computed based on the inputs, complexity and errors of the corresponding nodes. An optimal chain is obtained by using extended Dijkstra's search algorithm.

CONSTRUCTION OF THE CONNECTED GRAPH : The process of constructing the transformation graph can be split into three logical sets of steps shown in Figure 8. The first set of steps is *the global decisions* where the problem is defined. One needs to isolate the set of initial sensors that would be used and determine the target modality that needs to be sensed. Further, one must determine the working mode of the framework and define the optimization parameters. Once the global decisions are made, the second set of steps *the construction of the graph* can be executed. Firstly, the *Start* and *Goal* nodes need to be constructed based on the initial set of sensors and the target modality. Further, those nodes are connected to the relevant transformation nodes from the *bag of transformations*, which results in a connected graph. Afterwards the transformation nodes of the graph must be implemented and the error profiles for each transformation must be computed. Finally, *the construction of the graph* is completed by the computation of edge weights, based on the optimization parameters. This is followed by the third set of steps *computation of the optimal chain*. Where the link between the initial and target modalities is established based on the operation mode, using an extended weighted Dijkstra's search algorithm (Figure 7).

The entire process of construction of the *transformation graph* can be summarized in the following steps:

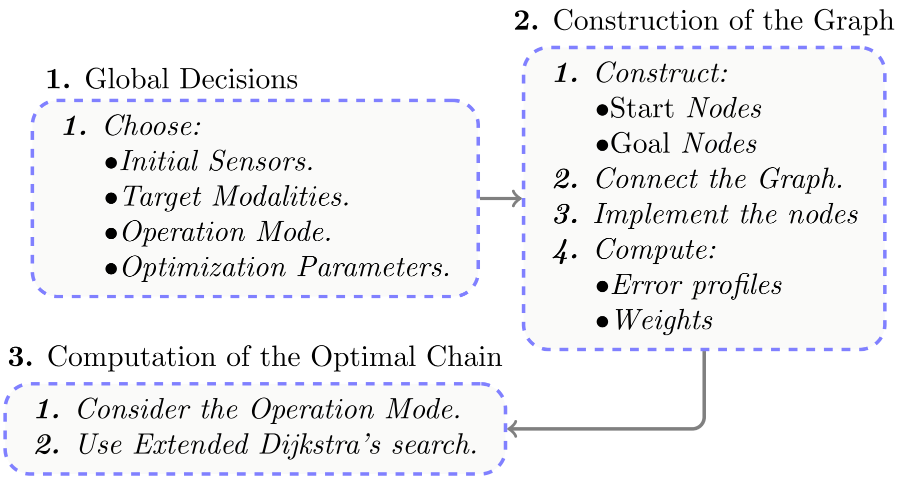


Figure 8: Sets of steps for the construction of the graph

3 CONNECTED GRAPH

3.1 CHOICE OF THE MODES AND OPTIMIZATION PARAMETERS

OPTIMIZATION FOR ERROR MINIMIZATION : The first step in the creation of the transformation graph is the choosing of the optimization parameters. Depending on the problem there are several available options. The first one is the optimization for minimum errors. This means that the weights of the connected graph will only reflect the occurring errors in them. Further, the path computation will only concentrate on the error propagation.

OPTIMIZATION FOR COMPLEXITY MINIMIZATION : The second one is the optimization for minimum complexity. This means that the main focus of the weights of the *transformation graph* will be on reflecting the complexity of each individual transformation block. This is done by taking into account the amount of data that is needed by the transformation block to perform. This is followed by a path computation process, during which the complexity of each transformation block will be considered. Hence the link between the *Start* and *Goal* with the least integral complexity will be estimated.

OPTIMIZATION FOR BOTH : The third one is the optimization for both minimum error and complexity. This means that the weights of the graph will reflect both of the parameters. The final optimal chain of transformations will reflect the best compromise between the minimum error and complexity.

3.1.1 *Dynamic Mode*

Next is the selection of the operation mode based on the problem. There are two possible scenarios. The first one is using the dynamic mode. In this mode the graph contains all the possible paths for the current configuration. During each iteration the optimal path is computed based on the current values of the sensor data. This ensures the reliability of the final result independent of the operating range. The drawback of this mode is the computational time. An example scenario where the dynamic mode can be useful is the following: one has an already built multi-sensor

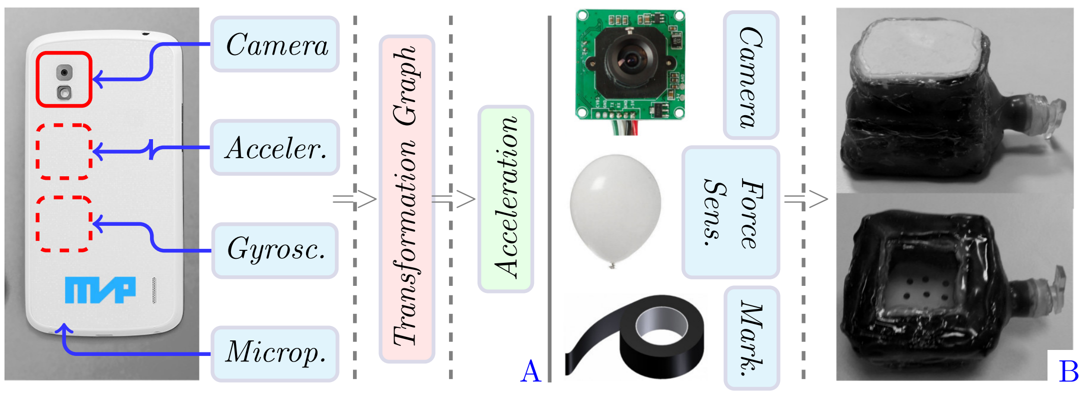


Figure 9: (A) An example case for usage of the dynamic mode. The task is to increase the reliability of acceleration readings of an already constructed multi-sensor platform. (B) An example case for usage of the Static mode. The task is to construct a 3D surface force sensor, using the set of available initial sensors.

platform that needs to be enhanced. This means that either an extra modality needs to be sensed that is not accessible through the sensors directly, or the modality can be sensed through one or more of the sensors; however, the values are unreliable. Such an example can be sensing accelerations using a modern cellphone that is equipped with a camera and an accelerometer, Figure 9(A). The latter example will be discussed in great detail in Chapter 7.

3.1.2 *Static Mode*

The second scenario is using the static mode. In this mode the graph is initially constructed with all the possible connections using all of the initial sensors and all of the relevant transformations. Further, the operating ranges of the sensors are considered and the optimal chain of transformations is estimated based on the expected values of the sensors. Note, the optimal chain is constructed only once in the beginning. This ensures the low speed of consecutive measurements. The drawback of this mode is that the system is calibrated for expected values, which means that the results are not always optimal. An example scenario where a static mode can be of use is the following. One needs to build a new sensor from scratch. Often there are some modalities that are not easily accessible directly through a single standard sensor. To reach them, one needs to perform some physical and mathematical transformations on the data from the initial standard sensors. Such a case can be a 3D surface force sensor construction using a camera, an elastic membrane and markers, Figure 9(B). The latter example will be discussed in great detail in Chapter 8.

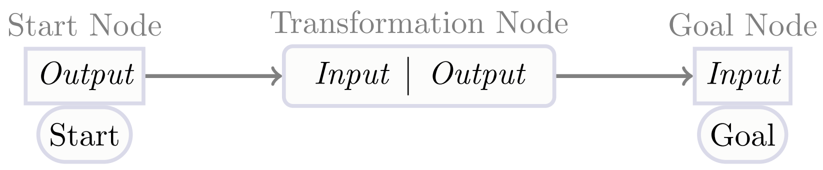


Figure 10: Illustration of all the three types of nodes, that are a part of a *transformation graph* . First node on the left is a *Start* node, those have only an output section, thus edges can only point out of them. Second node in the middle is a regular transformation node. Those nodes have both input and output sections, therefore edges can both point into them and out of them. Third node on the right is a *Goal* node. Those nodes have only an input section, hence edges point only into them.

3.2 CREATION OF THE CONNECTED GRAPH

As mentioned before, we define the available mathematical and physical transformations as nodes of our *transformation graph* . We call those nodes " *transformation blocks* ". Each *transformation block* contains two sections. The first section describes the input arguments required by the transformation. The second section describes the output values, i.e. the arguments that are a direct result of the transformation. An example of a transformation block is illustrated in Figure 10. Since the aim of our framework is to establish the most optimal set of transformations that link the modalities of the initial sensors to the target modality, we define each *transformation block* in such a way that the modalities of the input block are different than the modalities of the output block. This means that a series of mathematical and physical equations can be combined into a single *transformation block* only when the direct result of their application is a modality change of the initial set of arguments. The *transformation graph* is created by connecting the corresponding nodes to each other. The correspondence between the nodes is defined based on the matching input and output pairs, i.e. if the node *N0* has an output "*A*", node *N1* has an output "*B*" and the node *N2* requires "*A,B*" as an input. The three nodes will be connected by two edges. The first, pointing from the output of the node *N1* to the input of the *N2*, and the second one pointing from the output of the node *N0* to the input of the node *N2*, Figure 11.

The resulting connected graph is completed by the insertion of special *Start* and *Goal* nodes. Insertion of the *Start* and *Goal* nodes into the graph is performed similar to the connection process of regular nodes. The difference between the aforementioned special nodes and transformation blocks is that the *Start* nodes possess only the output section and the *Goal* nodes possess only the input section. This results in *Start* nodes only having edges that are pointing out of them and *Goal* nodes only having edges that are pointing into them.

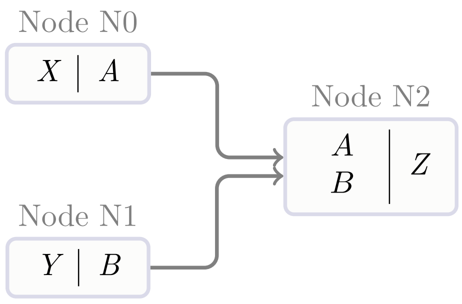


Figure 11: Visualization of the edge insertion process. The edges are inserted between corresponding input and output pairs.

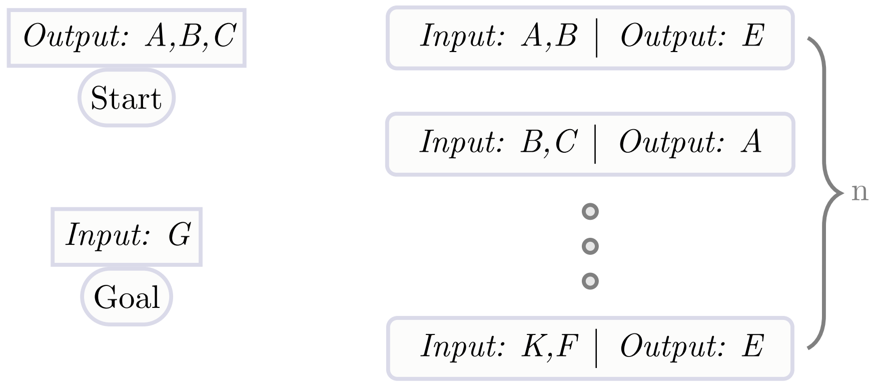
3.2.1 *Connection of Nodes*

There are multiple ways of finding and connecting the corresponding nodes of the *transformation graph*. In this section we will discuss several approaches and suggest what in our opinion is the best way to solve this problem. Consider Figure 12 for visual support.

NAIVE ALGORITHM : A naive approach for connecting the nodes would be to compare the output modalities of each node to the input modalities of each node and insert an edge in between if they match (Algorithm. 1).

Algorithm 1 A naive algorithm for connecting the *transformation graph*. The algorithm requires as an input the unconnected nodes. The latter will be referred to as: Graph

```
1:  for all $V_o \in$ Graph.nodes do
2:    for all $M_o \in V_o.\mathrm{Output.Modalities}$ do
3:      for all $V_t \in$ Graph.nodes do
4:        if $V_t ! = V_o$ then
5:          for all $M_t \in V_t.\mathrm{Input.Modalities}$ do
6:            if $M_o == M_t$ then
7:              Graph.insertEdge($V_o.\mathrm{getIndex}(), V_t.\mathrm{getIndex}()$);
8:            end if
9:          end for
10:       end if
11:     end for
12:   end for
13: end for
14: return Graph
```



Modality	Hash	Address	Input	Output
A	$f(A)$	1	1;5;7;..	2;6;4;..
B	$f(B)$	2	1;2;7;..	3;7;5;..
		⋮		
E	$f(E)$	p	4;7;	1;n;

Figure 12: An illustration of the graph connection process. The nodes of the graph are depicted on the right side. During the first loop over the nodes the indices of the modalities are stored in a 1 dimensional array that contains two vectors: Input and Output. The address of each modality is computed using a hash function after which the index is either pushed into the Input or Output vector depending on the location of the considered modality. Further, a loop is performed over the array, and edges are inserted into the graph based on the indices of Input and Output vectors (Algorithm 2).

The provided naive algorithm will result in a graph that has all the corresponding nodes connected. The positive side of this algorithm is the simplicity of implementation. The drawback of the algorithm is its complexity which is $(n^2\ p^2)$, where n is the number of nodes and p is the number of modalities. On the other hand the algorithm can be easily parallelized and the complexity can be reduced to $(n\ p)$.

HASH BASED ALGORITHM : A more complex approach for achieving a connected graph would be by using a hash function of the modalities. For each modality two vectors containing the indices of the nodes are needed, where the modality is present as an input or output. A single loop through all of the nodes will populate the input and output vectors for each modality. An additional loop through input and output vectors would be enough to insert all the edges. This approach is described in Algorithm 2.

The drawback of this algorithm is the more complex implementation than in the case of Algorithm 1; however, the complexity is significantly lower $(2np)$. The latter is comparable to the complexity of Algorithm 1 after parallelization. Thus, a preferred way of connecting the graph would be using Algorithm 2.

3.3 DESCRIPTION OF NODES

The *transformation graph* is constructed using different types of nodes and different types of edges. In this Section we will provide a detailed overview of all of them. We distinguish between three types of nodes: transformation blocks, *Start* nodes, *Goal* nodes, and three types of edges: basic edges, iteration dependent edges and high-order edges. The type of an edge is determined based on the class of the input block it connects to, and defines the weights that would be attached to it. Each node of the transformation graph contains information regarding its error profile. The error profiles correlate with the errors that either occur during sensing or because of the transformation to the current values of the arguments that are being processed. The profiles can be provided either in form of a function or of a look-up table.

3.3.1 *Transformation Blocks*

Transformation blocks represent a series of mathematical and physical transformations that result in a modality change between the initial and resulting arguments. An example of a simple transformation block can be an implementation of Newton's second law, where the modality N is obtained from the initial kg and m/s^2 modalities. As was mentioned before, each transformation block consists of two sections: input

Algorithm 2 Hash function based algorithm for connecting the *transformation graph*. The algorithm requires as an input the unconnected nodes. The latter will be referred to as: Graph

```
struct Mod                                        # Container for indexes
 vector<int> input;
 vector<int> output;
end struct

int nMod;                                         # Amount of modalities
Hash hash;                                        # Hash function
nMod = Graph.getNumberOfModalities();
Mod mod[nMod];                                    # Container for each modality
hash.init(nMod);
for all V ∈ Graph.Nodes do
   for all $M_o$ ∈ $V.Output.Modalities$ do
      int ind1 = hash.cmpIndex($M_o.Name$);
      int ind2 = $V.getIndex()$;
      mod[ind1].input.push_back(ind2);
   end for
   for all $M_i$ ∈ $V.Input.Modalities$ do
      int ind1 = hash.cmpIndex($M_i.Name$);
      int ind2 = $V.getIndex()$;
      mod[ind1].output.push_back(ind2);
   end for
end for
for all $ind \in [0..nMod]$ do
   int isz = mod[ind].input.size();
   int osz = mod[ind].output.size();
   for all $o \in [0..osz]$ do
      for all $i \in [0..isz]$ do
         int e1 = mod[ind].output(o);
         int e2 = mod[ind].input(i);
         Graph.insertEdge(e1,e2);
      end for
   end for
end for
return Graph;
```

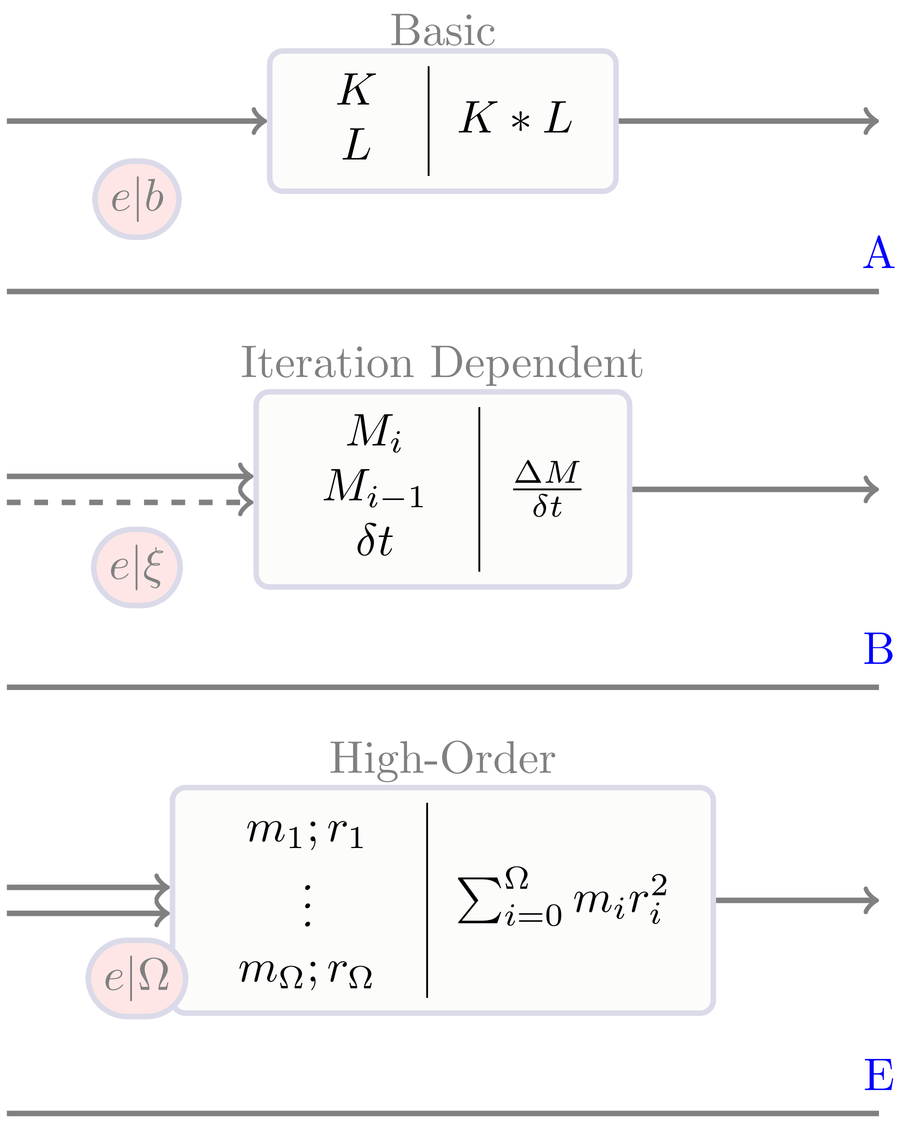


Figure 13: Different classes of transformation blocks. (A) A transformation with a basic input block. Those are linear transformations that require only one value as its input argument. Here b is the multiplicative weight that is normally set to one for these blocks. (B) A transformation with an iteration dependent input block. Those represent the first class of non linear transformations that require sequential values of the input argument from the previous and current iterations. Here ξ is the multiplicative of the weight that is higher than one due to the non linear nature of the transformation. (C) Transformations with high-order input blocks. Those require multiple consecutive values of the input argument, and do not retain state. Here Ω is the multiplicative weight, usually higher than two. In all of the cases above e is the additive weight. All of the weights depend on the optimization parameters of the problem and will be discussed in Section 3.5.

and output. The output section determines the nodes to which the current node will be connected too. The role of the input section is more complex. It determines where the connections can be received from as well as the amount and the re-usability of the values of input arguments. Based on the type of input arguments that the transformation requires, we differentiate between three classes of input blocks: basic, iteration dependent and high-order. Each of these classes is aimed at modeling a specific type of mathematical or physical transformations. In addition, each transformation block is equipped with an error profile. The error profiles aim to model the errors that occur due to the transformation and are used heavily when the goal of the framework is to optimize for minimal error accumulation.

BASIC INPUT BLOCKS : Basic input blocks belong to transformations that require only one unique value for each input argument. An example of a transformation with such an input block can be a simple multiplication by a scalar. Figure 13 (**A**) illustrates a case with a basic input block. Let node *N1* require *K* and **L** arguments as input, where *K* is provided from the output of node *N2* and **L** is a constant that is provided externally. Then, *N1* will only need to receive one value of *K* from *N2* and one value of **L** externally to perform the transformation $K * L$. Transformations with a basic input block can be used to model Newton's second law, to compute force from acceleration.

ITERATION DEPENDENT BLOCKS : These input blocks belong to transformations that, along side with the current value of the argument, require their value from previous iterations. An example of a transformation that has an iteration dependent input could be a computation of a derivative, e.g. to determine velocity from displacement. These transformations retain state. Let *N1* be a node that computes derivatives and requires M_i, M_{i-1} and external constant $\delta\mathbf{t}$ as input arguments and *N2* be a node that provides M_i as output. During the first iteration *N2* will provide *N1* with M_1. However, since *N1* requires two arguments to perform the transformation, it will not proceed during this iteration and will be halted. Nonetheless, it will store the value of M_1 to be used in the next iteration. In the second iteration *N2* will provide *N1* with M_2, this time *N1* will have all the required input arguments and will perform the transformation $M_2 - M_1$. After the transformation is performed *N1* will store the value of M_2 for further use in the next iteration. Further operation of the *N1* node will be similar to the second iteration. An example of an iteration dependent block is illustrated in Figure 13 (**B**).

HIGH-ORDER INPUT BLOCKS : The third class of input blocks belongs to transformations which require multiple consecutive values of their input argument and do not retain state, see Figure 13 (**C**). A mathematical example of a transformation requiring a high-order input block is numerical integration. Such a transformation block could be an implementation of discrete computation of the moment of inertia. The order of the block is determined by the amount of required values. Let *N1* be a second order high-order node that requires $[m_1; r_1], [m_2; r_2]$ pairs of arguments to preform, and *N2* provides $[m_i; r_i]$ as output. During the first iteration *N2* will provide the arguments $[m_1, r_1]$ to *N1*. Since the latter needs two values to operate, it will store the value but will not perform the transformation. During the second iteration *N2* will send $[m_2; r_2]$ to *N1*. Now that both of the needed arguments are provided, *N1* will perform the transformation $m_1 r_1^2 + m_2 r_2^2$; however, after the transformation is performed *N1* will delete both of the values. The following two iterations would be performed similarly to the first and second iterations.

The difference between the high-order and the iteration dependent input blocks is that the latter store the current value of the argument for use in the next iterations, whereas the high-order input blocks do not maintain any state.

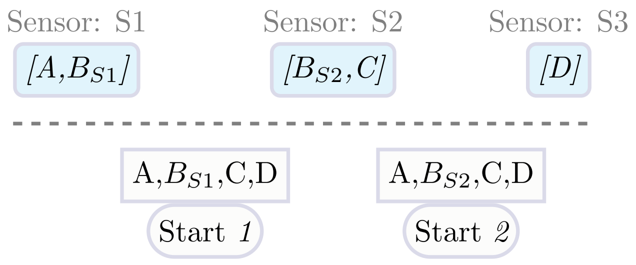


Figure 14: An example of a *Start* node creation, from a given set of initial sensors.

3.3.2 **Start *Nodes***

Start nodes are special nodes that describe all of the available input sensors. In contrast to transformation nodes, these consist only from one section, i.e. output. As a result all the edges of *Start* nodes are pointing out of them. These nodes are constructed using the operation modalities of the initial sensors. In cases when all of the initial sensors have non-repeating operation modalities, they are all compiled into an output block of a single *Start* node. In cases where some of the initial sensors have the same sensing modality, more than one *Start* node must be created. The amount of *Start* nodes needed is equal to the amount of the repeating modalities of the initial sensors. Consider Figure 14. Let *S1* be a sensor that can sense in [*A*,*B*] modalities, sensor *S2* in [*B*,*C*] modalities and sensor *S3* in [*D*]. In this case all of the sensing modalities of the initial sensors are unique except *B*, which can be sensed by two separate sensors. For simplicity we will mark values of *B* sensed from *S1* by B_{S1} and from *S2* by B_{S2}. Because of the two repeating modalities, two *Start* nodes will be created. The first with [*A*,B_{S1},*C*,*D*] and the second one with [*A*,B_{S2},*C*,*D*] output parameters. In case the framework is required to optimize for minimum error for each *Start* node, a separate connected graph is needed. Note that even though each start node is assigned to a separate *transformation graph*, they are all identical. The only difference between the separate graphs is the weights assigned to the edges, since they depend on the accuracy of the sensors. In cases where the framework needs to optimize only for complexity, a single graph with a single *Start* node can be used.

3.3.3 **Goal** *Nodes*

Goal nodes describe the desired modalities that need to be sensed. They are similar to *Start* nodes in the sense that they also have only one section, but in contrast to *Start* nodes that section is the input section. The arguments of the input section are set to the modalities that are needed to be sensed. As a result of having only an input section, *Goal* nodes have only edges that go into them.

