Karlsruhe Institute of Technology

Carnegie Mellon University

Learning and Recognizing Activity Patterns with Sensor Data

Bachelor's Thesis of

Robin Schmucker

at the Department of Informatics
Institute for Anthropomatics and Robotics (IAR)

Reviewer:	Prof. Dr.-Ing. Rüdiger Dillmann (KIT)
Second reviewer:	Prof. Dr.-Ing. Tamim Asfour (KIT)
Advisor:	Prof. Dr. Manuela Veloso (CMU)

01. December 2017 – 31. March 2018

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 27.03.2018

. . . . . .

(Robin Schmucker)

Abstract

Robots provide rich proprioceptive sensor data. By recognizing patterns in streams of low-level sensor readings, a robot can gain awareness about the activities that are performed by its physical body. Research in Human Activity Recognition (HAR) has been thriving in recent years mainly because of the widespread use of wearable sensors such as smartphones and activity trackers. By introducing HAR approaches to the robotics domain, we aim at creating agents that have an awareness of their own physical activities. We propose an online activity recognition pipeline that allows a robot to classify its current action by analyzing heterogenous, asynchronous streams of sensor readings. The approach is evaluated by recognizing different motions of the service robot Pepper and the humanoid robot Nao. Our experiments suggest that a multimodal approach to robot activity recognition can generate more accurate classifications than a unimodal model. We also show that our approach can detect deviations from expected activity execution. Through its generality, the recognition pipeline is transferable to other robots with comparable sensing capabilities.

Zusammenfassung

Roboter stellen eine Menge von propriozeptiven Sensordaten zur Verfügung. Durch das Erkennen von Mustern in Strömen von Sensordaten kann ein Roboter Einsicht in die Aktivitäten, die sein physischer Körper ausführt, erlangen. Getrieben von der großen Verbreitung von mit Sensorik ausgestatten, tragbaren Geräten wie Smartphones und Activity Trackern hat das Forschungsgebiet der Menschlichen Aktivitätserkennung in den letzten Jahren große Fortschritte gemacht. Durch den Transfer von Ansätzen aus dem Gebiet der Menschlichen Aktivitätserkennung in den Bereich der Robotik möchten wir Agenten kreieren, die sich ihrer physischen Aktivitäten bewusst sind. Wir stellen eine Pipeline für Online-Aktivitätserkennung vor, die es einem Roboter erlaubt, auf Basis von heterogenen, asynchronen Strömen von Sensordaten seine aktuelle Aktivität zu klassifizieren. Durch das Erkennen verschiedener Bewegungen des Serviceroboters Pepper und des humanoiden Roboters Nao wird der Ansatz evaluiert. Unsere Experimente legen nahe, dass multimodale Ansätze für die Roboteraktivitätserkennung genauere Klassifikationen als unimodale Ansätze generieren können. Des Weiteren zeigen wir, dass unser Ansatz dazu in der Lage ist, Abweichungen vom erwarteten Verhalten des Roboters zu erkennen. Durch seine Generalität ist unsere Pipeline auf andere Roboter mit vergleichbarer Sensorik übertragbar.

Acknowledgements

I wish to thank Prof. Manuela Veloso for her guidance, hospitality and welcoming me as a part of the research group CORAL. I want to express my gratitude Prof. Ruediger Dillmann and the CLICS exchange program for enabling me to write my bachelor's thesis at the Carnegie Mellon University in Pittsburgh during the winter semester of 2017. It was a great experience and an opportunity for a unique scientific and intercultural exchange. Also, I want to thank Margit Roedder who guided me through the application process and always had an open ear for my questions. During my stay, I had the pleasure to meet Chenghui Zhou and João Cartucho. With both of them I had many interesting discussions and learned a lot. Kevin Zhang, Travers Rhodes, Aaron Roth and I shared a laboratory on the second floor of the Gates Hillman Center. We helped each other out with questions regarding the use of the robots and software. Finally, I am very grateful to my parents and my brother who never failed to encourage me over all these years and always believed in me.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Introduction

When robots and humans collaborate side by side, it is desirable that the machines possess an awareness about their activities in the physical world. While the planning layer represents an abstraction that captures a robot's intentions, it makes no statement about its actual state and activity. Assume a robot wants to move a certain distance forward. During its movement it might collide with an obstacle, fall over and be unable to proceed in its program. A robot that can recognize the unexpected body state might be able to recover or call for help. In another scenario, a robot might be pushed by a human or be under remote control. If a robot understands what is happening to its body, it can give a warning when it is being moved in a way that is overly demanding on its mechanics. In case of remote control, it might even reject user commands to prevent damage. Furthermore, a robot that has an awareness of its own phyiscal activities is capable of narrating its actions to a remote user and can verify its own plan execution.