3.3.4 *Error Profiles*

Error profiles are an integral part of each transformation block. They represent the error that occurs due to the transformation of the current values of the arguments that are being processed. Those profiles are used during the computation of edge weights, in cases where one of the optimization parameters is set to error minimization. After the transformation blocks are implemented, the profiles must be computed. There are several ways of obtaining the error profiles. In the case of *Start* nodes the error profiles are usually given in the data-sheet of the sensors, if not they must be estimated experimentally.

ANALYTICAL ESTIMATION OF ERROR PROFILES : For some transformation blocks that implement simple mathematical operations the error profiles can be estimated analytically or are already known. We will demonstrate the estimation of the profile with an example transformation that estimates velocities based on the spacial changes in time. The mathematical implementation of the mentioned transformation block is a numeric estimation of a derivative.

Assume a one dimensional case where we want to estimate the velocity of a point that moves with constant acceleration a at time t_2. Let the position of a point at time t_1 be estimated as $d_1 + \delta d_1$ and at time t_2 be estimated as $d_2 + \delta d_2$, where d_1 and d_2

are the actual positions of the point and δd_1 and δd_2 are the estimation errors. Then the estimated velocity v' of the point at time t_2 would be computed as:

$$\begin{aligned} &v' = \tfrac{d_2-d_1+\delta d_2-\delta d_1}{t_2-t_1} && \Rightarrow \\ &v' = \tfrac{d_2-d_1}{t_2-t_2} + \tfrac{\delta E_{d_{12}}}{\Delta t} \\ &\Delta t \equiv t_2 - t_1 && \delta E_{d_{12}} \equiv \delta d_2 - \delta d_1 \end{aligned} \tag{1}$$

The actual velocity v and the position can be computed analytically from the acceleration:

$$\begin{aligned} v = \textstyle\int a\delta t \Rightarrow v = at \\ d = \textstyle\int v\delta t \Rightarrow \tfrac{at^2}{2} \end{aligned} \tag{2}$$

The relation between the actual and estimated velocities would be:

$$v' = v + E_v \tag{3}$$

where E_v is the estimation error. Our goal is to estimate E_v. From Equation (2) we can determine that the actual positions of the point at times t_1 and t_2 were:

$$\begin{aligned} d_1 = \tfrac{at_1^2}{2} \\ d_2 = \tfrac{at_2^2}{2} \end{aligned} \tag{4}$$

If we plug Equation (4) into Equation (1) we will get:

$$\begin{aligned} &v' = \tfrac{a(t_2^2-t_1^2)}{2(t_2-t_1)} + \tfrac{\delta E_{d_{12}}}{\Delta t} && \Rightarrow \\ &v' = \tfrac{a(t_2-t_1)(t_2+t_1)}{2(t_2-t_2)} + \tfrac{\delta E_{d_{12}}}{\Delta t} && \Rightarrow \\ &v' = \tfrac{a}{2} * (t_2 + t_1) + \tfrac{\delta E_{d_{12}}}{\Delta t} && \Rightarrow \\ &v' = \tfrac{a}{2} * (2t_2 - (t_2 - t_1)) + \tfrac{\delta E_{d_{12}}}{\Delta t} && \Rightarrow \\ &v' = at_2 - \tfrac{a\Delta t}{2} + \tfrac{\delta E_{d_{12}}}{\Delta t} && \Rightarrow \end{aligned} \tag{5}$$

Using Equations (5), (2) and (3) we can determine the error of velocity estimation:

$$E_v = \frac{\delta E_{d_{12}}}{\Delta t} - \frac{a\Delta t}{2} \tag{6}$$

We can see that the error depends on three parameters: the time step Δt, the errors of position estimation $\delta E_{d_{12}}$ and the acceleration a of the point. Depending on the problem, the value of acceleration might not always be available. In such cases we recommend an adaptive estimation of its value: initially make a rough estimate of the acceleration based on current and previous velocities and store the value. Then refine the value of the rough estimate by averaging it with the rough estimates from future iterations.

NUMERICAL ESTIMATION OF ERROR PROFILES : It is not always possible to estimate the error profile of a transformation block analytically. There are cases where either the complexity of the mathematical implementation of the block or the degree of the resulting equations is too large. For those cases we recommend the numerical estimation of the error profile. The estimation process varies on a case to case basis; however, it is possible to draw general guidelines for it. Initially one needs to isolate the set of variables that influence the occurring errors. Further, the operation range with a sampling rate for the mentioned variables must be determined. Once all of the above preparations are made, one must design an experiment that can determine the error-variable correlation in their operation range with the selected sampling rate. The collected data from the experiment needs further processing, where an "n" dimensional curve must be fitted to it. This results in a mathematical representation of error-variable correlation which can be used as an error profile for the considered transformation block. All of the guidelines can be summarized as follows:

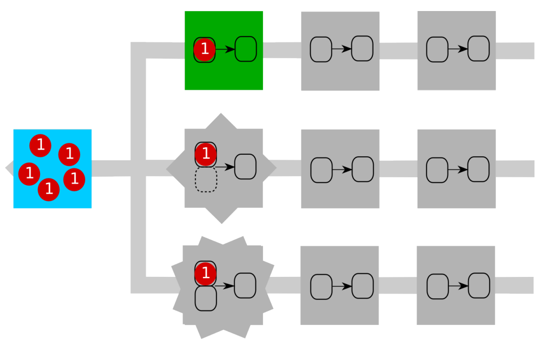
1. Isolate variables influencing errors.
2. Determine their operation range and sampling rate.
3. Perform an experiment to establish error-variable correlation
4. Fit an "n" dimensional curve to the data

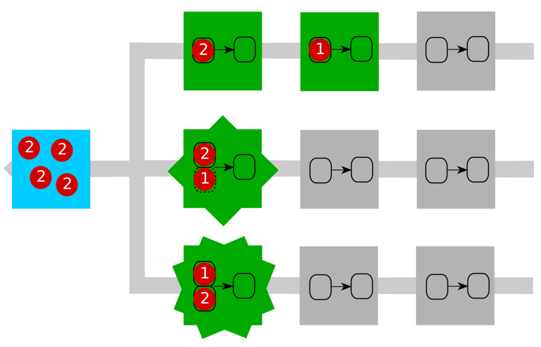
3.4 PROPAGATION OF INFORMATION THROUGH THE GRAPH

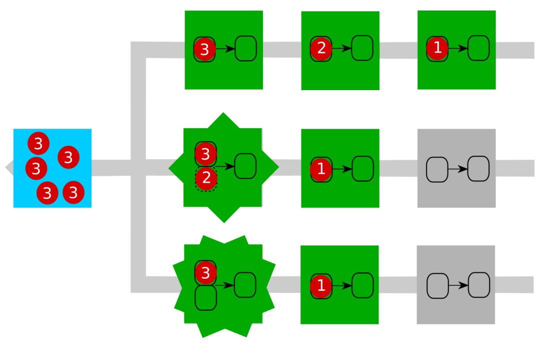
Now that all of the integral parts of the graph are known we will discuss the mechanics behind the information propagation through the transformation graph. To form a general idea on how the data spreads through the transformation graph and how different types of transformation blocks affect its spread we will discuss the mechanics of data propagation. To develop a better understanding of the nature of uncertainty propagation, we will consider a simple transformation graph and illustrate how the errors propagate through it. Finally, we will illustrate how the high-order transformation blocks influence the performance of the chain by discussing the mechanics of complexity propagation.

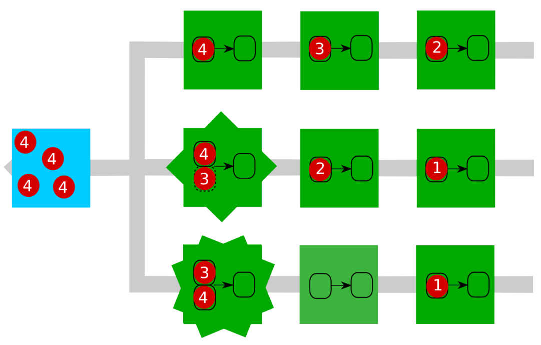
3.4.1 *Data Propagation*

Consider Figure 15. The figure depicts basic transformation blocks as squares, iteration dependent transformation blocks as 8 edge stars and high-order transformations as 12 edge stars. For simplicity reasons the high-order transformation block is set to


(a) First Iteration


(b) Second Iteration


(c) Third Iteration


(d) Fourth Iteration

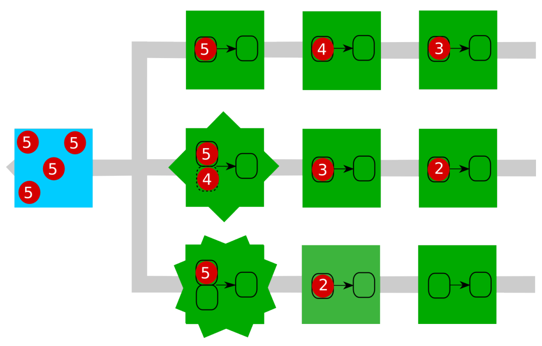

(e) Fifth Iteration

Figure 15: Illustration of the information propagation mechanics though the graph. The square boxes represent basic transformations, the eight edge star represents an iteration dependent transformation. The twelve edge star represents a high-order transformation, which for simplicity reasons is of second order.

second order. The transformation nodes that have not been active are depicted in gray and the transformation nodes that are currently active or have been active are represented in green. The data is depicted as red circles where the numbers in the circle correspond to the iteration during which the data was created. After the first iteration 3.15(a) all of the nodes in the first column receive the data from the initial sensor. The first basic transformation at $(1,1)$ is ready to perform, the iteration dependent at $(2,1)$ and high-order at $(3,1)$ transformations save the current data and do not proceed any further. After the second iteration 3.15(b) the second basic block at $(1,2)$ is activated and a new set of data is generated by the initial sensors and passed to the transformation blocks at $(1,1)$,$(2,1)$ and $(3,1)$. Now both, the iteration dependent and the high-order transformations are activated and can perform. After the third iteration 3.15(c) the basic transformation blocks $(1,3)$,$(2,2)$ and $(3,2)$ are activated. Both, the iteration dependent as well as high-order blocks have performed at least once. The iteration dependent block has stored the data from the second iteration, has received the data from the third iteration and is ready to perform. The high-order block received the third iteration data from the initial sensors and is waiting for further data to perform. After the fourth iteration 3.15(d) all of the transformation blocks in the graph have been activated. Note that both transformations at $(2,3)$ and at $(3,3)$ are active and are ready to process the data from the first iteration. In contrast to them the transformation at $(1,3)$ has already been active and is processing the data from the second iteration. Similar to the second iteration the iteration dependent transformation block has saved the data from third iteration, has received the new data from the fourth iteration and is ready to perform. In all further iterations the iteration dependent transformation block will perform similarly, thus we will not discuss it further. The basic transformation block at $(3,2)$ is idle since the high-order transformation did not perform during the last iteration. The high-order block has received the fourth iteration data and combined with the saved third iteration data is ready to perform. After the fifth iteration 3.15(e) the situation is similar to the third iteration 3.15(c), with the exception that all of the transformation blocks have been activated now. The further flow of the information through the graph would either look like the fourth iteration or the fifth iteration. Consider the transformation blocks at $(1,2)$,$(2,2)$ and $(3,2)$ in the fifth iteration, Figure 3.15(e). All of them have data to process and are active. However, the transformation at $(1,2)$ is already processing the data from the fourth iteration, the transformation at $(2,2)$ is processing the data from the third iteration and the transformation at $(3,2)$ is only processing the information from the second iteration. In all the further iterations the transformation at $(2,2)$ will be processing data with a delay of one in comparison to the transformation at $(1,2)$. The transformation at $(3,2)$ will be processing data half as old as the transformation at $(1,2)$, i.e. if the transformation at $(1,2)$ is processing data from iteration eight, then the transformation $(3,2)$ is only processing information from iteration four. The reason is that the non basic transformations introduce delays in the system. The iteration dependent transformations introduce a delay of one. The high-order transformations cut the rate of the information m times, where m is the order of the transformation.

3.4.2 *Uncertainty Propagation*

Uncertainty or error propagation through the chain of transformations is a complex issue. To get a good understanding of its nature, we will conduct a thought experiment by considering a simple hypothetical transformation chain. The chain consists of five basic transformation blocks, Figure 16. Each transformation block is identical to the next one and implements a simple mathematical equation "$(A/2) + 1$". The pseudo-code of the entire transformation chain is provided at Algorithm 3. Line 2 of the algorithm reveals that in cases where the input variable A is not divisible by 2 the node introduces a rounding error due to the conversion to integer.

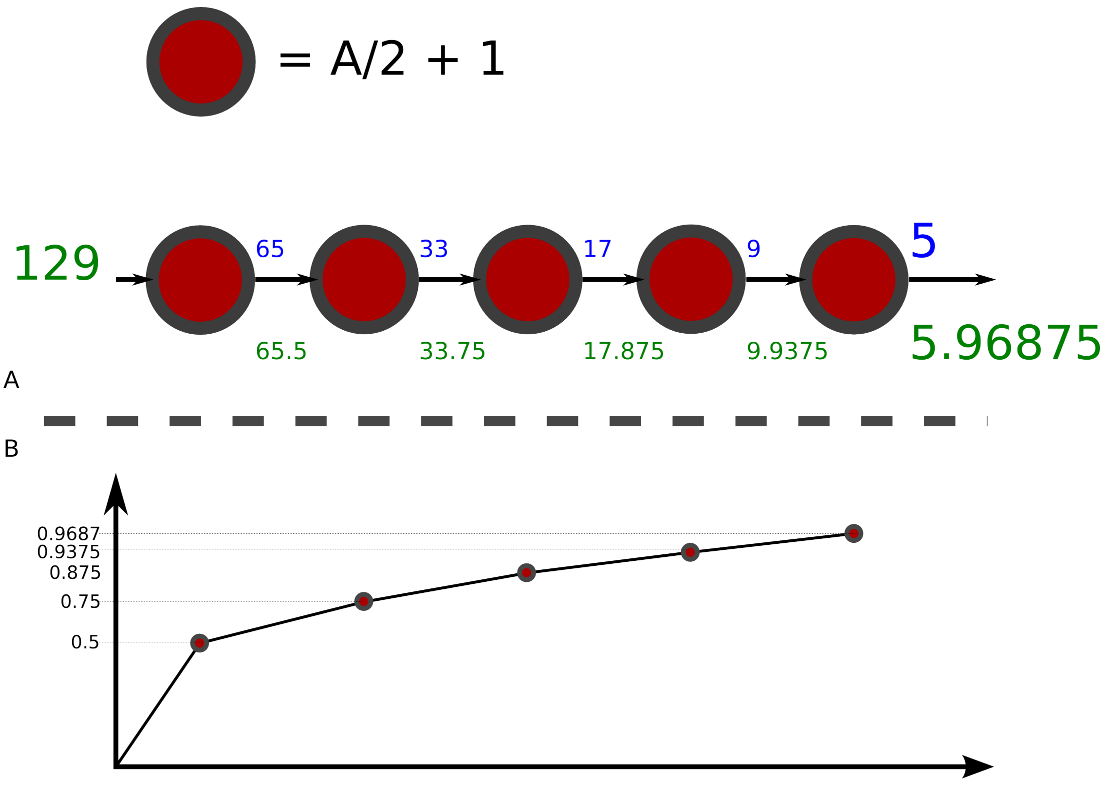


Figure 16: Illustration of the error propagation model through the transformation chain. A) Depicts the transformation chain itself. The latter consists of sequential basic transformations that are all equivalent. Each transformation in the chain implements the "$(A/2) + 1$" simple equation. A complete pseudo-code implementation of the transformation chain is given in Algorithm 3

Let us consider Figure 16 A) for the case where the value of the initial input variable is 129. The values depicted in green are the values that the variable would have taken in the case where there were no rounding errors and the values in blue are the values computed with the rounding error. We can see that the error occurring due to the first transformation is equal to 0.5, Figure 16 B). Further, one can see that each additional

Algorithm 3 Implementation of the transformation chain in Figure 16 A)

```
function processRed(A)
    int res = int($A/2) + 1$
    return res
end function
main($A = 129$)
for all $V \in Range[0:4]$ do
    $A =$ processRed($A$)
end for
return $A$
```

block introduces more errors, so that at the end of the transformation chain the error produced by the system is 0.96875. Note that every single transformation is adding to the previous uncertainty, and that with each additional transformation the error increases due to this addition. Further we can see from the graph that although each transformation in the chain is implementing the same algorithm, the amount of the added uncertainty is not the same. This is due to the fact that the added errors are dependent on the value of the input parameter. As a result of our thought experiment we can see that the nature of the uncertainty/error propagation through the chain is additive and depends on the input value of the variable.

3.4.3 *Complexity Propagation*

In this section we will discuss the effects on the complexity of the entire chain due to the addition of high-order transformation blocks. To do so it is important to understand the difference between run-time and complexity. Run-time is the time it takes for a program to run on a computer. Although at first glance this seems to be the logical way of measuring the efficiency of an algorithm, it is not. The problem with run-time is that it is very hardware specific. The same algorithm can run fast on a modern i7 processor and run very slowly on an older Pentium 166. Ever since the transition from assembler to high level programming languages such as C, C++, Java, etc. the algorithms are being developed on an abstraction level that does not take into account the underlying processor architecture. Therefore a new metric such as complexity has been introduced. This measure is designed to count the number of steps an algorithm has to take to be able to perform. Using the number of steps as a measure of algorithm efficiency is a more logical way of generalization independent of the underlying processor architecture. The complexity of an algorithm is a part of a broader computational complexity theory [6] which is too complex to be explained within the scopes of this thesis.

Algorithm 4 Implementation of the transformation chains in Figure 17 A)

```
1: Transformation chain with basic transformations only
2: main(A)
3: B = basicONE(A)
4: C = basicTWO(B)
5: D = basicTHREE(C)
6: E = basicFOUR(D)
7: return E
```

```
1: Transformation chain with one high-order transformation
2: main(A)
3: Static D1 = None
4: Static D2 = None
5: B = basicONE(A)
6: C = basicTWO(B)
7: D = basicTHREE(C)
8: if D1 == None then
9:     D1 = D
10:    return None
11: end if
12: D2 = D
13: E = high-orderONE(D1, D2)
14: D1 = None
15: D2 = None
16: return E
```

To get more acquainted with the terms, as well as to understand the effects on complexity increase caused by the addition of high-order transformation blocks, let us consider Algorithm 4 and Figure 17 A). The transformation graph consists of two transformation chains which share the first three basic transformations. The fourth transformation in one case is a basic transformation and in the other case a high-order transformation. We will be referring to the latter chain as non-linear and to the former chain as linear. For the sake of simplicity lets first assume that all of the transformations have a similar run time, e.g. 1 second, and that the high-order transformation is of second order. Further the architecture on which the algorithms are executed carries a sequential nature and has no parallel elements.

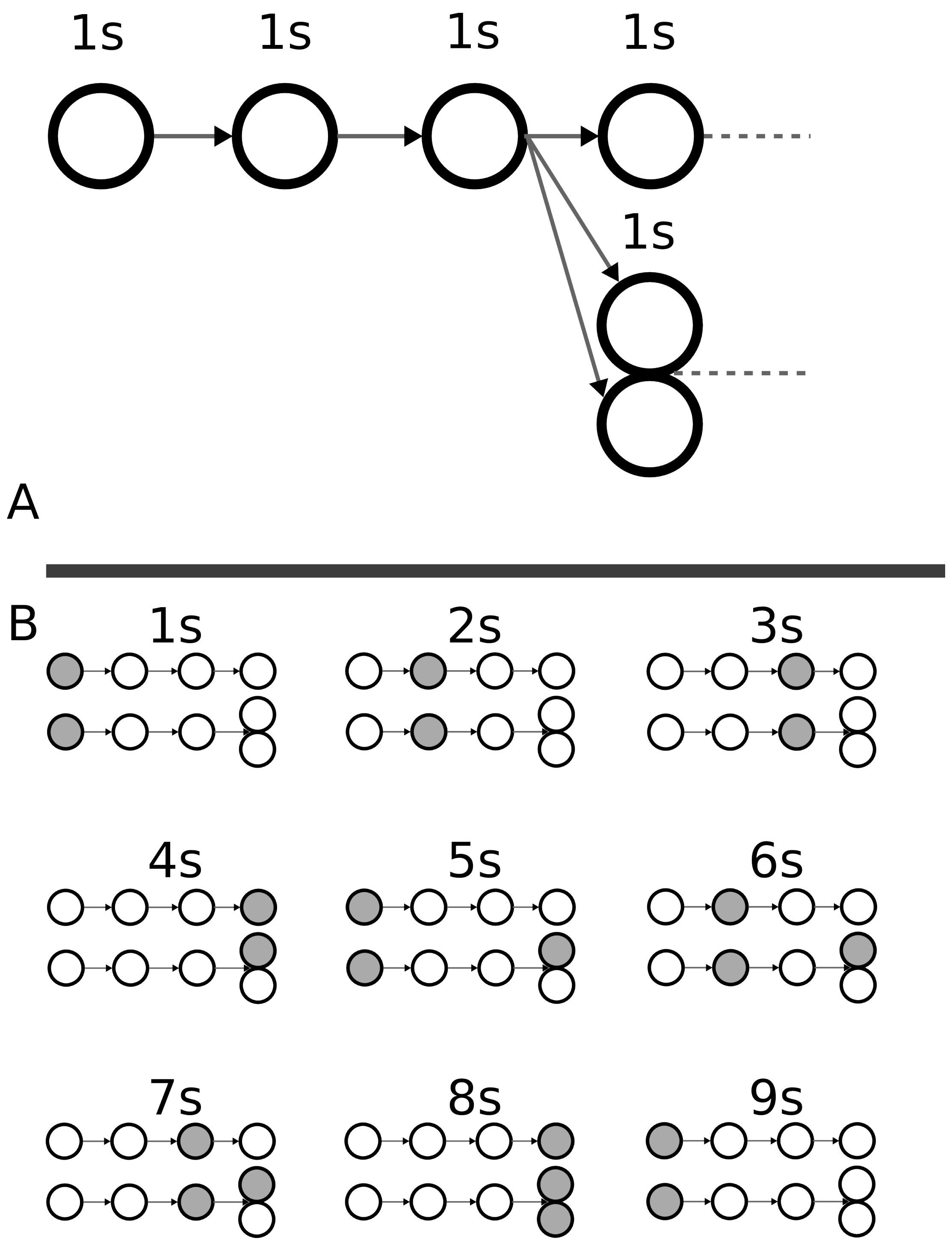


Figure 17: Influence of the high-order transformation blocks on the complexity of the transformation chain. Illustration of a sequential case where the initial sensors operate at a comparable frequency to the processor and the transformations take equal amount of time to perform.

We can see that both the linear and non-linear chains have exactly the same execution sequence and run-time for the first three seconds, Figure 17. In the fourth second the fourth transformation in both of the chains receives the data to process. The transformation in the linear chain is a basic one, thus it can execute and the algorithm will provide a return value. Whereas the transformation in the non-linear chain needs an additional value, thus it is halted and the algorithm returns a blank. Further in the interval from fifth to ninth seconds the linear algorithm processes and returns the data a second time. Contrary to that, the non-linear algorithm uses the time in the interval from fifth to eighth seconds to process the sensory data through the chain and to provide enough information to the high-order transformation in order for it to execute. This results in the non-linear chain returning the first valid value after 9 seconds whereas in the same time the linear chain has returned twice. The main reason why the non-linear chain has needed twice as much time to return than the linear chain is that the high-order transformation was located at its end and the information had to be processed through it twice in order for the high-order transformation to execute. Due to the fact that each transformation takes 1 second to execute the execution time and the number of statements in both of the algorithms are equal.

In the above provided example we assumed that both the initial sensors and the processor on which the algorithms are executed have similar clock frequencies. Lets consider a case where the initial sensors operate at a very slow rate, Figure 18.

Here we have the same scenario with linear and non-linear transformation chains, with the difference that the initial sensors provide information every 30 seconds. In the first four seconds both of the algorithms run as it was in the previous case. After the fourth second the linear algorithm returns a value and the non-linear returns a blank. For the next 25 seconds both of the chains are still and are waiting for the initial sensors to provide data. Once the sensors generate the data, both of the algorithms process and return a value in the 33rd second. Next, both of the transformation chains are still and are waiting for the sensors to provide data in the 60th second. We can see that in the case with slow sensors both transformation chains have exactly the same amount of operations as in the case of fast sensors. The non-linear transformation chain returns values two times slower than the linear one. However, in this case the location of the high-order transformation does not play a role. In such situations the bottleneck of the system is the amount of times the initial sensors need to provide information and not the complexity of the chains themselves. Thus optimization of the chain for complexity does not make much sense.

Finally lets consider the same two transformation chains with the same complexity for the case where the sensors are fast and the transformations take non uniform amounts of time to execute, Figure 19. In this case the shared first three basic transformations take 4, 3 and 1 seconds to execute respectively. Whereas the fourth transformation in the linear chain takes 3 seconds and in the non-linear chain 2 seconds

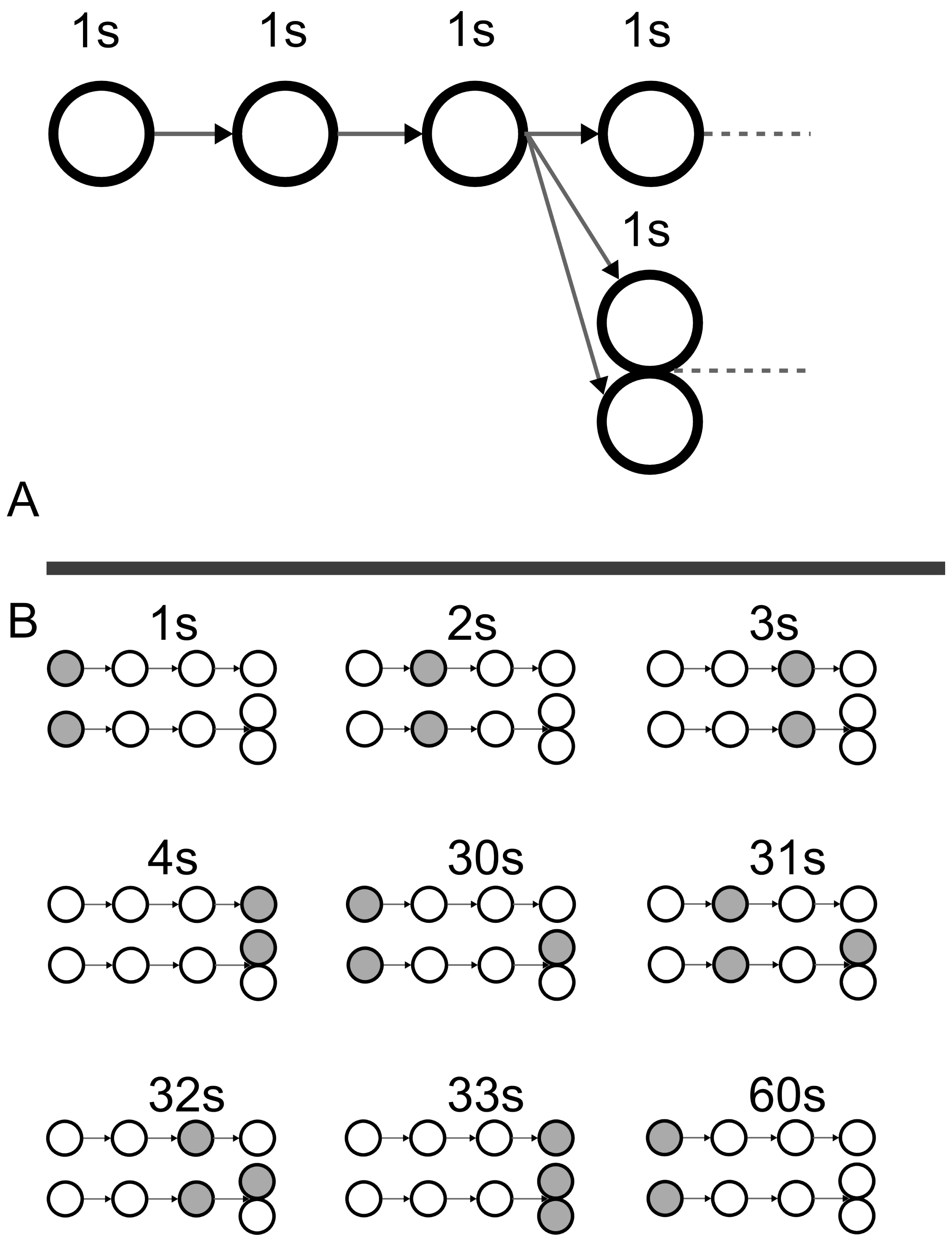


Figure 18: Influence of the high-order transformation blocks on the complexity of the transformation chain. Illustration of a sequential case where the initial sensors operate at a significantly slower frequency compared to the processor and the transformations take equal amount of time to perform.

to execute. Note that due to the increased amount of data to process, the high-order transformations typically take longer to execute than the basic transformations. How-

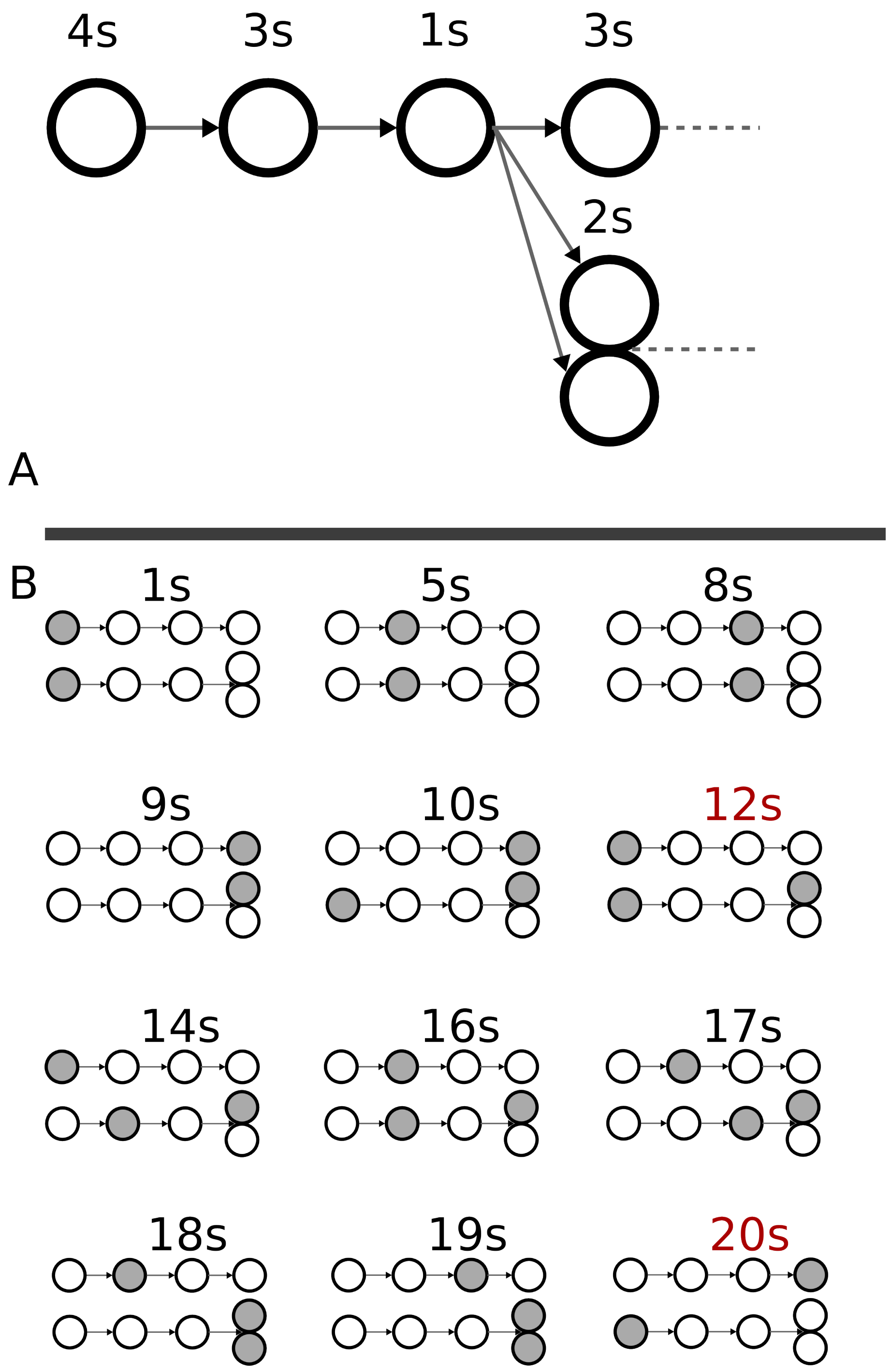


Figure 19: Influence of the high-order transformation blocks on the complexity of the transformation chain. Illustration of a sequential case where the initial sensors operate at a comparable frequency to the processor and the transformations take different amounts of time to perform.

ever, in this case we chose a faster high-order transformation to emphasize the importance of its location in the transformation chain. To avoid confusion we will first consider the linear transformation chain. We can see that the linear transformation chain returns a value after the first 12 seconds, Figure 19 B). After the first 20 seconds the linear transformation chain has processed the data through the shared transformations and will return the second value 3 seconds later once the transformation executes, i.e. it will begin the new processing cycle in the 24th second. It is easy to show that the total time of execution t_{linear} of the linear chain is equal to the sum of execution times of all the individual transformations:

$$t_{linear} = t_{shared} + t_{self} \tag{7}$$

where t_{shared} is the total execution time of the shared transformations, and t_{self} is the execution time of the fourth basic transformation itself. The non-linear chain will get the first value to the high-order transformation in the first 9 seconds and starting from the 10th second it will begin processing the second value through the chain. In the 20th second the non-linear chain will return a value. Note that in this case the execution time of the non-linear transformation chain is not 2 times longer, but instead 1.6 times as long. Although the complexities of the algorithms remain the same in all the three cases the execution time differs. It is easy to show that the execution time $t_{non-linear}$ of the non-linear chain can be computed as follows:

$$t_{non-linear} = 2 * t_{shared} + t_{self} \tag{8}$$

where t_{shared} is the total execution time of the shared segment of the transformation chain and t_{self} is the execution time of the high-order transformation itself. From Equation 7 and Equation 8 we can derive that a high-order transformation increases the run-time of the system $n * t_{prev}$ times, where n is the order of the transformation and t_{prev} is the total execution time of the transformations located before the high-order transformation.

3.5 EDGES AND WEIGHTS

We distinguish between three classes of edges. The class of the edge is determined based on the class of the input block they connect to and is named accordingly. Figure 13 illustrates all three classes. Basic edges are depicted as a single line, iteration dependent edges are depicted as a double line with one of the lines dashed, and the hight-order edges are depicted as two solid lines. Each edge of the *transformation graph* is equipped with a weight. The weights of edges consist of two parts: additive and multiplicative. The values of the weights can depend on several parameters such as: the class of the input block they connect to, the errors that occur in the block they originate from, the parameters that need to be optimized and the operation mode. The weights are usually depicted as light red ovals such as shown in Figure 13. The left argument in the oval represents the additive part of the weight and the right one represents the multiplicative part of the weight. Each of these parts is designed to model a specific attribute of the transformation they are associated with.

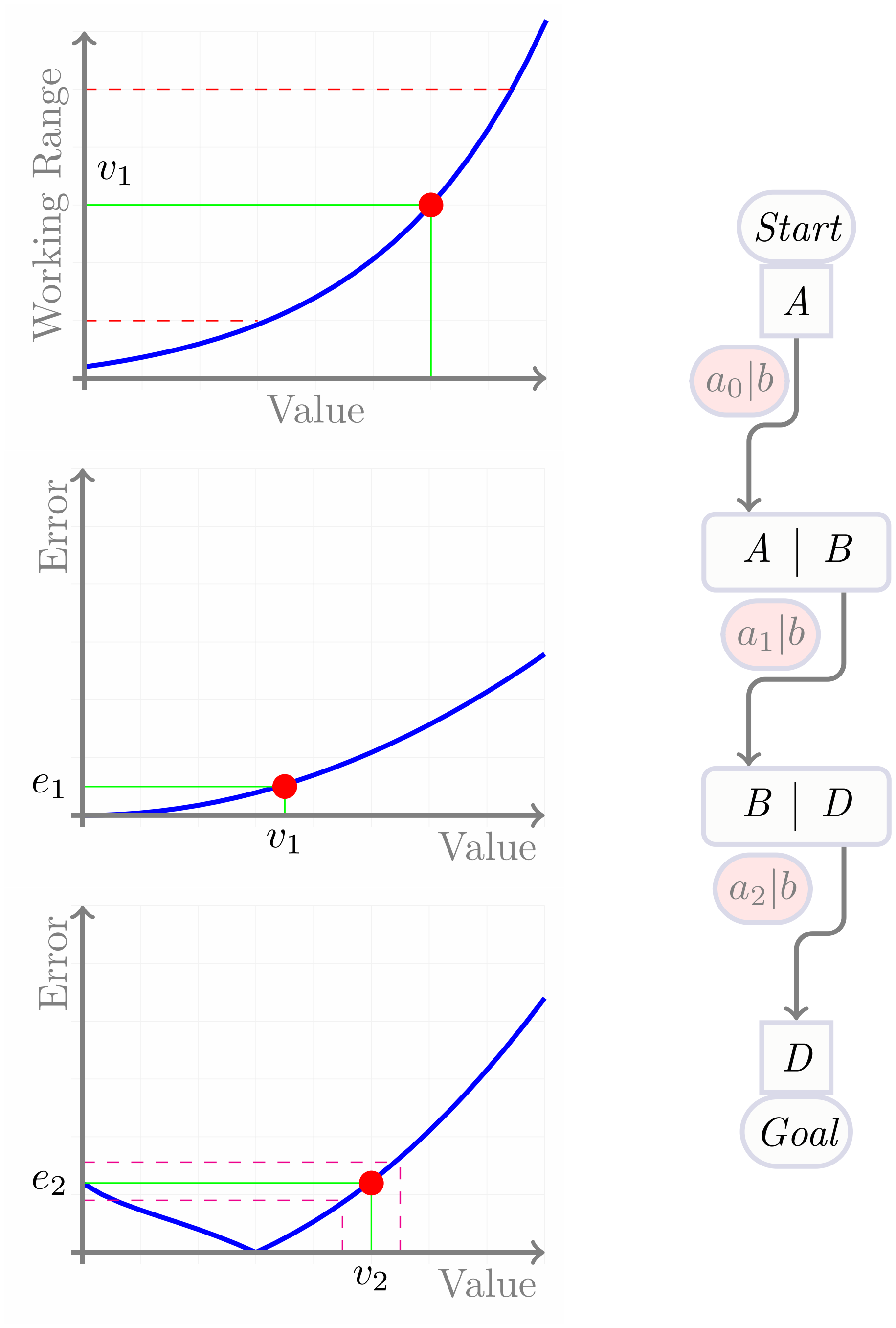


Figure 20: Illustration of the process of assigning additive weights to the edges for the case of error optimization. A generic transformation graph is depicted on the right. The error profiles of the transformation blocks are depicted on the left.

3.5.1 *Additive Weights*

The additive weights are meant to model the occurring errors due to the transformation and are associated with edges leading out from the transformation. Thus, when the optimization parameter of the system is set to minimize complexity alone, all of the additive weights are set to 1. When one of the optimization parameters is set to minimize error, each additive weight is computed using the error profile of the corresponding transformation node. There are several ways of computing these additive weights, and each of them comes with its advantages and disadvantages. We will discuss two extreme cases here. Consider Figure 20, where the *transformation graph* for a generic case is depicted on the right and the error profiles for the *Start*, first and second nodes are depicted on the left accordingly. Here, we will show the process of additive weight computation for static mode only, as computation for dynamic mode is done in a similar fashion. Our optimization is set only for minimum error. The *Start* node represents a sensor that is operating without errors and has an expected value of v_1. Thus the first additive weight a_0 will be set to null. Since we are considering dynamic mode we will propagate the value v_1 with modality "*A*" to compute the additive weight of the next transformation block. Because of the transformation from modality *A* to modality *B* the value v_1 will also change to value v_2 with modality *B*. To compute the additive weight a_1 we take a look at the error profile and see that the error magnitude for value v_1 is e_1. Since v_1 was estimated with no errors there is only one way for computing a_1:

$$a_1 = \frac{e_1}{v_1} \tag{9}$$

Further, in the second transformation block the value v_2 of the modality *B* will be converted to value v_3 of the modality *D*. Since the v_2 was estimated with errors, there are two ways to compute the additive a_2. First, we could use the same technique as we used for computation of a_1. Second, we consider the region $[v_2 - e_1, v_2 + e_1]$ of the error profile of the second transformation, and compute the additive weight a_2 as follows:

$$a_2 = \frac{max(f(x))}{x} \Bigg| \, x \in [v_2 - e_1, v_2 + e_1] \tag{10}$$

where $f(x)$ is a function representing the error profile of the second transformation. The difference between the two methods for computing a_2 is as follows. The first method is computationally fast and does not have a complex implementation. However, the produced error estimates are quite liberal. In contrast to the first method, our second method is more computationally intensive and has a complex implementation. However, the error estimates computed using this method are higher. Thus it puts stricter limitations on the transformation block. In the case that the optimization parameters are set to both minimum error and minimum complexity the additive weights are computed similarly. The one difference being that they all have an additional value of one added to them.

3.5.2 *Multiplicative Weights*

The multiplicative weights are designed to emulate data-flow through the *transformation graph*. Accordingly, they can be used to model the complexity of a particular node. There are two factors that influence the complexity of said node. First, the amount of data required on the input block, as it is directly correlated with the amount of data that needs to be processed at least once. Second is the number of loops that occur within the node itself. This factor is not explicit and varies on a case to case basis since it depends on the implementation of the node itself. Hence, it is hard to provide generic guidelines for it. In the case where the optimization parameter is set to only minimum error the multiplicative nodes are all set to one.

OPTIMIZATION ONLY FOR MINIMUM COMPLEXITY : Consider the case of a transformation node that computes the pose of a camera from a single image. Here, the node will require $(m \times n)$ pixels as input and will provide the pose $[R|T]$ of the camera as output. Where R is a 3×3 rotation matrix and T is a 3×1 translation vector. Then the first factor of representing the complexity will be $(m \times n)$. If no further information is available about the implementation of the node then the best case scenario complexity would be $(m \times n)$ and the multiplicative weight can be set to this value. On the other hand, if it is known that the node contains l loops, stricter restrictions can be used. An $(m \times n \times l)$ would be the worst case complexity. It is possible to directly correlate the first factor influencing the complexity to the class of the input block. By definition, transformations with Basic input blocks require only one value per argument, thus a liberal estimate of the multiplicative weight would be one, while a conservative estimate would be equal to the number of input arguments. Iteration dependent blocks require one value of the argument from the current iteration and have already stored the value of the argument from the previous iteration. This means that in the best case scenario the transformation has to process only two values, therefore even the liberal estimate of the multiplicative weight has to be greater than one. However, since the transformation did not have to wait for the second value to arrive, a tolerant estimate for the multiplicative weight would be less than 2. A strict estimate of the multiplicative weight (i.e. worst case scenario) would be to set to the amount of input arguments plus stored arguments multiplied by the amount of loops, if any are present. Lastly, high-order transformation blocks require Ω input argument values in order to function. This case is similar to calculating camera pose estimation from an image. Thus a tolerant multiplicative weight would be equal to the order of the block, and a strict one would be equal to the order multiplied by the number of loops.

OPTIMIZATION FOR MINIMUM COMPLEXITY AND ERROR : Since there is no strong physical or mathematical correlation between the error accumulation of an algorithm and its complexity, it is hard to strike an exact balance in the assignment of individual multiplicative weights. Therefore a normalized approach is needed. To do so, one needs to consider all of the transformation nodes of the *transformation graph* and isolate the three classes of the transformation blocks into three weight ranges. This means that the multiplicative weights would have values in the following span: basic $\in [1 : b_1]$, iteration dependent $\in (b_1 : b_2]$ and high-order $\in (b_2 : \infty]$, where $1 \leqslant b_1 \leqslant b_2 \leqslant b_3$. The actual values of each weight would depend on the problem. If error minimization has a higher priority than complexity minimization, the multiplicative weights must be assigned liberally. If the opposite prioritization is desired, then a stricter assignment of multiplicative weights is needed. For a balanced approach we will suggest the assignment of all the multiplicative weights of basic edges to one. The amount of processing required by them is minimal; however, they still require time and therefore need weights assigned to them. For multiplicative weights of iteration dependent nodes we would suggest a value between 1 and 2. The individual weights should be computed as relative values where the complexity of the most complex basic node is taken as minimum and the complexity of the most complex iteration dependent block is taken as maximum. If all of the iteration dependent blocks have the same complexity then all of them should be assigned the same weight. Finally, the high-order blocks would get multiplicative weights of 2 and higher, ideally between 2 and 3. Here, the weights are also computed as relative values, similar to the case of iteration dependent nodes.

4 SEARCH THROUGH THE GRAPH

4.1 INTRODUCTION AND OVERVIEW

INTRODUCTION : In this Chapter we discuss generic examples, which illustrate all the modes and steps of our framework. Throughout the process we emphasize the significance of our weight assignment process. Note that all of the weights and values used here will be generic and will not carry any physical or mathematical meaning. The graphs are designed to contain all of the previously described transformation block classes. The latter being placed both at the begging of the graphs as well as at the end in order to accentuate the effects of corresponding weights on the computation of global path distance.

OVERVIEW : Initially we describe Dijkstra's search algorithm, which will be followed by the discussion on modifications that we have made to it. Further we demonstrate the entire work-flow of our framework by using a generic *transformation graph*. Next, we illustrate the path computation of the *transformation graph*, using a variety of optimization parameters. Finally, we conclude by presenting differences in work-flow for static and dynamic operation modes.

4.2 DIJKSTRA'S SEARCH ALGORITHM

Dijkstra's search algorithm is the most popular algorithm in computer science for single source, shortest path estimation. It is an iterative algorithm with a worst case complexity of $\mathcal{O}(n^2)$. It requires a connected graph, where the indices of the source and target nodes are known. The algorithm can be divided into three logical parts: preprocessing, distance computation and reasoning. The preprocessing part is performed only once at the beginning of the execution. At this point, all nodes of the graph, except the source node, are marked as unvisited and their distance is set to infinity. The distance of the source node is set to zero and the node itself is tagged as current. During the computation segment the distance between the current node and all of its adjacent not visited nodes is computed. Further if the newly computed distance to the adjacent node is shorter than the one already present, the adjacent node is updated with the new value. The current node is then tagged as visited and the node with the smallest path distance becomes the current node. This is followed by the reasoning segment, where the current node is compared to the target node. If the match is made then the system returns, if not then the computation segment for the new current node is executed. The path distance to an adjacent node is computed as follows:

$$d = p_c + n_d \tag{11}$$

where d is the path distance to the adjacent node, p_c is the path distance of the current node and n_d is the distance between the current and the considered adjacent nodes. The algorithm is presented in Algorithm 5

Algorithm 5 Dijkstra's search algorithm for finding the shortest path in a connected graph. The algorithm takes as an input a connected graph, the index of the source node and the index of the target node. **Inputs:** Graph, $source$, $target$

```
for all V ∈ Graph do
    dist[V.index] = inf;
    visited[V.index] = false;
end for
dist[source] = 0;
C = Graph[start];
label: 1;
if C.index = target then
    return  dist[C.index];
end if
for all N ∈ V.neighbors do
    if not visited[N.index] then
        d = distance(N,C);
        if dist[N.index] > d then
            dist[N.index] = d;
        end if
    end if
end for
visited[C.index]=true;
C = Graph[index(min(dist[]))];
goto 1;
```

4.3 EXTENDED DIJKSTRA'S ALGORITHM

As already mentioned, we use an Extended Dijkstra's algorithm with amended assignment of weights, and computation of the path distance value to establish the set of optimal transformations. We modified the equation for distance computation between the current and an adjacent node to:

$$d = (p_c + a) * m \tag{12}$$

where d is the newly computed path distance to the adjacent node, p_c is the path distance value of the current node, and a, b are the additive and multiplicative weights of the edge that connects the current and the adjacent nodes.

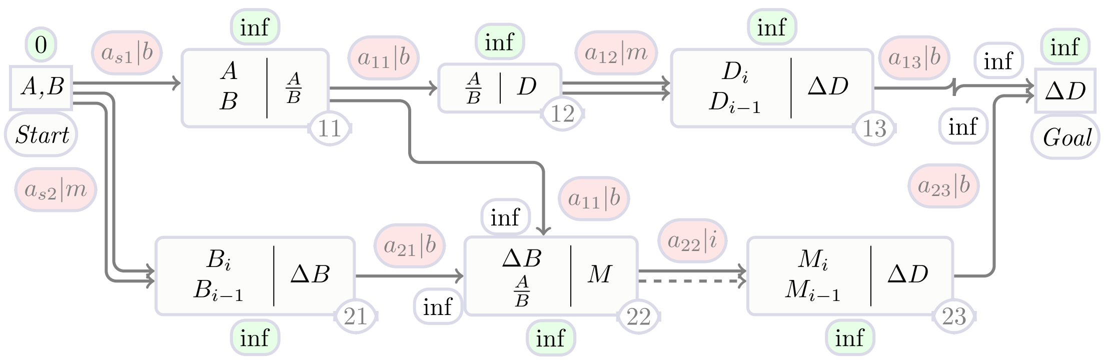
APPLICATION EXAMPLE OF THE EXTENDED SEARCH ALGORITHM : Figure 21 illustrates the generic graph that was constructed according to the rules described in Chapter 3. For ease of describing the process we assigned coordinates to all of the nodes, that can be seen at the bottom right corner. Note that the coordinates are not a part of the framework and are only provided here to simplify explanations. As in the case with the standard Dijkstra's algorithm we begin by assigning the path distances of all the nodes to inf, with the exception of the *Start* node, Figure 4.21(a). Those nodes that are connected to more than one node require an additional individual subpath distance for each connection. For those cases the path distance to the node would be calculated as the summation of the subpath distances. Following this, the path distance is computed to all the nodes of the transformation graph using the same steps as in the standard algorithm.