Sensor-based Human Activity Recognition (HAR) utilizes wearable sensors such as accelerometers and gyroscopes to capture human activity and finds application in areas including mobile computing [29], ambient-assisted living [6] and health care [2]. The field has been thriving in recent years mainly because of the widespread use of smartphones and activity trackers. HAR develops methods to recognize the activities of the human body (e.g. walking or climbing stairs) by detecting activity patterns from streams of sensor data. This happens in the pursuit of creating more intelligent and personal devices which possess an awareness about their user's activity . While sensor-based HAR needs to go through the process of attaching and calibrating sensors for each individual user, robots feature a wide variety of inbuilt sensors that give insight into their physical states. By combining HAR approaches with the rich proprioceptive sensor data that is provided by robots, we aim at creating agents that possess an awareness about their body's activity.

In this work, we propose an online activity recognition pipeline that enables a robot to recognize its own activities by analyzing heterogeneous, asynchronous streams of raw sensor data. The pipeline features a Long Short Term Memory (LSTM) [14] based neural network. The approach is evaluated with the service robot Pepper and the humanoid robot Nao both developed by Softbank/Aldebaran Robotics. In our first experiment, we recognize a set of one and two-arm based movements executed by the Pepper robot in the simulator Gazebo. After this proof of concept, we evaluate our approach on the real Pepper and Nao robots. In our second experiment, the pipeline recognizes a set of 7 different base movements by utilizing a pre-trained LSTM

based neural network that leverages multimodal sensor data (joint states, electrical current, orientation, angular velocity and acceleration). The proposed activity pattern recognition approach combines information from heterogeneous sources in the pursuit of achieving better recognition results compared to a unimodal robot activity recognition approach. Finally, we detect human interference that stops Pepper from moving forward. Through its generality, our architecture can be utilized on other robot with proprioceptive sensing capabilities.

The rest of this bachelor's thesis is organized as follows:

- Chapter 2 discusses related work in the field of robotics and HAR. We take a look at how a robot's proprioceptive sensor data is used for monitoring and Collision Detection and how wearable sensors are utilized to recognize human activity.
- Chapter 3 gives an overview about the theoretical foundations of this work as well as an introduction of the used robots and software environment.
- Chapter 4 formulates the task of sensor-based activity pattern recognition, discusses the proprioceptive data provided by Pepper and Nao and proposes a pipeline for online activity recognition.
- Chapter 5 explains the experimental setup and presents and discusses the findings.
- Chapter 6 concludes this work and suggests future research directions.

2. Related Work

In this work, a robot's proprioceptive sensor data is utilized to gain insight into its internal state. This data is then analyzed in order to provide the machine with a basic awareness of its actions in the physical world. In this pursuit, models are learned that use streams of sensor data to recognize the machine's activities. In the field of robotics related work can be found in the areas of Collision Detection and Execution Monitoring. In addition, similar approaches can be seen in the field of Human Activity Recognition. There, systems equipped with sensors observe human activities to provide their users with more personal and intelligent functionality.

2.1. Proprioceptive Sensor Data in Robotics

A robot's sensors capture rich data about its internal state and external environment and are an essential part of robotics. Together with control and actuators, sensors form a closed loop that enables the machine to function in the world. While exteroceptive sensors capture a robot's environment, proprioceptive sensors capture the internal state of the machine. By analyzing streams of proprioceptive sensor data, one can observe a robot's state over time and draw conclusions about its physical condition. For example, a sudden change in a robot's acceleration vector can indicate a collision. While the fields of Collision Detection and Execution Monitoring are interested in fault avoidance and fault detection/recovery respectively, we aim at providing robots with a basic awareness about their activities. In that regard, we see our work more related to the field of Human Activity Recognition.