Figure 4.21(b) depicts an intermediate case where transformation 22 requires two inputs that are provided from different sources. In this case we compute the distance to the considered node as a summation of the two subpath distances. According to the illustration, the distance to one of the inputs is already computed and is set to p'_{22}; however, the distance of the second input is not computed yet and is infinite. Therefore, the global distance to the node is set to $p'_{22} + inf = inf$.

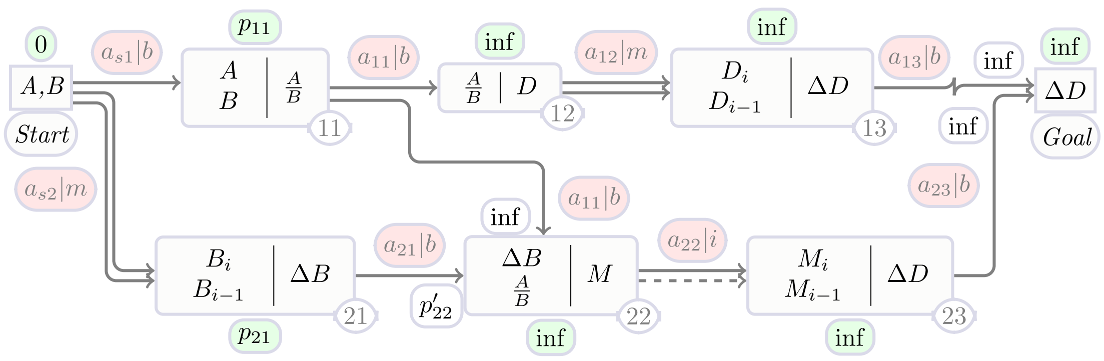
Finally the chain with the shortest path distance is selected as the optimal chain of transformations. Consider Figure 4.21(c), if $p''_g < p'_g$ then the optimal link would consist of $(11, 12, 13)$ transformations. If the opposite is true, then the optimal link would consist of $(11, 21, 22, 23)$ transformations.

4.4 PATH ESTIMATION FOR DIFFERENT OPTIMIZATION PARAMETERS

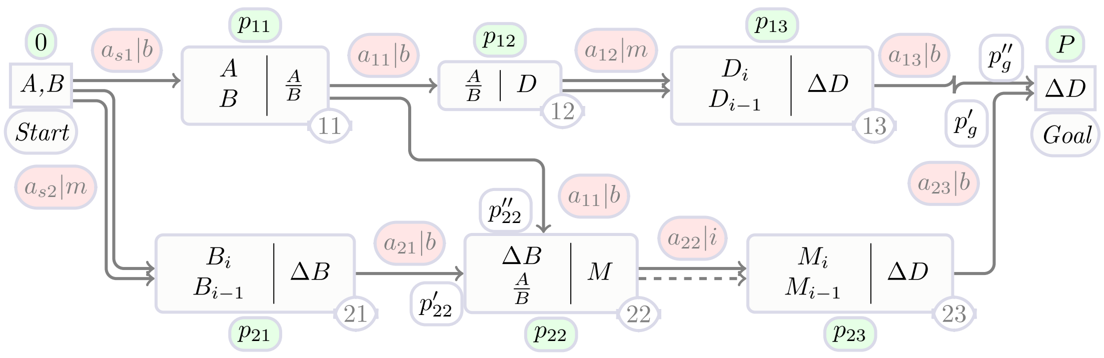
In this section, we will discuss how the "spacial" occurrence of different classes of transformation blocks influence the global path distance computation, considering all three cases of the optimization parameter. We will be using the same generic graph from Section 4.3. Since we are using a purely generic *transformation graph* we will not be addressing the physical meaning behind the transformations. However,



(a) First Step



(b) Intermidiat Step

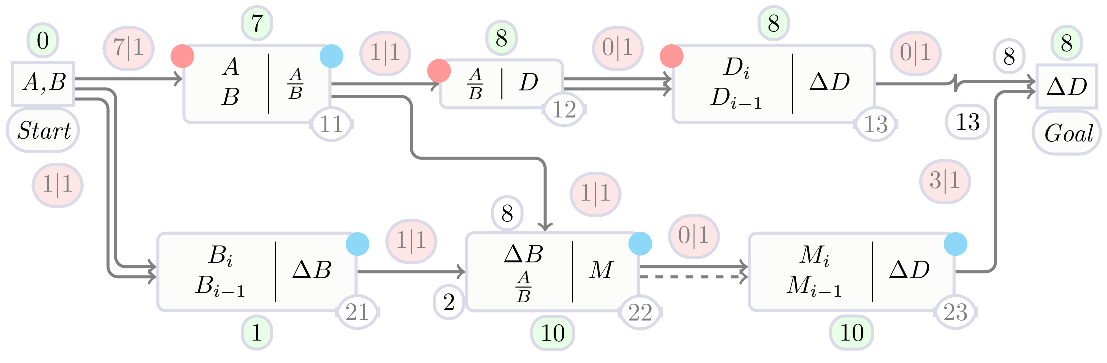


(c) Final result of the search

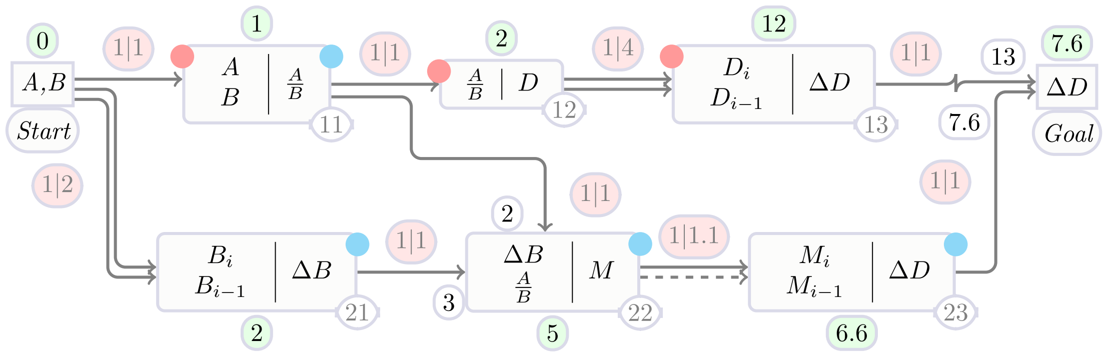
Figure 21: 4.21(a) A depiction of the first step of the search algorithm, here all of the path distances are assigned to inf expect the *Start* node. 4.21(b) A depiction of an intermediate step of the search algorithm. The step was selected to illustrate the handling of multiple inputs of a transformation. 4.21(c) The final result, after all the distances to the goal have been computed.

we will give some mathematical explanations to illustrate the complexity of the non basic blocks. The generic graph in Figure 22 contains three non basic transformation blocks, out of which two are high-order (21, 13) and one is an iteration dependent (13). The first high-order transformation block (21) computes a simple average of both of the input values and contains only one loop. Where the second transformation block (13) computes a weighted average of the input values and contains 2 loops. The transformation with the iteration dependent input block (23) represents a numerical derivative. In this case the derivative is computed according to Equation 13, where the M_{i+1} and M_i are the values of the argument from current and previous iteration steps and h is the step of the function. Since the operation between the two values is a scaled subtraction the complexity of this block is not large. However, it is still an iteration dependent transformation and has a disadvantage to basic transformations.

$$M' = \frac{M_{i+1} - M_i}{h} \tag{13}$$



(a) Optimization for minimum error



(b) Optimization for minimum complexity

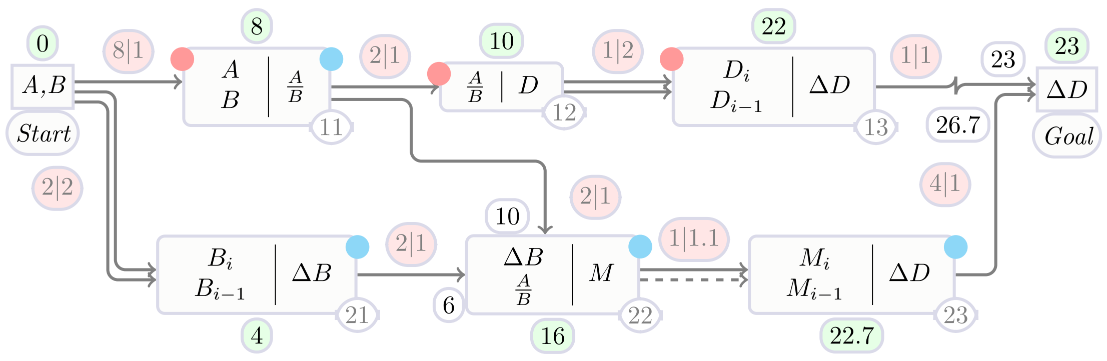
Figure 22: Effects of weight assignment

4.4.1 *Optimization for Minimum Error*

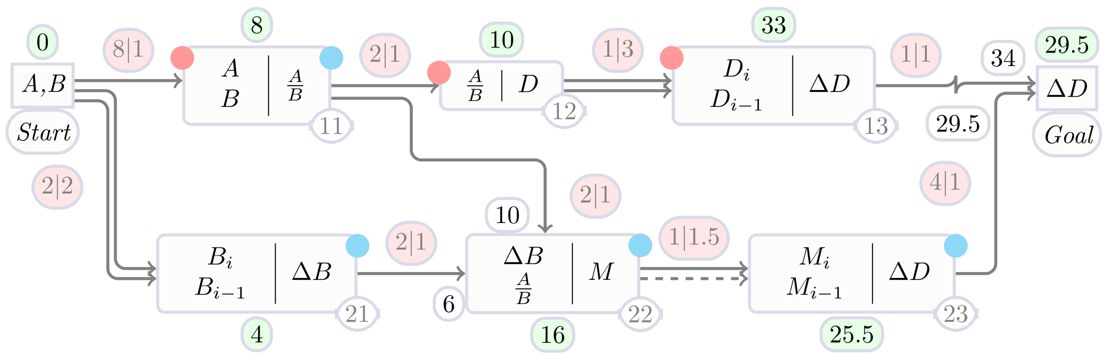
Consider Figure 4.22(a). We have two possible chains: one with a lesser amount of transformations (marked with red dots) and one with more (marked with blue dots). Since in this particular case we are only interested in the minimization of error, all of the multiplicative weights would be set to one. Thus, they will not have any effect on global path distance computation. The initial sensors do not have repeating modalities, therefore we have only one *Start* node. The latter provides [*A*,*B*] as output arguments and has its distance set to 0. The initial sensor that senses in *B* modality, has a relatively small error. In contrast to that, the sensor that senses in *A* modality has a comparatively large error. The *Start* is connected to two transformations (11) and (21). Since (11) requires both high error *A* and low error *B* arguments as its input, the additive weight is set to a comparably high value of 7. On the other hand, (21) only requires *B* as its argument, therefore its additive weight is set to a low value of 1. The transformations (12), (13) and (22) do not introduce any errors, therefore their additive weights are set to 0. This results into them not contributing to the global path distance computation (see path distance change from (12) to (13) or from (13) to *Goal*). From the final result one can see that the global path distance of the link marked with red dotes is significantly smaller than that of the link marked with blue dots. There are two main reasons behind it. First, the red link contains only three transformations, out of which only one contributes to error accumulation. In contrast to that, the blue link contains 4 transformations. The second reason is that the transformation with most error (11) is shared between the two links. Note that the sole reason for error accumulation is the existence of transformation blocks with nonzero additive weights. Thus the error accumulation in the chain is directly correlated to the amount of transformation blocks with nonzero additive weights.

4.4.2 *Optimization for Minimum Complexity*

Consider Figure 4.22(b), since the optimization parameters are set to minimum complexity only, all of the additive weights are assigned a value of one. For reasons described in Section 3.5.2 we assign all of the multiplicative weights of all the basic transformation blocks to one. As was mentioned before, the high-order transformation block (21) only contains one loop, thus its multiplicative weight would be equal to the amount of arguments it requires on the input. On the other hand, the high-order transformation (13) contains two loops, therefore its multiplicative weight would be equal to the amount of arguments multiplied by the amount of loops, i.e. 4. Further we consider the iteration dependent transformation (11), where we assign



(a) Liberal



(b) Balanced

Figure 23: Optimization for minimum error and complexity

the multiplicative weight to 1.1, since it is not a complex block that contains only one scaled subtraction. After the computation of global path distance for both of the chains (red and blue) we can see that the blue chain has a smaller path distance than the red one, although it is considerably longer. The reason behind this is that the high-order transformation block (13) is located almost at the end of the blue chain. Since it requires two values of the same argument at its input, the information has to travel through the chain two times, thus the actual length of the entire chain turns from 5 (i.e. *Start* -> 11 -> 12 -> 13 -> *Goal*) into effectively 8 (i.e. | *Start* -> 11 -> 12 | & | *Start* -> 11 -> 12 | -> 13 -> *Goal*). Further, because of the two loops inside (13) the information that arrives on the input of the block must be processed twice, which introduces more complexity. Whereas the blue chain has the high order transformation at its source, which means its effective length is 8 (i.e. | | *Start* & *Start* | -> 21 | & | *Start* -> 11 | -> 22 -> 23 -> *Goal*), but in contrast to the red chain the last transformation is a relatively low complexity iteration dependent block.

4.4.3 *Optimization for Both Minimum Complexity and Minimum Error*

The final example (Figure 23) is set to optimize for both minimum complexity and error. As we discussed in Section 3.5, it is hard to strike a balance for the assignment of weights since the errors and complexity are not strongly correlated. Thus we will demonstrate two cases: First, where the weights are assigned liberally, i.e. the loops in high-order blocks are not considered and iteration dependent loops are only at a slight disadvantage to basic blocks. Second, a balanced approach where the weights for non basic blocks are assigned based on their comparative complexity in their category. All of the additive weights are incremented by one. This is done because of the nodes that introduce no errors but have complexity, e.g. (22), (13) and (12).

LIBERAL WEIGHT ASSIGNMENT : Figure 4.23(a) demonstrates the results for the liberal weight assignment. We can see here, that the chain with small error accumulation (red chain) has the smaller global path distance. Which means that as a result of liberal weight assignment the optimization for minimum error was given a higher priority. This is due to the fact that for this case, the multiplicative of the node (13) has a smaller impact than it would have for a stricter weight assignment.

BALANCED WEIGHT ASSIGNMENT : In the second example (Figure 4.23(b)) we used a balanced approach for weight assignment. Here the multiplicative weights of the high-order blocks were assigned in the range of $[2-3]$, according to their relative complexity to each other. The iteration dependent blocks were assigned weights between $[1-2]$ based on their relative complexity. Note that since the node (13) has a higher impact value now, the global path distance is shorter for the link with smaller complexity (blue chain). This means that in this case, the optimization for minimum complexity was given a higher priority.

4.5 OPTIMIZATION MODES

Our framework can operate in two modes: static and dynamic. As in case of optimizing for minimum complexity or error there is a trade-off here as well. The dynamic mode estimates the additive weights of the nodes based on the current values of the argument. This means that during every iteration it computes the additive weight for each node and recomputes the optimal path distance. This results in more accurate results with a drawback of longer execution time. Because of its nature, the dynamic mode only makes sense for the cases where optimization for minimum error has high priority. In contrast to dynamic mode, static mode estimates the optimal chain only once in the beginning. It computes the additive weights only once based on the expected values from the sensor. As a result static mode has a short execution time, but the provided results for the minimization of error are less reliable. Thus, it is preferable to use the system in static mode when the emphasis is on minimization of complexity. We will describe the operation for both of the modes based on the generic example illustrated in Figure 24. Here we have two sensors that have one overlapping modality "B", which results into two *Start* nodes. For visual purposes only we depict both of the start nodes as one.

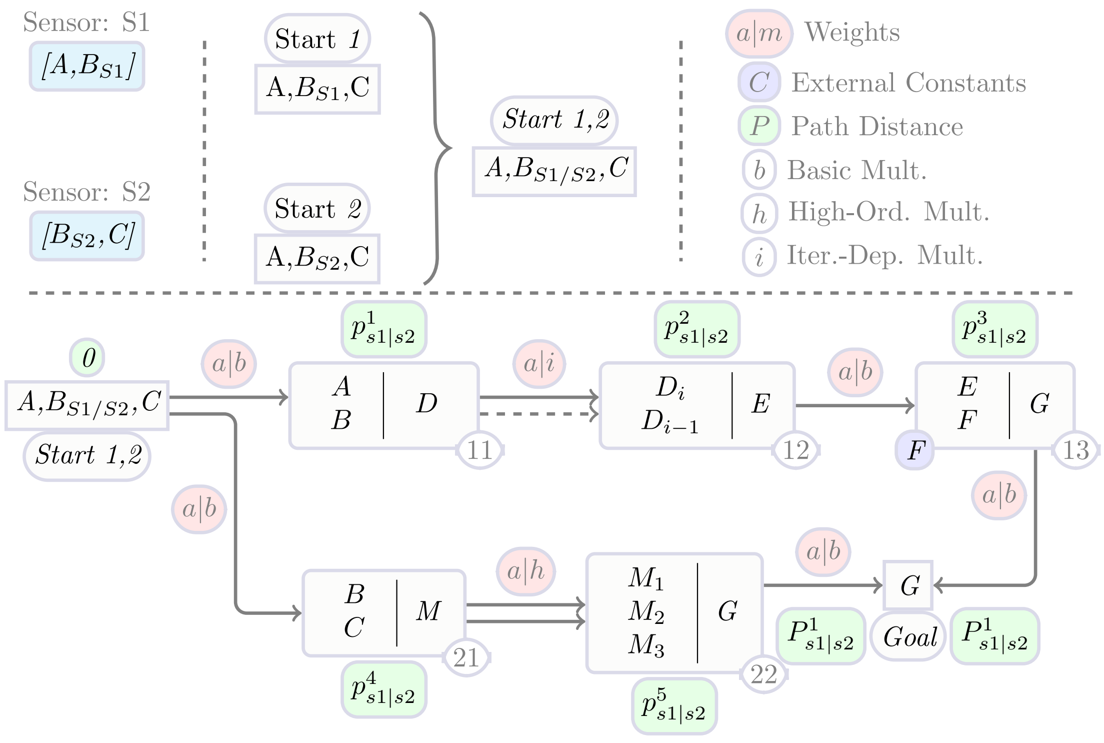


Figure 24: A Generic example to help illustrate the work-flow of the framework in different modes. The system consists of two sensors that have one overlaying modality "B". This results into two *Start* nodes that are illustrated here as one for clarity purposes.

DYNAMIC MODE : We begin with the dynamic mode. Since we have two *Start* nodes we have to repeat the entire process twice. Initially we consider the first *Start* node. It has two edges pointing out of it: one into node (11) and one into (21). Node (11) requires "A, B_{S1}" arguments; thus, we use their current values and the error profile of node (11) to compute the additive weight p^1_{s1}. Further, we use the current values of "B_{S1}, C" combined with the error profile of node (21) to compute the additive weight p^4_{S1}. This concludes the computations for the *Start* node. Further, we consider the first adjacent node to the *Start* node, i.e. node (11). It has only one node pointing out of it into node (12). We use the current value of the output argument of node (11) combined with the error profile of node (12) and compute the additive weight p^2_{S1}. Next, node (21) is considered and its additive weight p^5_{S1} is computed using the same technique. We repeat the process until all of the p^*_{S1} additive weights are computed. Once done, we consider the second *Start* node and use the same technique as in the case for the first *Start* node to compute all of the additive weights p^*_{S2}. Once all of the additive weights are established, we run the extended Dijkstra's search algorithm on both of the graphs. Finally we pick the result that originates from the link with the shortest path distance in both of the graphs. During the next iteration we repeat the entire process again for the new values from the sensor.

STATIC MODE : As in the dynamic mode, we begin with the first *Start* node and its adjacent nodes. In contrast to the dynamic mode, we use the working range of the sensors to establish the expected values for each sensing modality. We compute the additive weights p^1_{S*} to p^5_{S*} (Figure 24) using the same techniques as before with the difference between them being the values used. Once all of the additive weights $p^*_{S1|S2}$ are computed we run the extended Dijkstra's search algorithm and establish the optimal link based on the shortest global path distance. Further, once the link is established we discard all of the transformations and *Start* nodes that are not a part of the link. Finally, during the entire operation of the constructed sensor we use only the values resulting from the optimal link.

4.6 SPECIFICATIONS OF THE TRANSFORMATION GRAPH

In this section we will provide the specifications of the framework for dynamic sensory substitution. The information provided here is the result of a statistical summation for the experiments that will be discussed in the following chapters.

COMPLEXITY OF THE GRAPH CREATION AND SEARCH : Based on the Algorithm 2 it is easy to show that the worst case scenario complexity of the graph creation algorithm is directly correlated to the following expression $n(p + k)$. Where n is the number of available transformations in the bag of transformations, p is the maximum number of input parameters a node can have and k is the maximum number of output parameters a node can have. The maximum number of input parameters that have been encountered in our experiments was equal to 3 and the maximum number of output parameters was equal to 2. Since the maximal number of the input and output parameters is at least by a magnitude smaller than the number of the transformations in a bag of transformations, the complexity of the graph creation algorithm can be written in O notation as $O(n)$. The latter means that the graph creation algorithm has a linear complexity. Each of the connections in the transformation graph are established automatically. The original Dijkstra's algorithm has a complexity of $O(v^2)$ where v is the number of nodes in the transformation. Since our extension to the Dijkstra's algorithm does not result in an introduction of additional loops, the complexity stays the same $O(v^2)$. Since the path creation and establishment is only done once in the static mode, the run-time of the system is merely dependent on individual execution times of each of the transformations as well as their type. The dynamic mode requires a reclassification of the paths during each iteration. Thus, the run-time of the evaluation algorithm with quadratic complexity will be added to the run-time of the system.

SYSTEM SPECIFICATIONS : On average 10 connections are established between 8 transformation blocks in a typical transformation graph. In all of the experiments that we conducted the maximum number of alternative transformation chains in the graph was 2. During the course of our experiments we distinguished between 25 modalities that are listed in Table 1.

N	Name	Representation	N	Name	Representation
1	Gray Image	$\{u, v, g\}$	14	2D Force	$\{ f_x, f_y \}$
2	Color Image	$\{u, v, r, g, b\}$	15	3D Force	$\{ f_x, f_y, f_z \}$
3	Time	t	16	Circle	$\{x, y, r\}$
4	1D Point	p	17	Rectangle	$\{x, y, w, h\}$
5	2D Point	$\{u, v\}$	18	Ellipse	$\{x, y, w, h, \alpha\}$
6	3D Point	$\{x, y, z\}$	19	Square	$\{x, y, a\}$
7	1D Acceleration	a	20	1D Translation	$\{T\}$
8	2D Acceleration	$\{ a_x, a_y \}$	21	2D Translation	$\{t_x, t_y\}$
9	3D Acceleration	$\{ a_x, a_y, a_z \}$	22	3D Translation	$\{t_x, t_y, t_z\}$
10	1D Velocity	v	23	2D Rotation	$\{R_{2D}\}$
11	2D Velocity	v_x, v_y	24	3D Rotation	$\{R\}$
12	3D Velocity	v_x, v_y, v_z	25	External Constant	K
13	1D Force	f			

Table 1: Table of modalities used during the experimentation process

Part II

APPLICATION EXAMPLES

5 SHORT INTRODUCTION TO THE EXPERIMENTAL VALIDATION

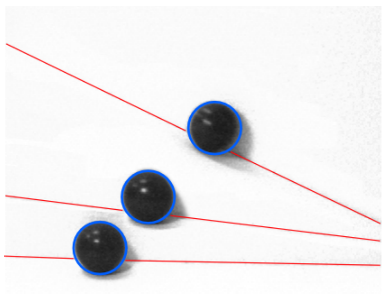
In the second part of this dissertation, we will provide some application examples of our framework as well as experimental evaluation. We begin with a validation of error estimation accuracy, where we demonstrate how the absolute errors estimated using our framework compare to the real world errors. Following this, we demonstrate how our framework performs in optimization for minimal errors in dynamic mode, where the framework is tasked to dynamically select the most optimal chain of transformations depending on the operation mode of the initial sensors. We will conclude this part by demonstrating the usage of our framework in the design and construction of a sensor. The constructed sensor is first analyzed using our framework to pick the most optimal chain of transformations. Next, a real world sensor is constructed based on the prior analysis. Finally, we present the characteristics of the constructed sensor.

6 VALIDATION OF THE ERROR ESTIMATION ACCURACY

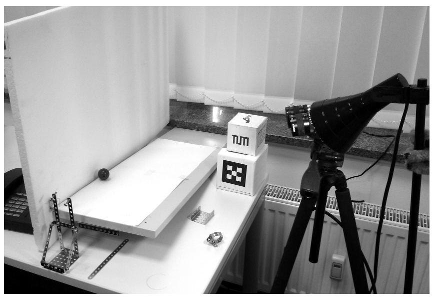
6.1 INTRODUCTION AND OVERVIEW

INTRODUCTION : In this chapter we will experimentally demonstrate how the errors estimated using our framework compare to real world errors. To do so we designed an experiment where a solid ball is let to freely roll down a slope influenced only by the force of gravity, Figure 25. The ball is simultaneously observed by a high frame-rate camera that estimates its velocity using some mathematical and physical manipulations. This experiment is modeled after a well known physics problem which allows us to analytically determine the acceleration and the velocity of the ball at any given point in time.

OVERVIEW : We will initially discuss the *transformation graph*, its operation mode and the optimization parameters. Following this, we will analytically compute the error profiles of the transformation blocks. Next we will describe in detail the experimental setup and the conducted experiments that are based on the error profile of the entire transformation chain. We will conclude by demonstrating the results from the experiments.

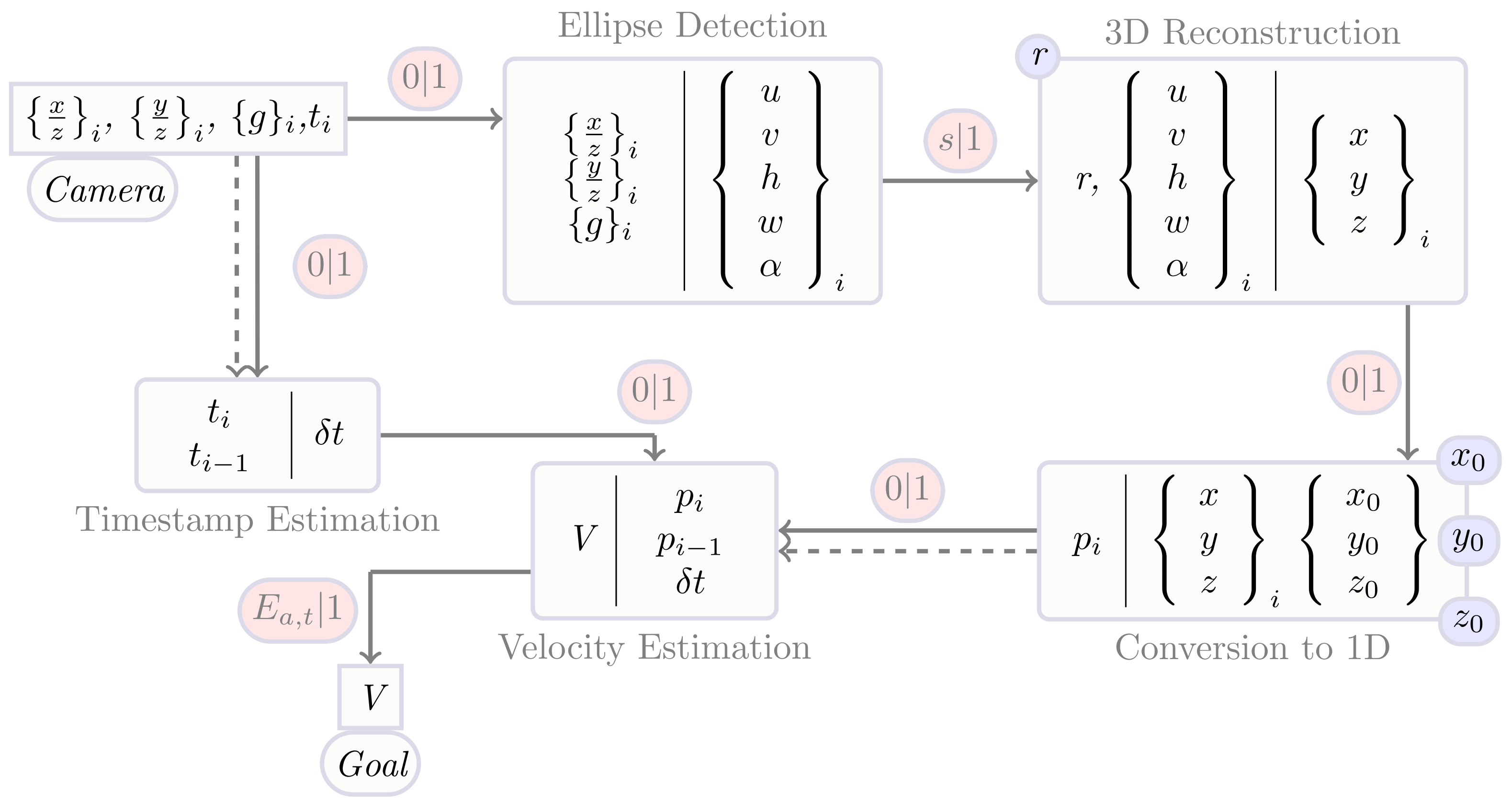


(a) Different angles of tilt



(b) Setup

Figure 25: Illustration of the setup for the experiments conducted to estimate the relation of the real world errors to the computed global path distance. A ball was let to roll freely on the flat surface. The acceleration of the ball was measured for different angles of tilt.



$\{\frac{x}{z}\}_i, \{\frac{y}{z}\}_i, \{g\}_i, t_i$
Camera
0|1
Ellipse Detection
$\{\frac{x}{z}\}_i$ $\{\frac{y}{z}\}_i$ $\{g\}_i$
$\{u, v, h, w, \alpha\}_i$
s|1
3D Reconstruction
r
$r, \{u, v, h, w, \alpha\}_i$
$\{x, y, z\}_i$
0|1
0|1
t_i t_{i-1}
δt
Timestamp Estimation
0|1
Velocity Estimation
V
p_i p_{i-1} δt
0|1
Conversion to 1D
p_i
$\{x, y, z\}_i$ $\{x_0, y_0, z_0\}$
x_0
y_0
z_0
$E_{a,t}|1$
V
Goal

Figure 26: The *transformation graph* for estimating the velocity of a rolling ball. The framework is set to operate in dynamic mode with the optimization set to only minimize errors. Some of the transformations introduce no errors therefore their additive values are set to zero in advance.

6.2 TRANSFORMATION GRAPH BASED ANALYSIS

Figure 26 illustrates the *transformation graph* of the experiment. Since the goal of our experiment is to determine the relation of the estimated errors to real world errors, the operation mode is set to dynamic and the optimization parameters are set for error minimization. This results in all of the multiplicative weights being set to one. The only sensor of our system is a high frame-rate camera. Therefore, we have only one *Start* node that provides the rays pointing from camera center to the environment alongside with color and timestamps. The data provided by the *Start* node is sensed without errors. Therefore, the additive weights of the edges pointing out of it are set to zero. Since the angle and the distance between the camera and the rolling ball is fixed, the only errors occurring in the transformation responsible for "*Ellipse Detection*" are due to the flickering of light and discrete nature of the input data. Those errors have a static nature. Thus, the additive weight of the edge pointing out of the transformation is set to a constant "s". A projection of a sphere on a plane is always a circle; therefore, the transformation responsible for "*3D Reconstruction*" implements scaling of the input data and does not introduce any errors of its own. The transformation responsible for "*Conversion to 1D*" implements a vector subtraction and norm computation. It too does not introduce any new errors; thus, the weight of the edge pointing out of it is set to zero. The errors occurring due to numeric derivation (transformation responsible for "*Velocity Estimation*") where computed in Section 3.3.4 and are equal to:

$$E_{a,t} = \frac{\delta E_{d12}}{\Delta t} \pm \frac{a\Delta t}{2} \tag{14}$$

where $E_{a,t}$ is the error of velocity estimation, δE_{d12} is the error with which the pose was detected, a is the acceleration of the ball and Δt is the time-step. Since we only have one chain we can estimate an analytical equation for computing the amount of the global error accumulation using the rules described in Chapter 3:

$$L = E_{a,t} + s = \frac{\delta E_{d12}}{\Delta t} \pm \frac{a\Delta t}{2} + s \tag{15}$$

where E_{d12} is directly related to the static error s. Therefore, it has a static nature. Hence the only variables influencing the error are the acceleration a and the time-step Δt.

6.3 EXPERIMENTAL VALIDATION

EXPERIMENTS : As it was shown in the previous Section there are two parameters that are influencing the error of the system: the time-stamp Δt and the acceleration of the ball a. Therefore, the conducted experiments are split into two parts. In the first part we let the ball roll down the ramp with a fixed tilt. We register the entire motion using our high frame-rate camera which captures 120 frames per second. Further we use the captured data and compute the velocity of the ball during rolling. This is repeated several times for different values of Δt. In the second part we conduct the same experiment. However, now we keep the time-step Δt constant and alter the acceleration a of the ball. This is realized by altering the tilt angle of the ramp, Figure 6.25(a). Further, the errors for both cases are computed using the relations of current measured values to the ones that are analytically estimated.

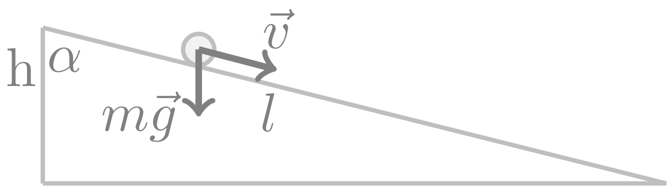


Figure 27: Sketch of the experimental setup. The ball was let free to roll on the ramp, influenced only by gravity. The experiment was repeated with different slopes.

ANALYTICAL ESTIMATION OF THE ACCELERATION : Consider a solid ball with a mass m and radius r that was let to roll freely on a ramp with height h and length l, Figure 27. The goal is to estimate the acceleration a of the ball, since it will completely determine its velocity at any given point in time $v = at$. According to the law of energy conservation:

$$mgh = \frac{mv^2}{2} + \frac{I\omega^2}{2} \tag{16}$$

where g is the magnitude of the gravity vector, I is the moment of inertia of the ball, ω is the angular and v is the linear velocity of the ball at the bottom of the slope. Since the forward propulsion of the ball is solely due to its rotation, the linear velocity of the ball is equal to the linear velocity of the surface of the ball:

$$v = \omega r \tag{17}$$

Thus we can write the Equation (16) as follows:

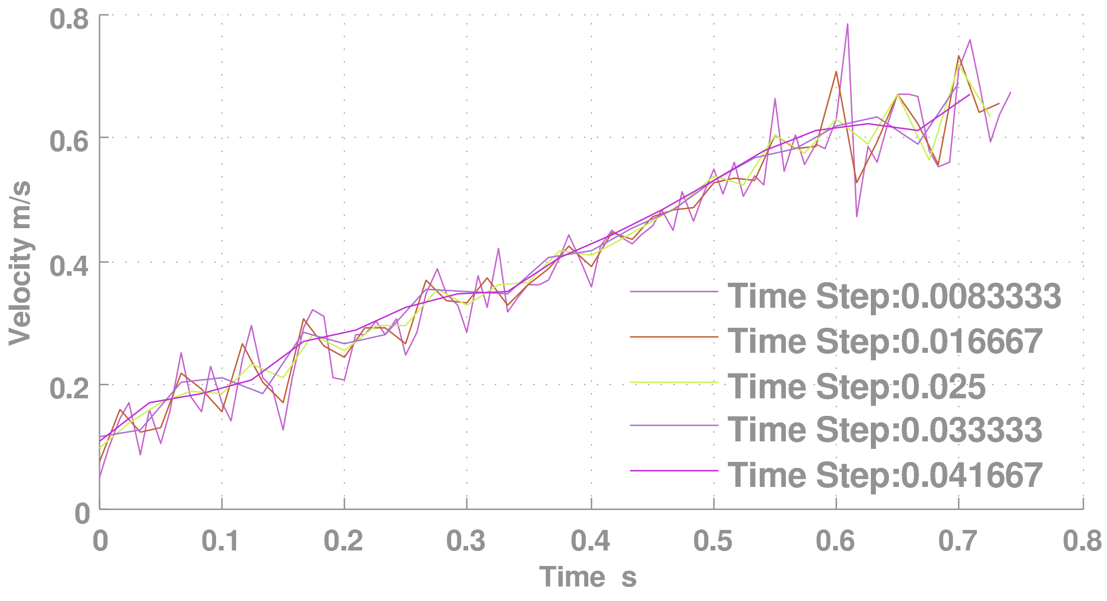
$$mgh = v^2(\frac{m}{2} + \frac{I}{2r^2}) \tag{18}$$

The moment of inertia for a solid ball can be computed as such:

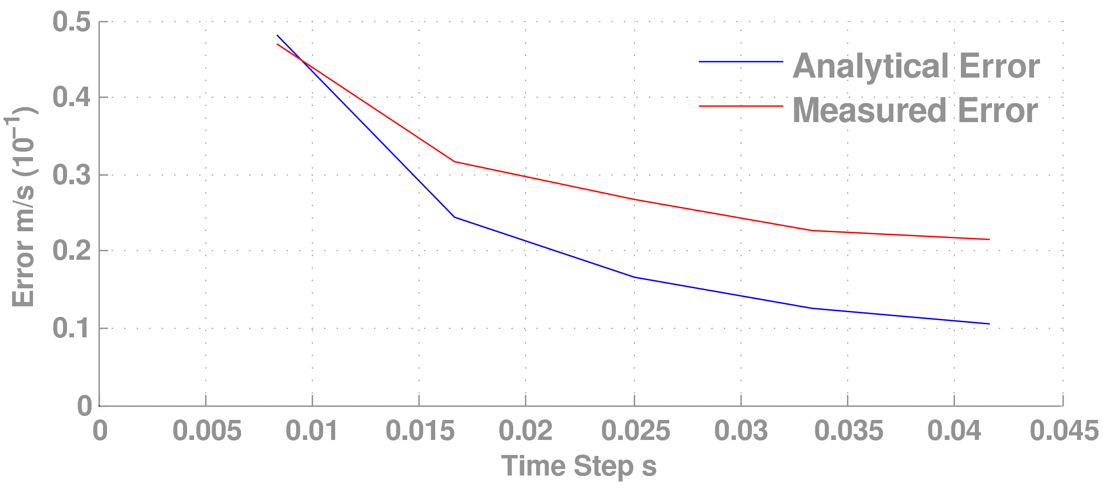
$$I = \frac{2}{5}mr^2 \tag{19}$$

We combine Equations (19) and (18), which results into:

$$mgh = v^2(\frac{m}{2} + \frac{2mr^2}{10r^2}) \tag{20}$$



(a) Alternating Time-Steps



(b) Resulting Errors

Figure 28: Illustration of the experimental results. Here the ball was let free to roll on a surface with a fixed slope. Figure 6.28(a) illustrates velocities of the ball that were computed using different time-steps. Figure 6.28(b) illustrates the resulting relative errors for different time-steps.

Finally after canceling the m and r^2 we get the following equation for the velocity of the ball at the bottom of the ramp:

$$v^2 = \frac{10}{7}gh \tag{21}$$

The exit velocity v can be defined as a time integral of acceleration a; thus, for the entire time of travel it is equal to $v = at$. h can be represented as $l \cos \alpha$ where l is the length of the ramp. In this particular case l is also equal to the distance that the ball has traveled to achieve velocity v thus it can be computed as an integral of velocity over time or a second integral of acceleration over time $l = \frac{at^2}{2}$. After plugging the mentioned values into Equation (21) we get the following:

$$a^2t^2 = \frac{5}{7}at^2 g \cos \alpha \tag{22}$$

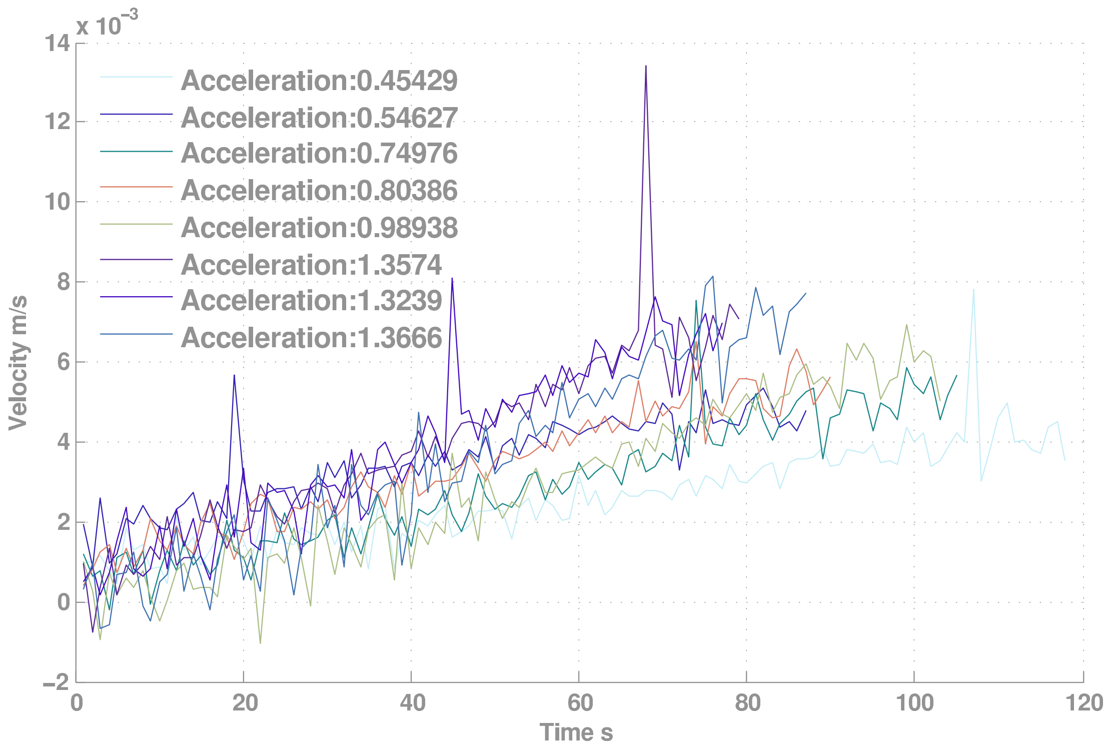
which results into an equation for computing the acceleration of the ball:

$$a = \frac{5}{7} g \cos \alpha \tag{23}$$

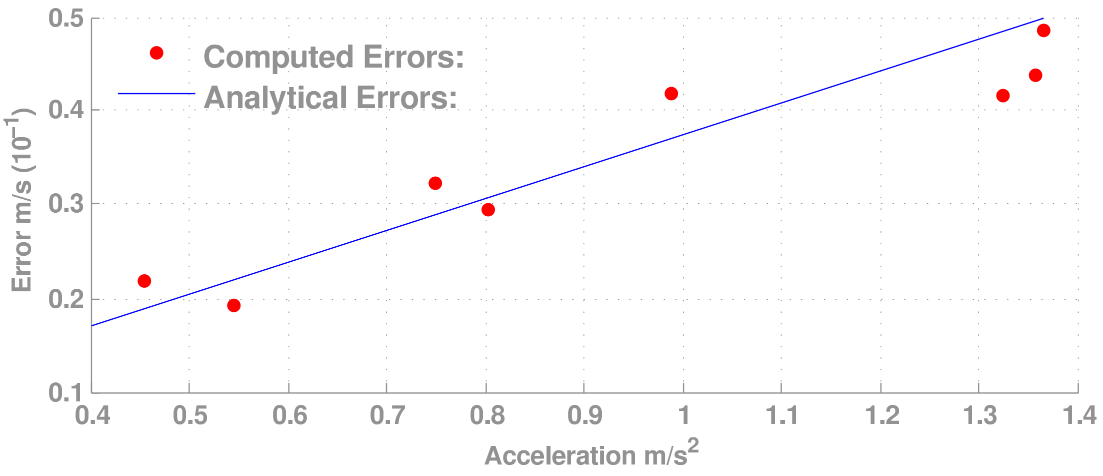
6.4 EXPERIMENTAL RESULTS

We conducted the first set of experiments for altering time-steps and fixed acceleration. Figure 6.28(a) illustrated the computed velocities. The velocities depicted in bright green have the smallest time-step $\Delta\ t = \frac{1}{120}$. For the consecutive measurements the time-step was increased to $\frac{2}{120}, \frac{3}{120}, \frac{4}{120}, \frac{5}{120}$ accordingly (see the legend of the illustration). Figure 6.28(b) demonstrates the comparison of the measured errors (red line) to the analytically calculated errors (blue line). According to the Equation (15) the error dependency to the time-step is a summation of a linear and an inverse functions. For small time-steps the error behaves similarly for both of the cases. The small drift in measured errors can be explained with some noise that was not taken into account.

Figure 6.29(a) illustrates the estimated velocities for the case when the tilt of the ramp was altered (i.e. the acceleration of the ball) and the time-stamp was kept constant. According to Equation (15) the error dependency to the acceleration is linear (Figure 6.29(b), blue line). This corresponds to the actual measured errors (red dots), with an addition of some noise that was not taken into consideration. Note that in both cases [fixed acceleration/altering time-step] and [altering acceleration/fixed times-step] the error estimating using our framework relate quite closely to the real world errors.



(a) Alternating Time-Steps



(b) Resulting Errors

Figure 29: Illustration of the experimental results. Here the ball was let free to roll multiple times on a surface. For each roll the slope of the surface was altered. Figure 6.29(a) illustrates the estimated velocities for different accelerations. Figure 6.29(b) illustrates the occurring relative errors.

7 OPTIMIZATION FOR MINIMAL ERRORS IN DYNAMIC MODE

7.1 PROBLEM STATEMENT

In this chapter we will demonstrate how the framework performs in a multi-sensor environment. The goal is to show how the selection of the optimal path is conducted in dynamic mode. We use a modern cellphone as a multi-sensor platform. The latter is equipped with a camera and an accelerometer that are treated as the initial sensors. We set the desired modality to the acceleration of the cellphone. The accelerations registered from the accelerometer on the cellphone have a low signal to noise ratio for smaller accelerations and a large signal to noise ration for large acceleration. On the other hand, accelerations computed from the camera have lower errors for smaller accelerations but tend to be unreliable for large acceleration. This is due to the fact that the derivation error due to time discretization is directly proportional to the magnitudes of acceleration and jerk, Equation (24).

$$E_T = \frac{f''(\xi) * \Delta t}{2} \tag{24}$$

where E_T is the error due to the derivation, $f''(\xi)$ is the second time derivative of the function and ξ is just a number in the range $[t_0; \Delta t]$ (t_0 being current time).

7.2 THE TRANSFORMATION GRAPH

OVERVIEW OF THE GRAPH : The transformation graph of the system is illustrated in Figure 30. For simplicity of visualization we have divided the *Start* node into two parts. The part responsible for data acquired from the camera is depicted at the top of the image and is marked as "Camera", the part responsible for data collected from the accelerometer is depicted at the bottom left corner and is marked as "Accelerometer". As it was mentioned before, there are two ways of obtaining the accelerations of the cellphone: directly through the accelerometer, or through a set of transformations performed on the camera image. Since the system operates in dynamic mode, the additive weights are determined online based on the error profiles for each transformation. The obtaining of the error profiles will be discussed in Section 7.3. In each iteration the system computes the path value for both of the chains. Further, it estimates which value is more reliable based on the computed path values.

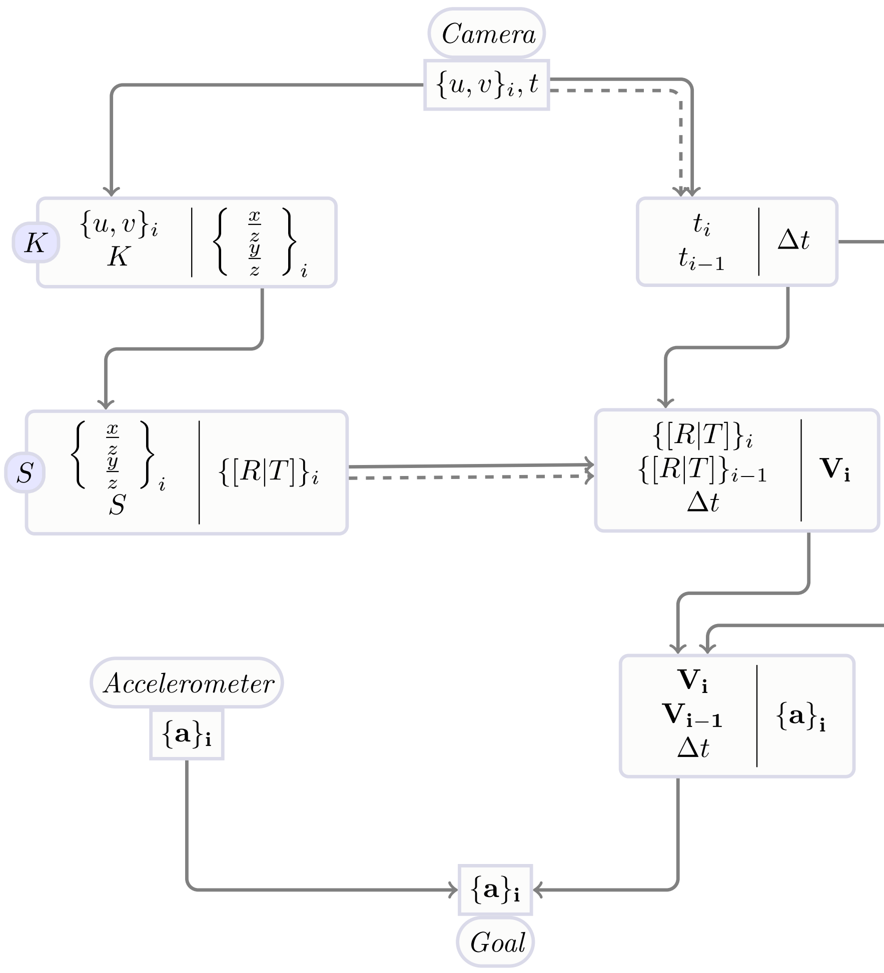

Camera
$\{u,v\}_i, t$
K
$\{u,v\}_i$ K
$\left\{ \frac{x}{z}\ \frac{y}{z} \right\}_i$
t_i t_{i-1}
Δt
S
$\left\{ \frac{x}{z}\ \frac{y}{z} \right\}_i$ S
$\{[R|T]\}_i$
$\{[R|T]\}_i$ $\{[R|T]\}_{i-1}$ Δt
$\mathbf{V}_i$
Accelerometer
$\{\mathbf{a}\}_i$
$\mathbf{V}_i$ $\mathbf{V}_{i-1}$ Δt
$\{\mathbf{a}\}_i$
$\{\mathbf{a}\}_i$
Goal

Figure 30: Depiction of the transformation graph designed to estimate the best acceleration value of a cellphone. There are two ways to perform. First, by reading the data directly from the accelerometer located on the cellphone. Second, by computing the accelerations based on the images acquired by the camera located on the cellphone.