2.1.1. Collision Detection

In Collision Detection a robot's inbuilt body sensors are utilized to allow the machine to handle intentional or accidental contact with its physical environment. Haddadin gives an overview about current proceedings in the field [12]. The first step of the detection process analyzes a robot's sensor data to detect collisions. After a collision has been detected, it is desired to isolate the part of the robot's structure that is affected as well as the directions and magnitudes of the acting forces. This information is then used to allow the robot's control to initiate a suitable reaction. One of the central motivations of Collision Detection is to enable the machine to share a common workspace with humans by preventing injuries caused by forceful impacts as well as preventing damage to the robot's body.

2.1.2. Execution Monitoring

The area of Execution Monitoring (also known as Fault Detection and Diagnosis) observes the sensor data of a machine over time to detect and classify faults and their causes [23, 16]. Examples for faults are mechanical jams or the loss of hydraulic fluid [1]. The discipline is rooted in the field of industrial control and has found its way into robotics where it analyzes the state of a robot's body, sensors and processing units. Traditional Execution Monitoring approaches analyze the activities that a certain robot performs during normal operation. For each activity, a set of features is determined to indicate its correct execution. During runtime, these predefined features are then monitored and, in order to detect anomalies, are either compared with the expected system behaviour or directly subjected to pattern recognition methods. If a fault has been detected, the system communicates its finding to its users. Also, it is desired that the machine is capable of self-recovery to maintain a functional state. For example, this can be achieved by switching between redundant systems.

2.2. Human Activity Recognition

As a subdiscipline of Human Computer Interaction, Human Activity Recognition (HAR) uses sensors to collect data about user activity. This data is then analyzed in order to allow devices to recognize the activities that are performed by their users. This happens in the pursuit of creating more intelligent systems that utilize the knowledge about their user's behaviour to provide more intelligent and personal services. The field of HAR has been thriving in recent years mainly because of the widespread use of wearable sensors such as smartphones and activity trackers. Different HAR approaches can be described as camera-based, microphone-based or sensor-based. A robot is most similar to an HAR system that utilizes wearable sensors. Different inbuilt sensors stream information about the machine's state and can be used to recognize its performed activities. The rich research that has been done in HAR was a major inspiration for this work.

2.2.1. Sensor-based Human Activity Recognition

Wearable sensors, including accelerometers and gyroscopes, allow to capture data about the activities of the human body. Devices such as smartphones (Figure 2.1a) and smartwatches (Figure 2.1b) have inbuilt sensors to enable them to get insight into their users physical activities. This makes the devices more personal and provides them with rich information about their user's daily life. A combined approach between wearable- and camera-based recognition are motion capture methods (Figure 2.1c). Here, the human wears a suit which is equipped with markers that are tracked by multiple cameras to detect the positions of his joints in 3-D space. This information is similar to the joint angles that are provided by a robot. Another use case for wearable sensors is fall detection. Here, impaired people can be equipped with wearable devices that detect falls and call help if necessary [32]. Cornacchia provides a comprehensive survey about HAR with wearable sensors [7].

Traditional HAR often uses sliding window based techniques. It extracts handcrafted features, which are designed by domain experts, from the raw sensor readings. While these approaches achieve satisfying recognition results on simple activities such as lying, standing and walking, it is difficult to grasp more complex behaviours. This limitation mainly lies in the manually engineered features that are restricted by human domain knowledge [4]. Another factor is the limited number of observations in window based approaches. Recent advances in HAR utilize deep learning techniques because of their automatic feature generation and selection. Deep learning approaches such as LSTMs and Convolutional Neural Networks (CNN) can come up with task specific non-linear features and show promising results [30].

Another challenge of HAR is the cross-people activity recognition problem. Models that are trained with data from a specific person often have trouble when analyzing data from other persons [33]. This difficulty lies in the physical differences between people and the way they move. The acceleration that acts on a tall person's waist differs from the one that acts on a short person's waist. Another example is the walking style of a child and an elderly person that relies on a crutch. While differences between individual persons can be rather large, robots from the same type that execute the same tasks are more homogenous in their structure and behaviour.

(a) Smartphone
Source: [13]

(b) Smartwatch
Source: [5]

(c) Motion Capture
Source: [19]

Figure 2.1.: Different Human Activity Recognition systems.