THE TRANSFORMATION NODES : Since the camera is only registering images with timestamps, the first transformation (left) is a conversion from the image coordinate system to the world coordinate system. Here (u, v) are the image coordinates, K is the intrinsic camera matrix, $\{\frac{x}{z}\}_i$ and $\{\frac{y}{z}\}_i$ are the pixels in world coordinate system. The second step in the chain is the reconstruction of the camera pose (second transformation left). We obtain the rotation and the translation [R|T] pair that describe the pose of the cellphone by means of a square marker tracking approach, Figure 31. The input parameters of the transformation are the rays that point from the camera center to the corners of the marker $\{\frac{x}{z}\}_i, \{\frac{y}{z}\}_i$ and the Σ scale of the marker, i.e. the actual physical size of one of the edges of the square marker. The transformations on the right side have a physical nature. The first one on top is just a mathematical subtraction that computes the time-step from two consecutive timestamps $t_i; t_{i-1}$. It

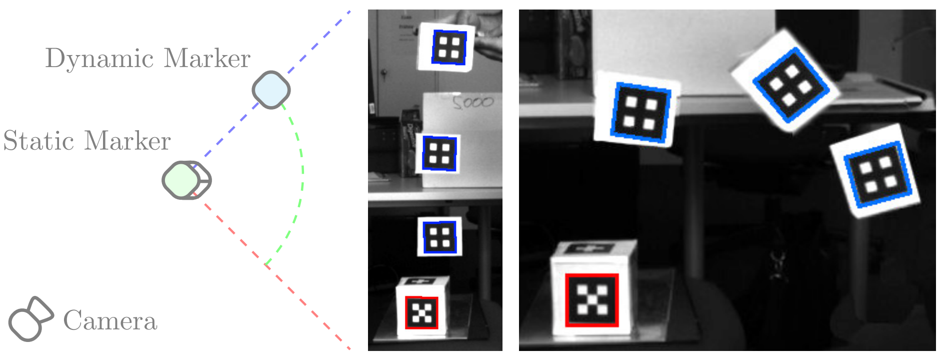


Figure 31: 7.31(a) Sketch of the experimental setup for obtaining the error profile. Note that the camera is placed in such a way that the effective distance to the static and dynamic markers remain practically constant. 7.31(b) Illustration of the experiment for computing the [error/target distance] dependency. 7.31(c) Illustration of the experiment for computing the [error/viewing angle] dependency.

is followed by velocity $\vec{v}$ estimation from two consecutive positions and a time-step, which in its turn is succeeded by acceleration $\mathbf{a}$ estimation from two consecutive velocities and a time-step.

7.3 COMPUTATION OF ERROR PROFILES

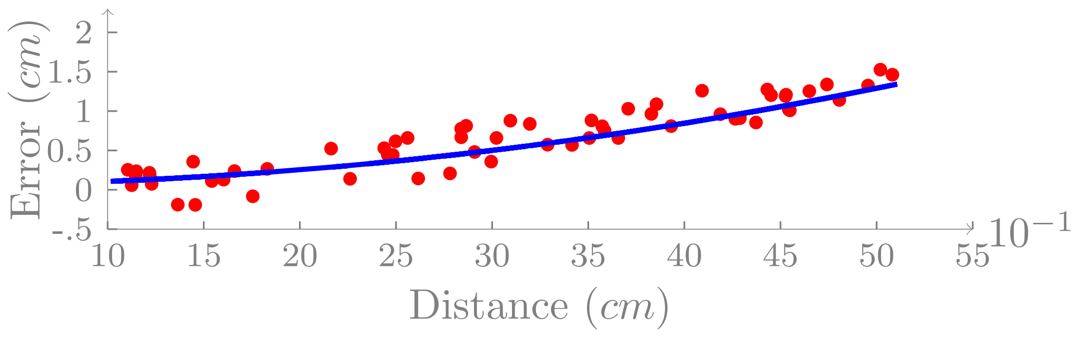
There are two ways of computing the error profiles of the transformation blocks either analytically or numerically (Section 3.3.4). In scopes of this example we conducted a series of experiments to determine the error profiles of each block. The errors that occur in the transformation block responsible for the conversion of the image coordinates into the world coordinates are constant, and mainly occur because of the discretization of space by the camera itself. These errors can be estimated through a regular camera calibration process using a chessboard. The second transformation block describes the detection of the markers in the image plane and the consecutive 3D reconstruction of their poses. In this case the magnitude of the occurring errors depends on two factors: the distance between the marker and the camera center, and the orientation of the marker in reference to the camera center. To compute those errors two experiments were conducted.

7.3.1 *Error Profiles of the "3D Reconstruction" Transformation*

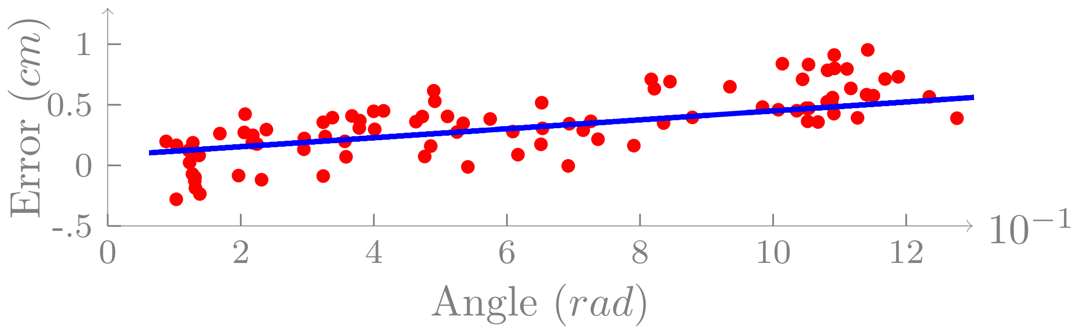
EXPERIMENTAL SETUP : The experimental setup is depicted in Figure 31. We used a high-resolution and high-framerate camera (from now on *Camera*) to record the ground truth. The *Camera* was placed two meters away from the main scene in such a way that it could observe it entirely. In the middle of the scene we placed a cube that had two square markers printed on two of its sides, Figures 7.31(b) and 7.31(c). The positions of the latter in reference to each other are known. During the experiment the cube was static with one of the mentioned markers facing the *Camera*. It is marked with a red rectangle in the image and from now on will be referred to as *Static Marker*. For simplicity we will refer to the second marker as *Cell Marker*. The *Cell Marker* is only visible to the cellphone camera during both of the experiments. To track the motions of the cellphone camera a third marker was statically attached to the cellphone itself in such a way that it was visible in the image plane of the *Camera* during the entire experiment (marked with a blue rectangle). We will be referring to the cellphone camera as *Dynamic Marker*.

ERROR DEPENDENCY TO THE DISTANCE : The first experiment is designed to measure the error dependency to the distance between the target object and the cellphone camera, Figures 7.31(a) and 7.31(b). We begin by holding the cellphone close to the static cube and gradually move it away on a straight line without changing the orientation of the phone. During the entire experiment the cellphone camera is registering the *Cell Marker* in its image plane and computing the 3D distance to it. On the other hand the *Camera* is simultaneously computing the ground truth. The latter is achieved by registering the *Static Marker* and the *Dynamic Marker* in its image plane and computing the 3D distance between the cellphone and the static cube. This is possible since the positions of the cellphone camera center in reference to the *Dynamic Marker* and the position of the *Cell Marker* in reference to the *Static Marker* are known. Note that the distance between the *Static Marker* and the *Camera* remains constant and the change in distance between the *Dynamic Marker* and the *Camera* is small enough to be negligible ($6\,\mathrm{cm}$), while the distance between the cellphone camera and the *Cell Marker* changes from around $10\,\mathrm{cm}$ to $50\,\mathrm{cm}$. Therefore, the above described method can qualify as ground truth registration.

ERROR DEPENDENCY TO THE ANGLE : The second experiment is designed to measure the error dependency to the angle between the target object and the cellphone camera Figures 7.31(a) and 7.31(c). We use the same experimental setup as in the previous experiment. The difference is that this time rather than changing the distance between the cellphone and the *Cell Marker*, we keep it constant and instead



(a)



(b)

Figure 32: 7.32(a) Illustrates the error dependency of the 3D reconstruction algorithm to the distance between the marker and the camera. 7.32(b) Illustrates the error dependency of the 3D reconstruction algorithm to the angle of view. Here the presented angle is the out-of-plane angle between the cellphone and the surface of the marker. Note that the in-plane rotation angels are not presented since they do not add any errors.

change the angle between them. There are three axes along which we can rotate the cellphone. The first is the z axis, this will result into an in-plane rotation of the marker in the image plane of the cellphone camera, and will not be adding any errors. The other two axes: x and y, will result in an out of plane rotation and will cause errors. Similar angles of rotation would have an equivalent error magnitude for both axes due to their homogeneity. Thus, we perform the experiment for one of the mentioned axes and use the same profile for both of them. Note that the rotations registered by the *Camera* are all in-plane rotations, whereas the rotations registered by the cellphone are out of plane rotations. Therefore the described experiment can qualify for ground truth registration.

Figure 32 illustrates the error dependency of the pose reconstruction to the distance and the viewing angle between the camera and the marker. Note that since the error

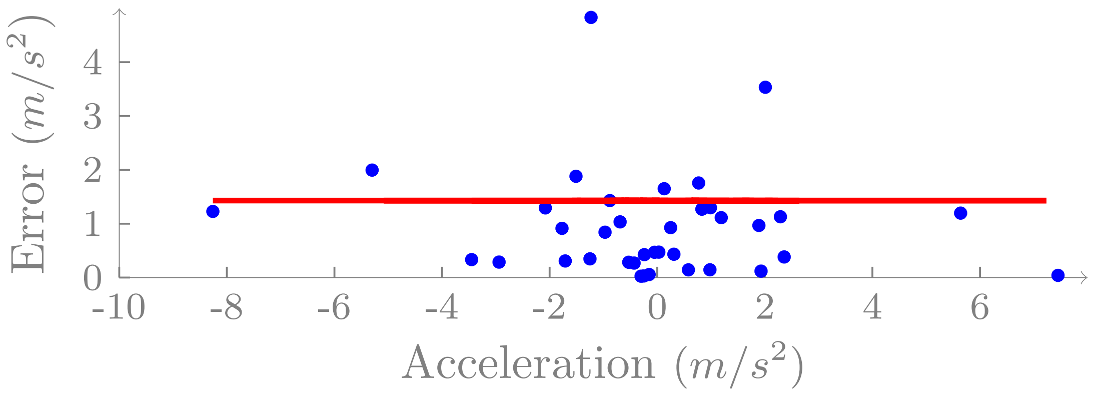


Figure 33: The error profile of the accelerometer that is embedded in the cellphone. Here the red line illustrates the standard deviation of the errors that is $1.43 m/s^2$.

does not depend on in-plane rotations the provided dependency is only for out-of-plane rotations.

7.3.2 *The Error Profile for Velocity and Acceleration Computation*

The third step in our chain (second transformation from the right) is the computation of the velocity (V_i) of the camera. This is done based on the current and previous poses of the camera and the time that elapsed between their registration. The error in this case can be computed analytically using Equation (24). Note that mathematically the computation of the acceleration based on the velocity from the current and previous timestamps is the same as the computation of the velocity based on the position change. Therefore, the same equation is used to compute the error profile of the transformation responsible for computation of acceleration (third transformation from the right).

7.3.3 *Error Profile of the Cellphone Accelerometer*

We constructed the error profile of the accelerometer of the cellphone using an industry grade accelerometer (XSense). We attached the latter to the cellphone and performed random motions, simultaneously measuring the accelerations from both devices. The computed error profile of the accelerometer is illustrated in Figure 33. The red line depicts the standard deviation of the error, which is $1.43\ m/s^2$.

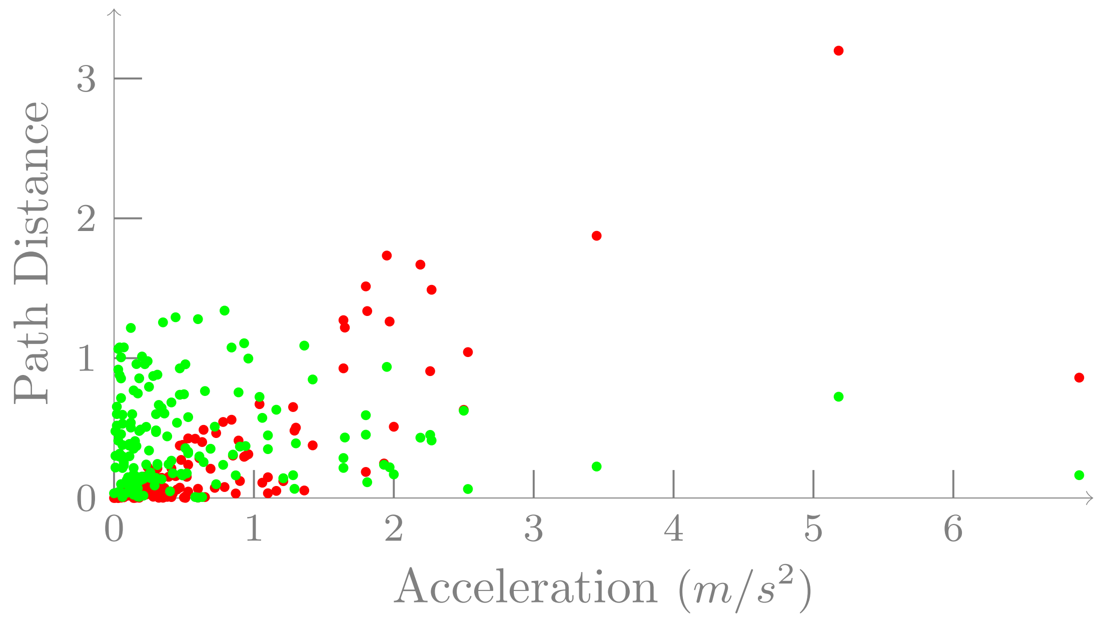
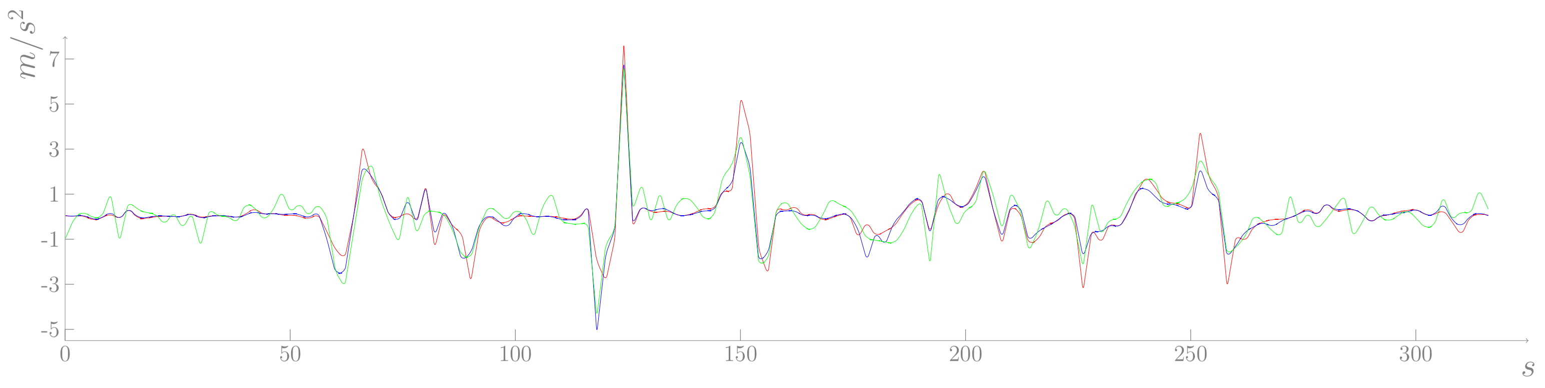


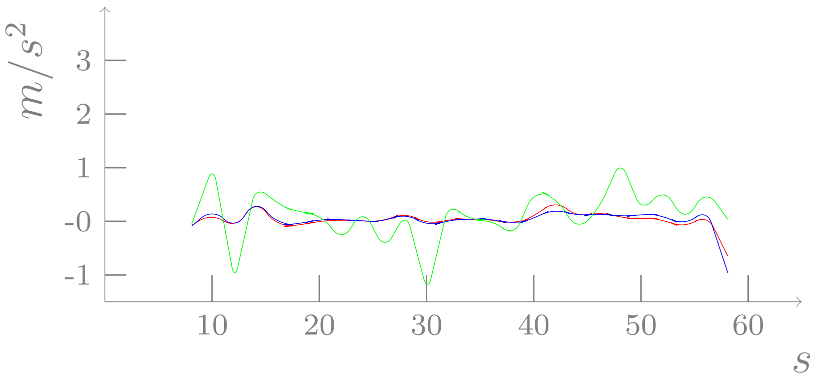
Figure 34: Final analysis for the two acceleration registration methods. The green dots represent path distances of the data registered by the accelerometer located on the cellphone. The red dots represent the path distances of the accelerations computed from the camera images.

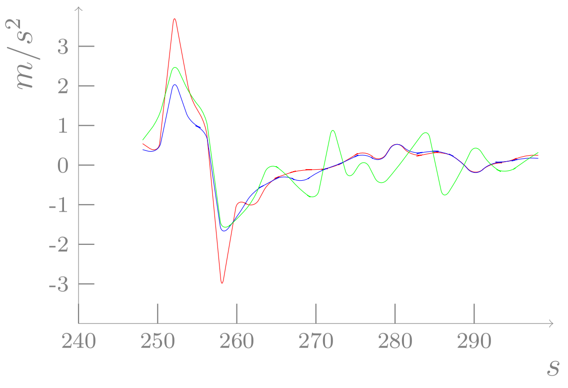
7.4 FINAL RESULT

The results for acceleration measurement for both of the methods plus the ground truth are illustrated in Figure 35. The blue line represents the ground truth, the red line the computed accelerations and the green line the accelerations registered by the cellphone accelerometer. Note that as it was expected for low accelerations, the measured data from the cellphone accelerometer are quite unreliable, while the computed accelerations are closer to the truth, Figure 7.35(b). On the other hand, for the high accelerations the opposite takes place, Figure 7.35(c). The results of path distance analysis are illustrated in Figure 34. The green dots represent the path distances of the accelerometer measurement and the red dots represent the path distances of the reconstruction from the camera. Overall our system was able to pick the most accurate solution in around 75% of the cases. The decisions were mostly inaccurate in places where the path distances from both methods had similar magnitudes.



(a)





(b)

(c)

Figure 35: 7.35(a) Illustration of the registered accelerations. Here the data registered from the industrial accelerometer (XSens) is plotted in blue. The reconstructed accelerations from the cellphone-camera are plotted in red and the accelerations registered from the accelerometer located on the cellphone itself are in green. Note that for lower accelerations the error from the cellphone accelerometer is larger than the error from the camera; however, for large accelerations this changes. To ensure a better observability of the previous statement we provide two zoomed sections 7.35(b), 7.35(c) of the 7.35(a) graph.

8 OPTIMIZATION FOR BOTH MINIMUM ERROR AND COMPLEXITY IN STATIC MODE

8.1 INTRODUCTION AND OVERVIEW

INTRODUCTION : The goal of this chapter is to demonstrate how our framework performs for optimizing both error and complexity in static mode. This is illustrated with an example of a sensor creation. The resulting sensor is a sensitive fingertip (Figure 36) that is capable of estimating the force field applied on its surface as well as the shape of the object that the sensor is interacting with. Our sensor consists of a CCD Camera that performs a marker based 3D reconstruction of the membrane. Following this, the system approximates the membrane by a grid of springs and based on its physical deformations it estimates the acting forces. There are two main ways for estimating the acting forces, Figure 37. The first is using Hook's law (marked by cyan dots), where the system estimates the acting forces on each node of the grid based on the stretches of adjacent springs. The second one is using Newton's second law (marked by red dots), where the acting forces on each node are estimated based on their accelerations and mass. This method is at a disadvantage to the one based on Hook's law since it requires a dynamic system. Nevertheless, we discuss both of the cases and illustrate the nature of the error accumulation, complexity and the reliability of the results for both of them. Based on the most optimal approach we construct the real world sensor and present the data it sensed along side with its error profiles.

OVERVIEW : Initially we will present the *transformation graph* and discuss the nature of path distance computation for both of the chains it contains. Further we will present the physical setup of the sensor, that was constructed based on the optimal chain. This will be followed by the characteristics of the newly constructed sensor.

8.2 TRANSFORMATION GRAPH BASED ANALYSIS

Figure 37 illustrates the *transformation graph* of our setup. Since we are optimizing for minimum error and minimum complexity all of the additive weights are incremented by one. This is done to ensure that the nodes that do not introduce errors still contribute to the global path computation due to their complexity. Since we have only

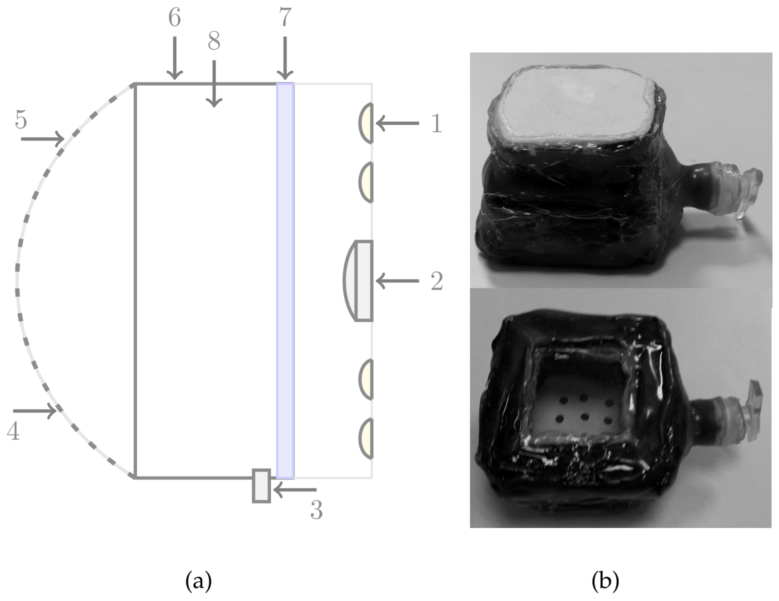


Figure 36: Figure 8.36(a) illustrates the blueprint of the finger: (1) LED, (2) CCD camera, (3) a valve to regulate the amount of air within the frame, (4) rigid circle markers, (5) rubber skin surface, (6) the airtight frame of the finger, (7) glass, (8) air. Figure 8.36(b) illustrates the constructed prototype of the sensitive fingertip sensor.

one initial sensor (CCD camera) there is only one *Start* node that provides data in $[u, v, t]$ dimensions. Where (u, v) are the pixel coordinates on the image plane of the camera, and t is the time of image registration. The chains for both of the methods start with the same two transformations. Those are the conversion from the image coordinate system to the world coordinate system, and the 3D reconstruction of the markers.

ADDITIVE WEIGHTS : Due to the image discretization and bad lighting conditions the 3D reconstruction algorithm operates with 2% relative error, therefore the additive weights of the edges pointing out of this transformation are set to $2 + 1$. As was shown in Section 6.3 the errors for computing derivation can amount up to 8%. Therefore, assuming Gaussian distribution we assign the additive weight of the edges pointing out of transformations that implement numerical derivative to $4 + 1$. Finally, the two transformations responsible for implementing Hook's and Newton's laws operate under the assumption that the elastic membrane is a grid of springs. Which results in around 2% expected errors in force computation. Thus, the additive weights are set to $2 + 1$ for the edges pointing out of them. All of the other transformations whose additive weights are set to 1 do not introduce any errors.

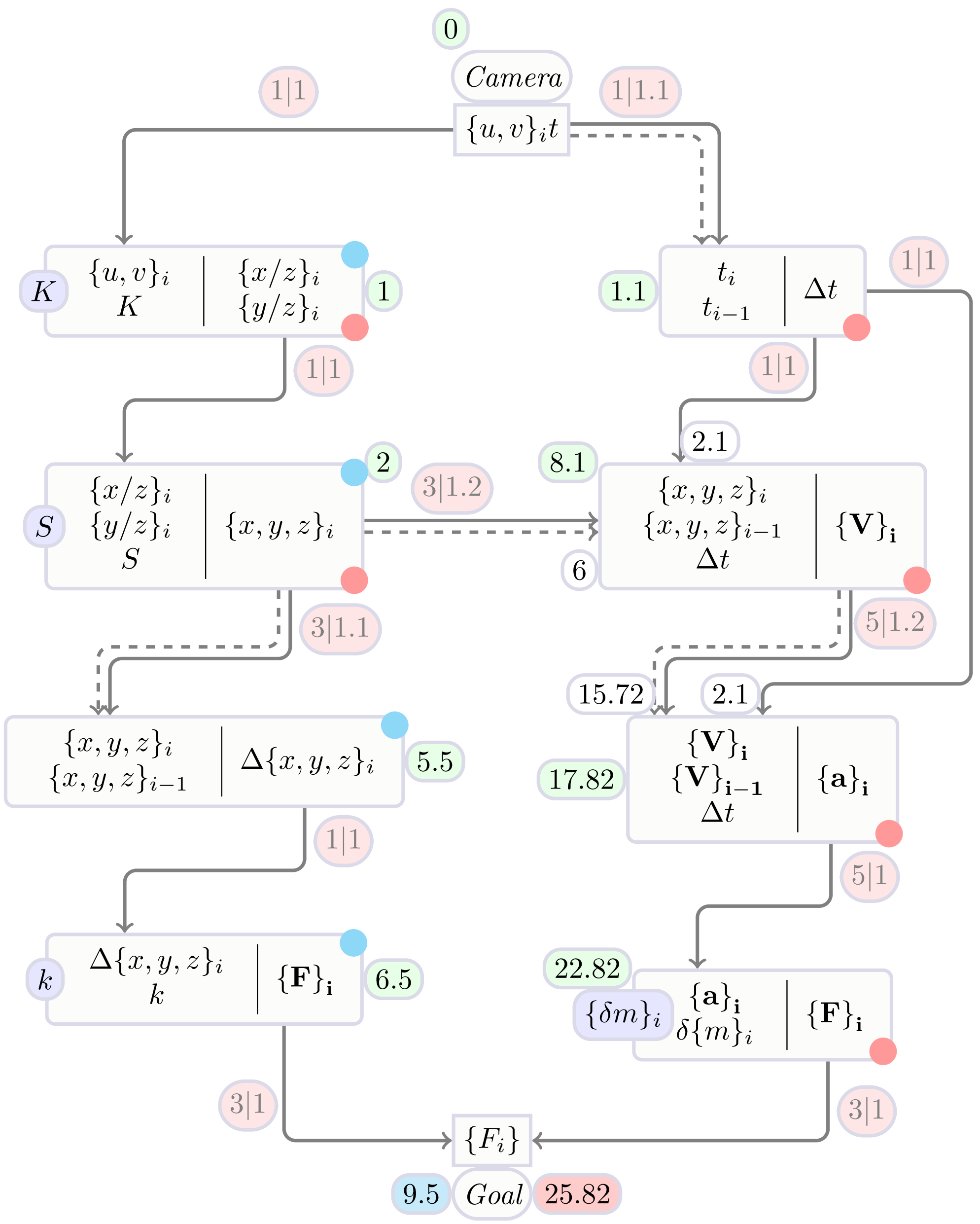

0
Camera
$\{u,v\}_i t$
1|1
1|1.1
K
$\{u,v\}_i$ K
$\{x/z\}_i$ $\{y/z\}_i$
1
1.1
t_i t_{i-1}
Δt
1|1
1|1
1|1
2.1
S
$\{x/z\}_i$ $\{y/z\}_i$ S
$\{x,y,z\}_i$
2
3|1.2
8.1
$\{x,y,z\}_i$ $\{x,y,z\}_{i-1}$ Δt
$\{\mathbf{V}\}_\mathbf{i}$
6
3|1.1
5|1.2
15.72
2.1
$\{x,y,z\}_i$ $\{x,y,z\}_{i-1}$
$\Delta\{x,y,z\}_i$
5.5
17.82
$\{\mathbf{V}\}_\mathbf{i}$ $\{\mathbf{V}\}_\mathbf{i-1}$ Δt
$\{\mathbf{a}\}_\mathbf{i}$
1|1
5|1
k
$\Delta\{x,y,z\}_i$ k
$\{\mathbf{F}\}_\mathbf{i}$
6.5
22.82
$\{\delta m\}_i$
$\{\mathbf{a}\}_\mathbf{i}$ $\delta\{m\}_i$
$\{\mathbf{F}\}_\mathbf{i}$
3|1
3|1
$\{F_i\}$
9.5
Goal
25.82

Figure 37: Transformation graph for a sensitive fingertip sensor. Here the system is set up to sense forces using a regular CCD camera. The graph contains two possible transformation chains. The first chain (marked with cyan dots) contains the set of transformations necessary to sense forces using Hook's law. The second chain (marked with red dots) contains the set of transformations necessary for sensing forces using Newton's second law.

MULTIPLICATIVE WEIGHTS : The first two transformation blocks from the left side are high-order transformation blocks. We decided to discard their high-order nature since both of them belong to both of the possible transformation chains. Thus, their contributions to the complexity are equal for both of the chains and therefore irrelevant. The other four non basic transformation blocks are iteration dependent, out of which two are just responsible for a numerical subtraction and the other two are responsible for numerical derivation. Since the subtraction is a less complex operation the multiplicative weights for those transformations are set to 1.1. The multiplicative weights for the other two transformations are set to 1.2 since they are only at a slight disadvantage to those who implement subtraction.

Note, the final distance to the goal of the chain based on Hook's law is significantly smaller than the second chain. There are a couple of reasons behind it. The large errors that result due to the transformations in the beginning belong to both of the chains. However, the chain describing the method based on Newton's law is longer than the other one. Therefore, the path distance computation in it will amount to a larger value. The second reason is that due to their dynamic nature, the transformations describing the two derivatives responsible for computation of acceleration from the spacial change, result in two iteration dependent transformations in a row. The latter significantly increases the path distance of the chain making it unfeasible to use. On the other hand, the chain describing Hook's law proceeds with two transformations, out of which only one is iteration dependent.

8.3 PHYSICAL IMPLEMENTATION

As a result of the analysis based on the transformation graph we have constructed the sensitive fingertip sensor using the model based on Hook's law. The resulting sensor (Figure 36) is capable of dynamically measuring the force fields over its fingertip with a relatively small error of 0.04N. It can reconstruct the shape of the target object and determine the nature of its deformations by dynamically sensing the changes in its shape.

8.3.1 *Physical Setup*

The design of our multisensor is rather simple. It consists of a thin white rubber membrane that has rigid black circles attached to its surface. The membrane is mounted on a hollow rectangular frame. The other side of the frame is sealed with a glass. The rectangular frame is also equipped with a small valve that allows to control the amount of air within the frame (Figure 8.36(a)). By increasing and decreasing the amount of air within the frame we can regulate the shape of the rubber membrane as well as its sensitivity to deformations caused by external forces. From the other side of the glass (outside of the rectangular frame) a CCD camera is mounted combined with a few LEDs to provide lighting. This is done in such way that it is always possible to keep the membrane in the image plane of the camera.

8.3.2 *3D Reconstruction of the Surface of the Membrane*

The 3D reconstruction of the surface of the membrane consists of several steps. First, we detect all the ellipses in the image and based on their radius and locations in the image we determine the position of the circle centers of the corresponding markers. Second, we compute the 3D coordinates of the markers using previously determined circle centers combined with their real world radius. The latter results into an unstructured point-cloud which is further sorted using simple triangulation and interpolation techniques. Lastly we describe the surface of the membrane using a SPLINE surface fitted to the structured point-cloud obtained from the previous step.

EXTRACTION OF THE CIRCLE CENTERS : It is a well known fact that the projection of a circle is always an ellipse unless the projection surface is parallel to the surface of the circle. Thus, given the image I_j captured from the camera we extract all the black ellipses $\{e_i\} \in I_j$ (Figure 38) according to [40] that correspond to the projections of the circles $\{c_i\}$. Since an ellipse is a conic section, in the most general case it is possible to write the ellipse equation in this form:

$$m_1 x_i^2 + m_2 x_i y_i + m_3 y_{2x_i}^2 + m_4 x_i + m_5 y_i + m_6 = 0 \tag{25}$$

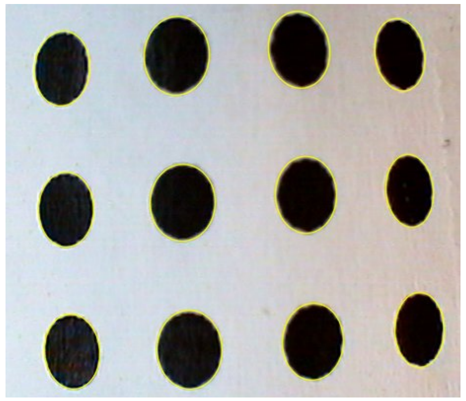
where x_i and y_i are the coordinates of all the points on the considered ellipse. The matrix representation of the conic section can be written as follows:

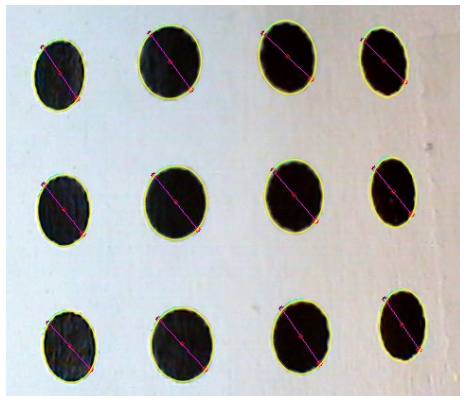
$$M = \begin{pmatrix} m_1 & m_2/2 & m_4/2 \\ m_2/2 & m_3 & m_5/2 \\ m_4/2 & m_5/2 & m_6 \end{pmatrix} \tag{26}$$

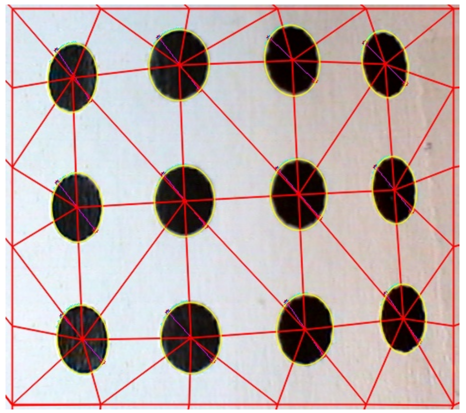
where the matrix elements are the corresponding coefficients from Equation (25). Since M is a symmetric matrix its eigenvector matrix V will be orthogonal and the following will hold:

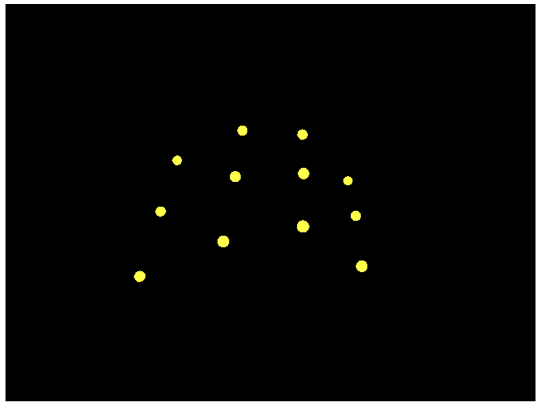
$$M = V * \Lambda * V^T \qquad \Lambda = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \tag{27}$$

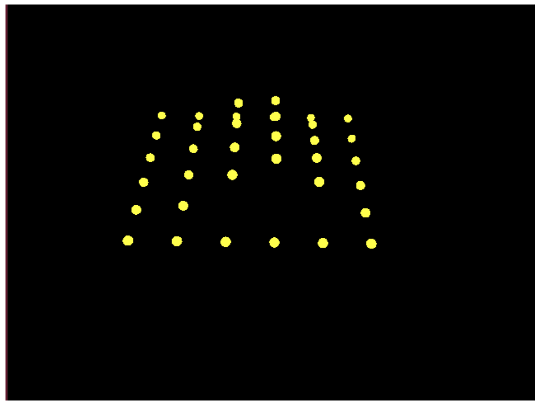
here $\lambda_1 > \lambda_2 > \lambda_3$ are the eigenvalues of M. As it is shown in [18] it is possible to estimate the position $P_i = \{u, v, 1\}^t$ of the projection of the center of the circle c_i in the image plane (Figure 8.38(b)) by:


(a) Ellipse detection


(b) Computation of the circle centers


(c) Triangulation and mashing


(d) Initial 3D reconstruction


(e) Mashing and biliniar triangulation

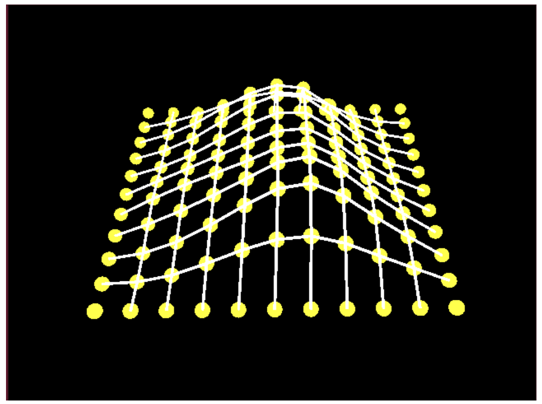

(f) B-SPLINE fitting

Figure 38: Step by step illustration of the membrane surface reconstruction process. The images have been copied from [1]

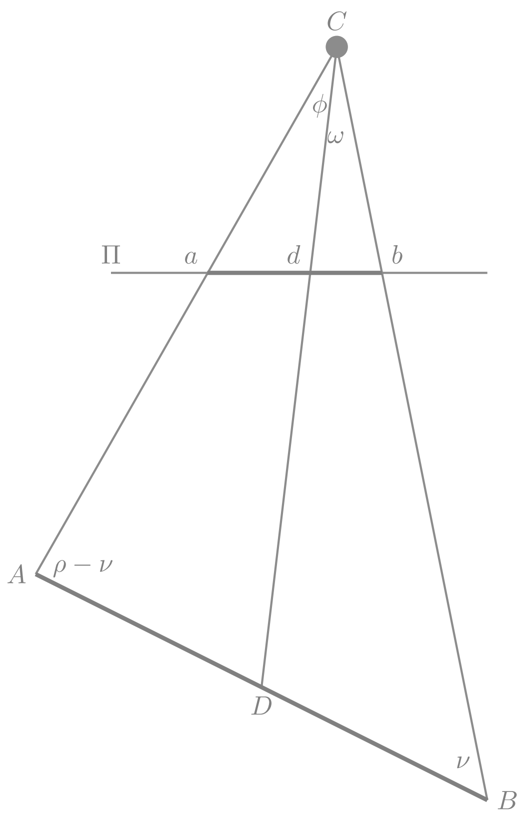


Figure 39: Here C is the camera center, Π is the image plane, a and b are two points on the ellipse, and line ab passes through d. The latter is the projection of circle center D, $AD = DB$. $\rho = 180 - \phi - \omega$

$$P_i = \pm V \begin{pmatrix} -\sqrt{-\lambda_3/\lambda_b egin1} sin\psi \\ 0 \\ \sqrt{-\lambda_1/\lambda_3} cos(\psi) \end{pmatrix} \tag{28}$$

where:

$$\psi = \arccos \sqrt{(\frac{\lambda_2 - \lambda_3}{\lambda_1 - \lambda_3})} \tag{29}$$

RECONSTRUCTION OF THE 3D POSE OF THE CIRCLE : Consider Figure 39, here D is the center of the circle, A and B are two points on the circle such that line AB passes through the center of the circle. This results in $AB = BA = r$, where r is the radius of the circle, Π is the image plain and C is the camera center, a,b, and d are the projections of A, B and D on the image plane correspondingly. Hence, a and b are located on the ellipse, and d is inside (Figure 8.38(b)). We are interested in computing the length of CD to determine the 3D pose of the circle. This can be computed by applying the sine rule:

$$\frac{\sin(\phi)}{r} = \frac{\sin(\rho - \nu)}{\overline{CD}} \tag{30}$$

$$\frac{\sin(\omega)}{r} = \frac{\sin(\nu)}{\overline{CD}} \tag{31}$$

where $\rho = 180 - \phi - \omega$. By dividing Equation (30) by Equation (31) we obtain:

$$\frac{\sin(\phi)}{\sin \sin(\omega)} = \frac{\sin(\rho - \nu)}{\sin \nu} \tag{32}$$

By simplifying Equation (32) we can derive ν to:

$$\cot \nu = \frac{\sin(\phi)}{\sin(\omega)\sin(\rho)} + \cot(\rho) \tag{33}$$

Further we combine Equation (33) and Equation (31) to derive equation for CD:

$$\overline{CD} = \frac{r * \sin(\nu)}{\sin(\omega)} \tag{34}$$

Here the remaining unknowns are the angles ω and ϕ that can be computed from $(\hat{\mathbf{Ca}} \cdot \hat{\mathbf{Cd}})$ and $(\hat{\mathbf{Cd}} \cdot \hat{\mathbf{Cb}})$ dot products correspondingly, where "$\hat{}$" indicates a unit vector. a can be chosen as an arbitrary point on an ellipse, where b will be defined as a point that is located on the intersection of the ellipse and a line that passes through a and d. For best accuracy we suggest to pick the line ab such that it is parallel to the major axis of the ellipse. After the depth $\overline{CD}$ of the center of the circle is known it is possible to re-project the point into 3D using the inverse of the intrinsic camera matrix and obtain the unstructured point cloud $\{p_i\}$ Figure 8.38(d).

MASHING AND SURFACE FITTING : Only the point cloud $\{p_i\}$ is not sufficient to extract information regarding the contact area of the sensor. Thus, as a next step we perform B-spline surface fitting. The point cloud $\{p_i\}$ is not structured and hence it requires some processing for the surface fitting. We consider the set $\{p_i\}$ of projected circle centers in the image plain and the rectangular profile of the frame which will be later used as boundary conditions. We assume that the pose of the rectangular profile in reference to the camera center is known. The former can be obtained by running the 3D reconstruction described in the previous chapter with an open valve. First, we obtain triangulation $\{t_i\} \in T$ by performing Delaunay triangulation for $\{p_i\}$ (Figure 8.38(c)). Next, we generate a 2D low resolution Cartesian grid within the frame. For each node $v_i \in t_i$ of that grid we compute its 3D pose by performing barycentric interpolation in t_i using the frame and the points located on the frame as boundary conditions (Figure 8.38(e)).

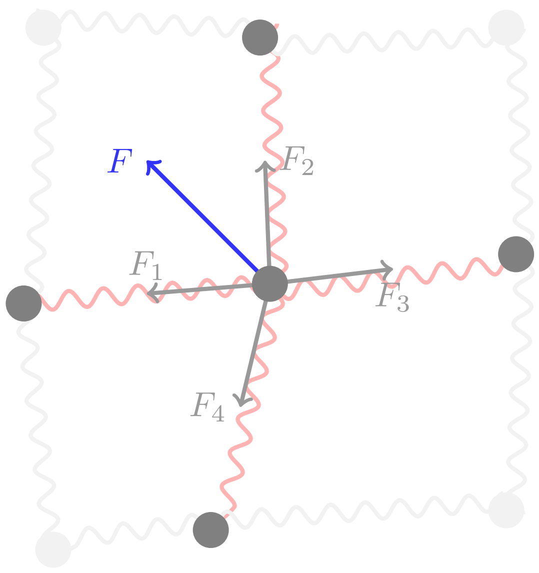


Figure 40: Computation of the force acting on a node. Here $\mathbf{F} = \mathbf{F}_1 + \mathbf{F}_2 + \mathbf{F}_3 + \mathbf{F}_4$ is the overall force acting on the considered node.

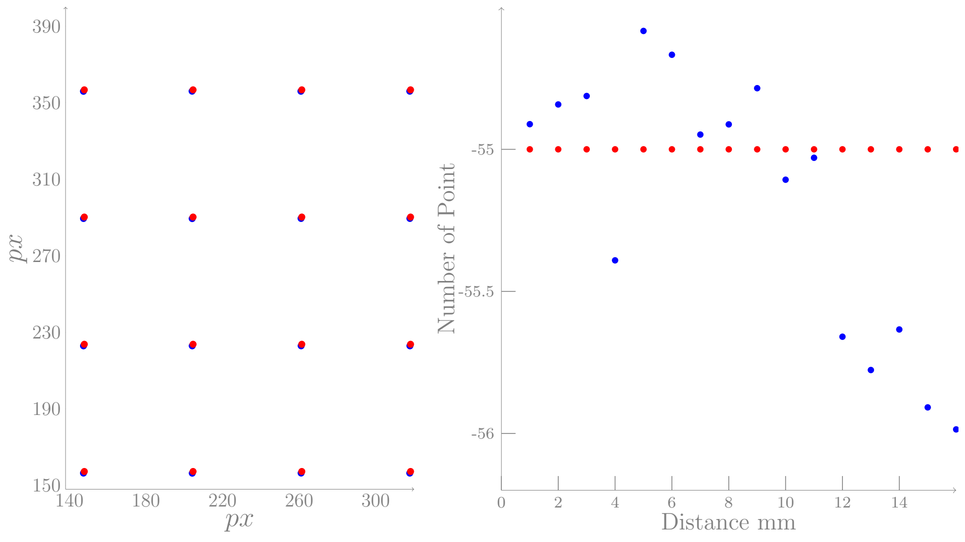
8.3.3 *Computation of Force Distribution*

In the final step we fit a B-spline surface to the 3D structured point cloud obtained from the grid (Figure 8.38(f)).

Due to its molecular structure the rubber is not a Hookean material, i.e. the relation between the stretch and the applied force is not linear [15], hence it is not possible to analytically compute the force distribution. To compute the relative force distribution over the surface of the membrane we make some assumptions. First we assume that the membrane has a uniform thickness (d). The thickness is considerably smaller then the width (w) and the length (l) of the membrane ($w \gg d$, $l \gg d$). The thinning of the thickness due to stretches is neglectful and the external forces are small, i.e. in the range of $[0.5 - 2]$N. Based on this assumptions we claim that the elasticity of the rubber membrane is constant within a small stretch range (proof by experiment Section 8.4). If the volume of air remains constant within the frame of the sensor the following is true:

$$\Delta F = c \Delta A \tag{35}$$

where c is a constant, ΔF is the overall change in the external forces acting on the membrane and ΔA is the change of the area of the membrane due to ΔF. From (35) and the above made assumptions, it follows that the membrane is stretching uniformly. Therefore to compute the direction and magnitude of forces over the entire surface we can approximate the membrane by a grid of springs (Figure 40). Since



(a) Circle center detection error in the image plane (b) Error of the initial 3D reconstruction

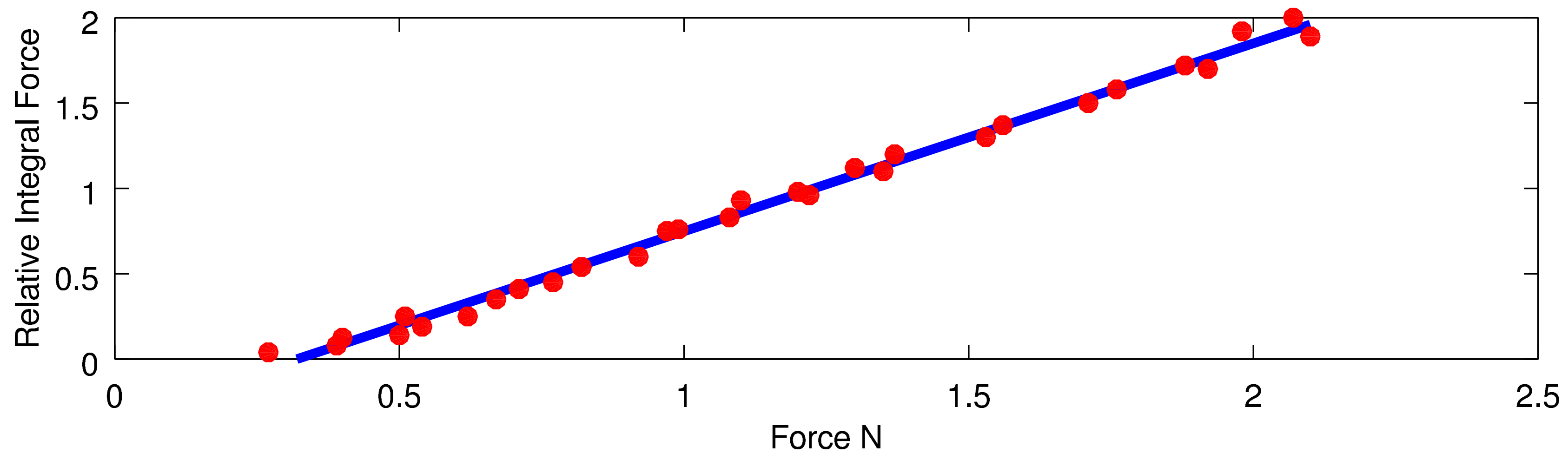
Figure 41: Image 8.41(a) illustrates the locations of estimated (blue dots) and known (red dots) circle centers. Image 8.41(b) illustrates the computed (blue dots) and known (red dots) absolute distances of the circles to the camera center.

the surface is stretching uniformly, each node of the grid will be equidistant to its neighbor. Thus the direction and magnitude of the force at that node will only depend on the curvature of the surface at that point (Figure 40).

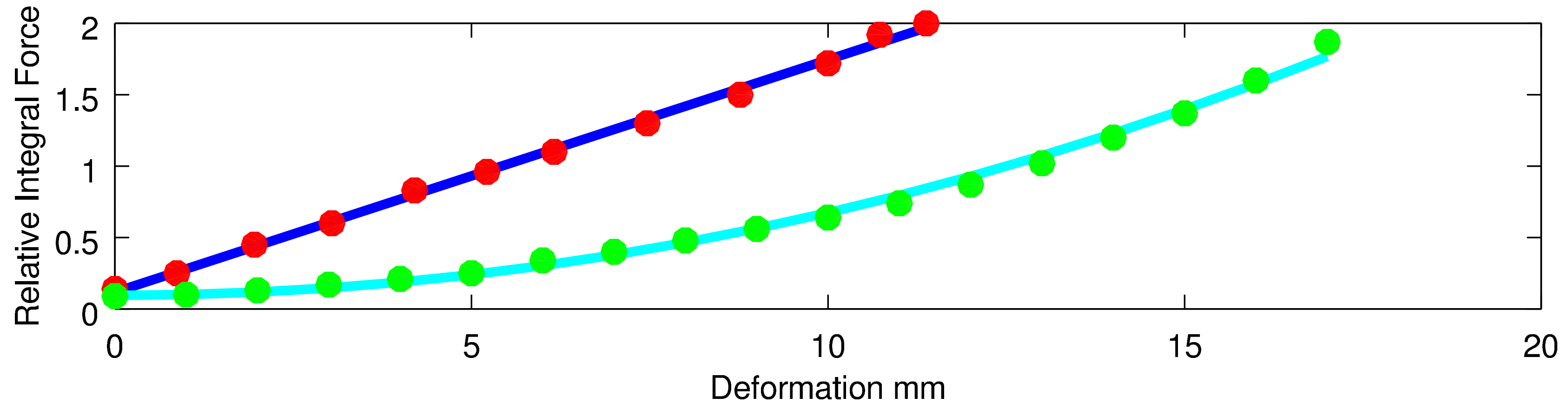
8.4 EXPERIMENTAL RESULTS

EVALUATION OF THE 3D RECONSTRUCTION : The 3D reconstruction of the elastic membrane is computed based on the projected ellipsoidal silhouettes of the circle markers on the image plane of the camera. As mentioned above in Section 8.3.2 the algorithm consists of several steps. Initially, a detection of all the ellipsoidal shapes in the camera image is performed. The latter is followed by computation of the location of the circle center projections, which is used to obtain an unstructured 3D point-cloud that is further sorted using simple triangulation and interpolation techniques. Lastly the elastic membrane is represented by a B-Spline surface that is obtained from the structured point-cloud. Examples of the final 3D reconstructions are presented for qualitative evaluation in Figure 43.

EVALUATION OF THE CIRCLE CENTER DETECTION AND INITIAL RECONSTRUCTION : To evaluate the circle center detection part of our algorithm we generated



(a) Force/Relative Force Profile for a solid object



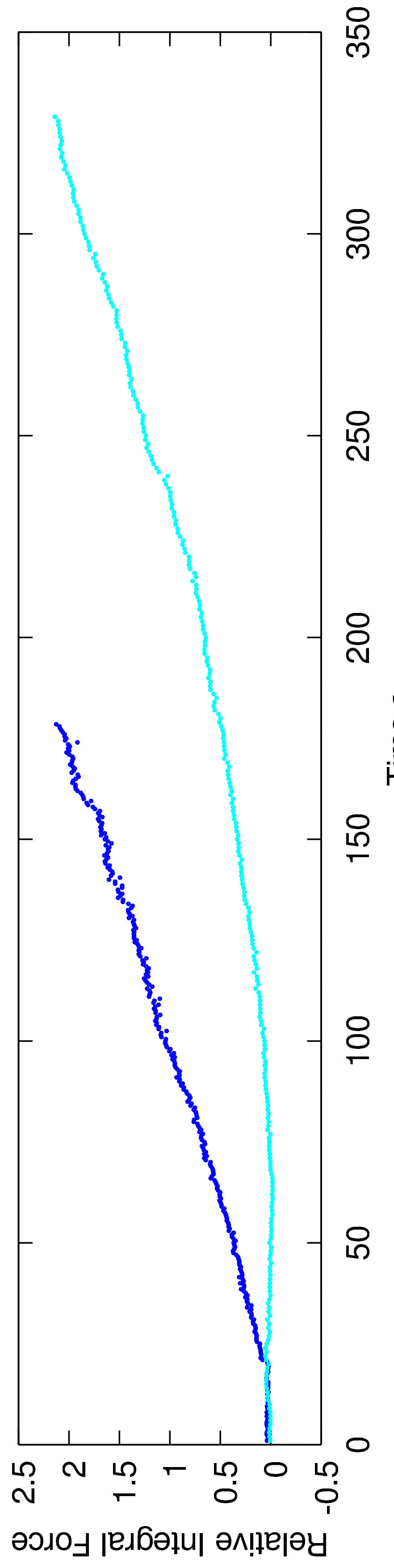
(b) Force/Deformation profile for soft (cyan) and solid (blue) objects

Figure 42: 8.42(a) Illustrates the relation of the computed relative integral force to measured force. Note that it is clearly visible that the relation between them is linear. Thus, the applied Hook's model within the scopes of this experiment holds.

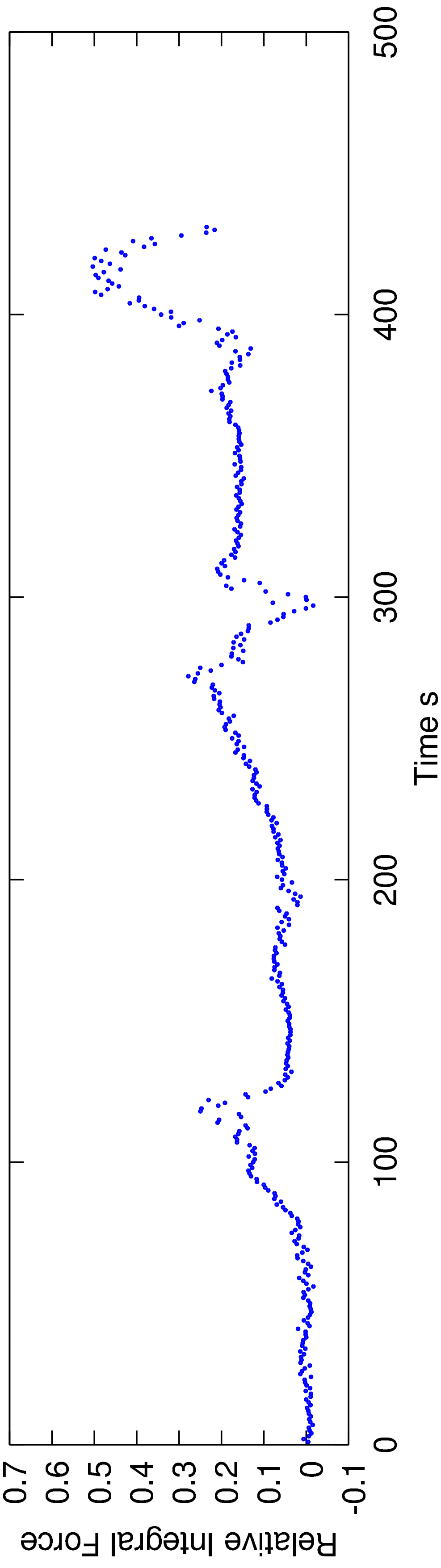
15 synthetic images. Each of the images contains 16 ellipses that represent the projections of the circle markers with known and varying poses and orientations. Further we have processed the mentioned images using the algorithm described in Section 8.3.2. The results are illustrated in Figure 8.41(a), where the red dotes illustrate the locations of estimated and the blue dots illustrate the locations of the known circle centers. As a consequence we achieved sub-pixel accuracy for circle center detection, meaning that the average error was approximately 0.5 px. Further, we used the above mentioned generated images and reconstructed the 3D positions of the circular markers. The latter is performed to evaluate the accuracy of the initial 3D reconstruction algorithm. The results from one of the images are depicted in Figure 8.41(b). Here the "x" axis represents the number of the markers, and the "y" axis the absolute distance to the camera center in mm. The standard deviation of the reconstructed points was computed to be approximately 0.39 mm.

EVALUATION OF FORCE COMPUTATION : Since the rubber is not a Hookean material the force to strain relation is generally nonlinear. Thus, it is not possible to analytically compute force from strain. However, we claim that for small strains the Hook's model holds. To support this we conducted an experiment where we attached a force sensor to our sensor and drove a cylindrical solid object into the rubber membrane with 1 mm increments. For each increment we registered the force from the force sensor and the estimated integral force acting on the membrane using our assumption of linearity. Figure 8.42(a) depicts the relationship between the measured force and the computed relative integral force. Note that the relation between the two forces is linear. Thus, the assumption of force/deformation linearity holds. Further it is possible to compute the strain coefficient by fitting a line to the gathered data. The overall average error from this experiment is equal to 0.04 N. To further reinforce the feasibility of our assumption we conducted the same experiment using two different materials (a solid cylinder and a soft sponge). The resulting force to deformation graph is illustrated in Figure 8.42(b). Here the blue line represents the solid cylinder and as was expected the relation of force to deformation is linear. The cyan line represents the results from the experiment with the soft sponge. Note, since the sponge has stiffness of itself the force/deformation graph is not a line anymore.

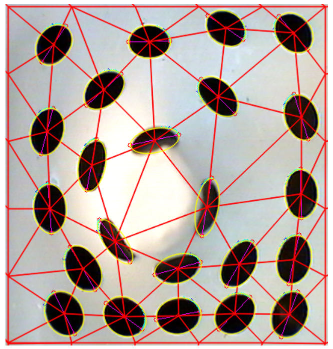
DETERMINATION OF THE MATERIAL TYPE : To determine the type of the material the sensor is interacting with, we performed a series of similar experiments to the ones described above. The difference is that this time the time to relative force profile was registered. Experiments were performed for 3 types of objects: solid, soft and amorphous. Soft and solid target objects were driven into the sensor continually. The results are illustrated in Figure 8.43(a). Note that from the shape of the profile it is

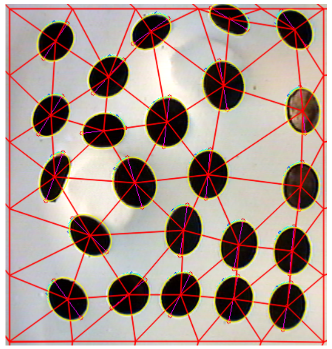


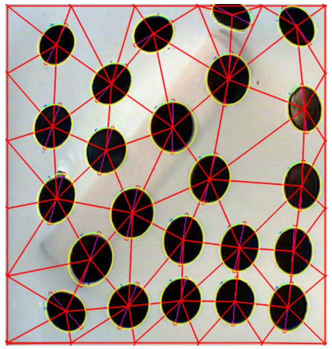
(a) Force/Time profile for soft (cyan) and solid (blue) objects

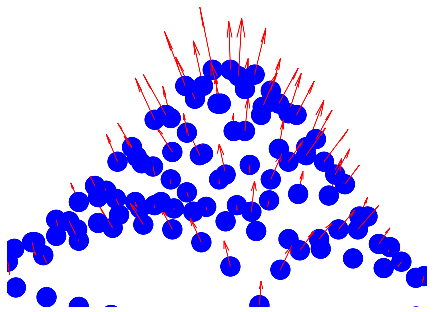


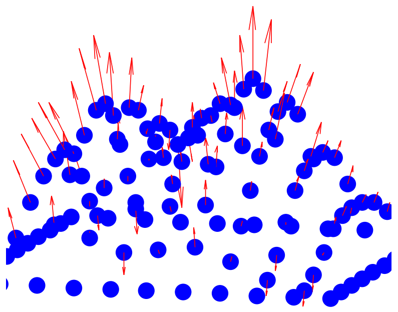
(b) Force/Time profile for and amorphus object easy to determine that the blue line represents the profile for the solid and the cyan line for the soft objects. In case of the amorphous object, the target was iteratively driven continually into the sensor and than left to relax. The results are illustrated in Figure 8.43(b). Note that in the section of the curve where the target was continually driven the force is increasing. However, in sections where the system was left to relax the force is decreasing. This is due to the fact that when the system is in the relaxation stage the amorphous object is deforming under the forces acting from the membrane itself and trying to come to a stage of a minimum energy.


(c)


(d)


(e)


(f)


(g)

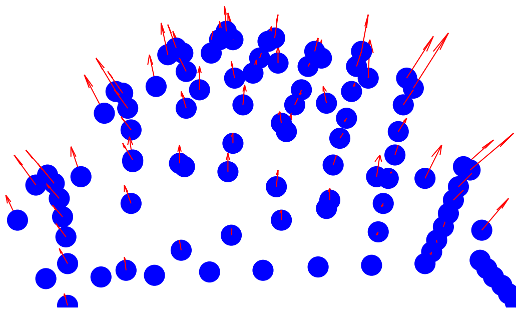

(h)

Figure 43: Figures 8.43(c), 8.43(d) and 8.43(e) illustrate the captured and processed images from the CCD camera for the three different objects, and Figures 8.43(f), 8.43(g) and 8.43(h) illustrate the respective 3D reconstructions and force distributions. Here the direction of the force vectors are inverted to ensure better visibility. The pictures are taken from [1]

9 CONCLUSION AND FUTURE WORK

9.1 CONCLUSION

In this thesis, we described a framework for optimal dynamic modeling of sensing modalities and sensory substitution. Sensory substitution is widely understood as the substitution of one sensing modality by another for humans or animals with impairments. Albeit in its general definition any conversion of one sensory modality to another through means of mathematical or physical transformations is a substitution of the former. This involves all the basic physical sensors that digitize such signals as temperature, acceleration, torque, etc. as well as more complex virtual sensors that extend the range of sensing modalities of basic sensors through mathematical and physical transformations. Due to its nature, sensory substitution is one of the most widely used topics in modern sciences. Although lately there have been major advancements in sensory substitution there are still several questions that remain open.

Firstly, sensory substitution is a multidisciplinary topic that is currently being treated by each discipline within its narrow bounds. This results in the problems of choosing correct connecting interfaces between separate disciplines. We have presented a framework for optimal dynamic modeling of sensing modalities and sensory substitution. Our framework is capable of dynamically establishing optimal connections between different sensory modalities through a set of physical and mathematical transformations. The latter are picked from a "bag of transformations" that contains known transformations that occur in multiple disciplines. Therefore, it is the inherit nature of our framework to treat the target problem in scopes of all the relevant disciplines.

Second, it is the automation of the construction process for virtual sensors. Since the construction of virtual sensors oftentimes has more than one solution, there is the major issue of selecting the optimal solution for the current problem. Depending on the problem definition the precision of the sensed data in the target modality could be of high priority; thus, the optimization needs to be done for minimum error. Equally, the speed of obtaining data in the target modality could be of high priority; thus, the optimization needs to be done for minimum complexity. Finally, some problems might require precise data in small time intervals; thus, the optimization needs to strike a balance between minimum error and minimum complexity. The main focus of our framework is the selection of the most optimal chain of mathematical and physical transformations between the initial and goal modalities. The latter allows the creation of virtual sensors that are able to sense in modalities that the original sensors were not designed to operate in. The selection of the set of transformations takes into account the problem specifications and makes automated decisions not only based on the amount of transformations but also on the expected errors that occur due to the transformation and the complexity of the entire transformation chain.

Third, complex multi-sensor platforms generally have more than one way of obtaining the target modality. The quality of the obtained data through a certain chain of transformations depends on the operation range of the initial physical sensors. Thus, a sudden change of the operation range will result in the increase of data quality through one transformation chain and decrease through another one. Hence, is the problem of dynamically adapting the system to current conditions. Our framework is capable of dealing with such issues when operating in dynamic mode which allows a dynamic reconfiguration of the transformation chains to adapt to the current operating ranges of the initial basic sensors.

We have experimentally demonstrated that the error estimates obtained through our framework relate quite closely to the real world errors as well as that our framework is capable of selecting the best measurement in cases where there is more than one initial sensor available. It selected the best possible solution in 75% of the cases and struggled only in cases where both of the possible solutions had very close values. Finally, we have shown how the framework can be used in the construction of new sensors on an example of the construction of a sensitive fingertip force sensor.

9.2 FUTURE WORK

Even though our framework is capable of successfully solving many issues associated with sensory substitution there are still several improvements that could be made. While the *Goal* nodes are completely satisfactory for modeling virtual sensors they struggle in representing real world input interfaces. The issue of such interfaces is that they not only require precise, robust information flow in a particular modality, but also impose limitations on the bandwidth of the information. An example of such interfaces is described in the Section 1.3, in problems dealing with visual sensory signal compensation. Thus an introduction of an additional weight unit that represents the bandwidth of the information is apparent. However, this will lead to additional compromises for the cases where all three parameters need to be minimized due to low correlation between them.

As shown in the experimental section, the framework is capable of successfully optimizing for minimum errors and minimum complexity. It can also strike a balance between optimizing for minimum errors and minimum complexity at the same time. However, in this case neither the errors are minimum, nor is the complexity minimum. The reason behind this is that the complexity and errors do not have a strong correlation between each other. Thus, our approach, which tries to treat them within the scope of a single path computation equation, must find a compromise between them. Therefore a decoupling of global path computation would lead to a better flexibility in prioritizing between minimum errors and minimum complexity. This would also ease the introduction of additional optimization parameters such as the ones representing bandwidth.

Our system is capable of operating in two modes, dynamic and static. Both have their advantages and disadvantages. Dynamic mode computes the edge-weights of each transformation during every iteration based on the current value of the considered argument. This means that it puts the emphasis on error minimization at a cost of execution time and makes most sense when used in cases where the execution time is not as important as the precision of the results. Static mode computes the edge-weights only once at the beginning based on the expected values of the arguments provided by the initial sensors. This results in the static mode being significantly faster than the dynamic mode at the cost of accuracy. Thus, it makes sense to use it for cases where the execution time is most valuable. Naturally, both of these cases can be used for cases where the optimization must be done for minimum error and minimum complexity. However, in static mode the error minimization would be given a smaller priority, and in dynamic mode the execution time would be given a smaller priority. Thus, an introduction of an additional operation mode that is tuned for the case of minimizing both complexity and errors would contribute to better results in such a case. There are several ways of implementing the additional mode. One can be to split the entire operation range into segments and only recompute the edge-weights once the value of the initial sensor arguments switches from one segment into other. The second would be to initially compute the edge-weights of the transformation based on the first value of the argument and recompute them once the difference between the current and first values is larger than a certain threshold.

There are many more improvements that can be made to better the framework and after their implementation many more would be left. We would like to conclude with words of *Leonardo da Vinci*:

"Art is never finished, only abandoned"

PUBLICATIONS

[1] Artashes Mkhitaryan and Darius Burschka. Vision-based haptic multisensor for manipulation of soft, fragile objects. In *IEEE SENSORS*, 2012.

[2] Artashes Mkhitaryan and Darius Burschka. Rgb-d sensor data correction and enhancement by introduction of an additional rgb view. In *IEEE IROS*, 2013.

[3] Artashes Mkhitaryan and Darius Burschka. Visual estimation of object density distribution through observation of its impulse response. In *VISAPP*, 2013.

[4] Artashes Mkhitaryan and Darius Burschka. A framework for dynamic sensory substitution. In *IEEE IROS*, 2014.

[5] Artashes Mkhitaryan and Darius Burschka. 3d reconstruction of dynamic scenes from two asynchronous video-streams. In *VISAPP*, 2014.

[6] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge, 2009.

[7] Schert A. B., Friedel P., and Hemman J. L. Snake's prespective on hear: Reconstruction of input using an imperfect detection system. *Phys*, 2006.

[8] A. Bicchi, E. P. Scilingo, and D. De Rossi. Haptic discrimination 70 of softness in teleoperation: The role of the contact area spread rate. In *IEEE Transactions on Robotics and Automation*, 2010.

[9] E. Bruel-Jungerman, S. Davis, and S. Laroche. Brain plasticity mechanisms and memory: A party of four. *The Neuro scientist*, 2007 October.

[10] Chorley C., Melhuish C., Pipe T., and Rossiter J. Development of a tactile sensor based on biologically inspired edge. In *International Conference on Advanced Robotics ICAR.*, 2009.

[11] G. Cannata, M. Maggiali, G. Metta, and G. Sandini. An embedded artificial skin for humanoid robots. In *Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on*, 2008.

[12] Damian D.D, Martinez H., Dermitzakis K., and Hermandez-Arieta A. Artificial ridged skin for slippage speed detection in prosthetic hand applications. In *Artificial Ridged Skin for Slippage Speed Detection in Prosthetic Hand Applications*, 2010.

[13] A. P Dempster. Upper and lower probabilities induced by a multivalued mapping. *The Annals of Mathematical Statistics*, 1967.

[14] E. W. Dijkstra. "a note on two problems in connexion with graphs". *Numerische Mathematik 1: 269 271*, (1959).

[15] Merritt D.R. and Weinhaus F. The pressure curve for a rubber balloon. *American Journal of Physics*, 1987.

[16] H. Kajimoto et al. optimal design method for selective nerve stimulation and its application to electrocutaneous display. *Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2002.

[17] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae*, August 26, 1735.

[18] Philip Johan. An algorithm for determining the position of a circle in 3d from its perspective 2d projection. Technical report, KTH, Stockholm, 1997.

[19] Franosch J.P., Sobtka M. C., Elepfandt A., and Hemman J. L. Minimal model of pray localization through the lateral-line systems. *PhysRev Letters*, 2003.

[20] Rauschecker J.P. Compensatory plasticity and sensory substitution in the cerebral cortex. *TINS, ELSAVIER*, 1995.

[21] Kamiyama K., Kajimoto H., Inami M., Kawakami N., and Tachi S. A vision-based tactile sensor. In *ICAT*, 2001.

[22] Kamiyama K., Vlack K., Mizota T., Kajimoto H., Kawakami N., and Tachi S. Vision-based sensor for real-time measuring of surface traction fields. *IEEE Computer Graphics and Applications*, 2005.

[23] Sato K., Kamiyama K., Kawakami N., and Tachi S. Finger-shaped gelforce: Sensor for measuring surface traction fields for robotic hand. In *IEEE Transactions on Haptics*, volume 3, 2010.

[24] P. Bach-y-Rita K. Kaczmarek. Tactile displays. *W. Barfield, T. Furness-IIIrd (Eds.), Advanced Interface Design and Virtual Environments, Oxford University Press*, 1995.

[25] S.J. Haase K.A. Kaczmarek. Pattern identification as a function of stimulation current on a fingertip-scanned electrotactile display. *IEEE Trans. Neural Syst. Rehabil.*, 2003.

[26] S.J. Haase K.A. Kaczmarek. Pattern identification and perceived stimulus quality as a function of stimulation current on a fingertip-scanned electrotactile display. *IEEE Trans. Neural Syst. Rehabil.*, 2003.

[27] K.A. Kaczmarek. Electrotactile adaptation on the abdomen: preliminary results. *IEEE Trans. Rehabil*, 2000.

[28] H. et al Kajimoto. Smarttouch augmentation of skin sensation with electrocutaneous display. *Proceedings of the 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, IEEE, pp. 40 46.*, 2003.

[29] R.E. Kalman and R.S. Bucy. New results in linear filtering and prediction theory. *Journal of Basic Engineering*, March 1961.

[30] E. Kruppa. Zur ermittlung eines objektes aus zwei perspektiven mit innerer orientierung. *Sitz.-Ber.Akad.Wiss., Wien*, 1913.

[31] Johnson L.A. and Higgins C.H. A navigation aid for the blind using tactile-visual sensory substitution. In *Proceedings of the 28th IEEE EMBS Annual International Conferece*, 2006.

[32] Mason and J. Samuel. "feedback theory - some properties of signal flow graphs". *Proceedings of the IRE*, 1953.

[33] Warren McCulloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 1943.

[34] N. Molton, S. Se, J.M. Brady, D. Lee, and P. Probert. A stereo vision-based aid for the visually impaired. *Image and Vision Computing*, 1998.

[35] M. E. J Newman. *"The structure and function of complex networks"*. PhD thesis, Department of Physics, University of Michigan., 2004.

[36] Friedel P., Young B. A., and Hemman L. Auditory localization of ground-borne vibrations in snakes. *PhysRev Letters*, 2008.

[37] J. Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. *Proceedings of the 7th Conference of the Cognitive Science Society,*, 1985.

[38] Petri. Communication with automata. Technical report, DTIC, 1966.

[39] Dirk Riehle. *Framework Design A Role Modeling Approach*. PhD thesis, Swiss Federal Institute of Technology Zurich, 2000.

[40] Suzuki S. and Abe K. Topological structural analysis of digitized binary images by border following. *CVGIP*, 1985.

[41] Windsor S. *Hydrodynamic imaging by blind Mexican cave fish*. PhD thesis, University of Auckland, 2008.

[42] M. M. Tai. A mathematical model for the determination of total area under glucose tolerance and other metabolic curves. *Diabetes*, 1994.

[43] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: A factorization approach. *nternational Journal of Computer Vision*, 1992.

[44] Web, 2015. URL `http://eigen.tuxfamily.org/index.php?title=Main_Page`.

[45] Web, 2015. URL `http://www.ode.org/`.

[46] Web, 2015. URL `http://www.cs.jhu.edu/CIPS/xvision/`.

[47] Web, 2015. URL `https://www-s.acm.illinois.edu/webmonkeys/book/c_guide/index.html`.

[48] Web, 2015. URL http://opencv.org/.

[49] Web, 2015. URL http://www.pymunk.org/en/latest/.

[50] Paul Bach y Rita. *Brain Mechanisms in Sensory Substitution*. Academic Press, 1972.

[51] Bach y Rita P. and Kercel W.S. Sensory substitution human-machine interfaces. *TRENDS in Cognitive Science, ELSAVIER*, 2013.