2.2.2. Multimodal Human Activity Recognition

Multimodal HAR approaches use data from multiple sensors to recognize human activities. This allows to capture more detailed data about the body's activities. By utilizing combined data from a variety of sensors, they can outperform unimodal approaches [20]. One example is the use of multiple accelerometers and gyroscopes worn by a human to capture information about the activity of his different body parts [21, 26, 31]. Another example combines readings from an inertial measuring unit with data from a depth camera [13] to recognize hand gestures. In [20], human activities

were captured by a variety of accelerometers, microphones, depth cameras, an impulse motion capture system as well as quad and stereo cameras. While sensor-based HAR relies on wearable sensors that need to be attached to the individual users, robots are already equipped with a variety of inbuilt sensors. This convenient access to heterogenous proprioceptive data facilitates the use of multimodal activity recognition methods in the field of robotics. In our later evaluation, we combine data describing a robot's joint angles, electrical currents, acceleration, orientation and angular velocity.

3. Basic concepts and principles

In this chapter, we discuss the underlying concepts of our work. This includes the formal definition of activity patterns, the formal methods how to learn to recognize activity patterns, a description of the used robots Pepper and Nao and the related software environment.

3.1. Theoretical Background

3.1.1. Nearest Neighbor Classification

The Nearest Neighbor algorithm is a non-parametric classification method that utilizes a metric d to measure similarity between values that take values in a given metric space X. As a basis for its classifications, the algorithm uses a set of n pairs $\{(x_1, \theta_1), \ldots, (x_n, \theta_n)\}$. For each $i \in \{1, \ldots, n\}$, $x_i \in X$ represents a measurement and $\theta_i \in 1, \ldots, m$ the category x_i belongs to.

Given a new pair (x, θ), it is desired to estimate the class θ measurement x belongs to by utilizing information contained in the set of known pairs. It does so by using metric d as a similarity measurement between points in X. We call measurement

$$x' \in x_1, \ldots, x_n \tag{3.1}$$

a nearest neighbor of x if

$$\min_{i \in \{1,\ldots,n\}} d(x_i, x) = d(x', x) \tag{3.2}$$

The algorithm decides x belongs to the same category θ' as its nearest neighbor x'. While the more general k-Nearest Neighbors algorithm takes a look at the k closest neighbors in the vicinity of x, the basic Nearest Neighbor algorihm only considers the point closest to x [8].

In a previous work, a nearest neighbor based algorithm has been utilized for behavior recognition in robot soccer [9]. In this context, time series containing multiple measurements are analyzed and classified. Each series describes a sequence of consecutive measurements recorded during a given time frame $[t_i, t_{i+k}]$ for $i, k \in \mathbb{N}$ as $T_A(t_i, t_{i+k}) = \{M_i, M_{i+1}..., M_{i+k}\}$ where $M_j = (x_1, ..., x_n) \in \mathbb{R}^n$ is the respective measurement at time $j \in [t_i, t_k]$. As a similarity metric between time series, the Hausdorff metric can be used in combination with the Euclidean distance d. The Hausdorff metric functions as a measurement between two sets by determining the

maximum of the minimal distances between two elements of different sets. Given two sequences T_1 and T_2 the Hausdorff distance is defined as:

$$H(T_1, T_2) = \max\{\max_{p \in T_1} \min_{q \in T_2} d(p, q), \max_{p \in T_2} \min_{q \in T_1} d(p, q)\} \tag{3.3}$$

We can reduce the computational cost for classifying a motion sequence T with m measurements to $\mathcal{O}(|C| * \max_{c \in C} |c| * m)$ by determining the centroid $c \in C$ for each class and averaging over a set of labeled training sequences. Using the Hausdorff distance allows us to measure the similarity between time series even if they have different durations.

3.1.2. Cross-Entropy

When training a neural network, it is essential to choose a suitable loss function. Cross-entropy can be used as a measurement of the difference between two probability distributions p and q over the same random variable x. For the discrete case, cross-entropy is defined as:

$$H(p, q) = -\sum_{x} p(x) log(q(x)) \tag{3.4}$$

During the training process, one can calculate the cross-entropy between the prediction vector generated by the network and the ground truth of the training data which is commonly a one-hot encoding containing the class that is to be predicted. The training process then minimizes the cross-entropy between network prediction and ground truth [11].

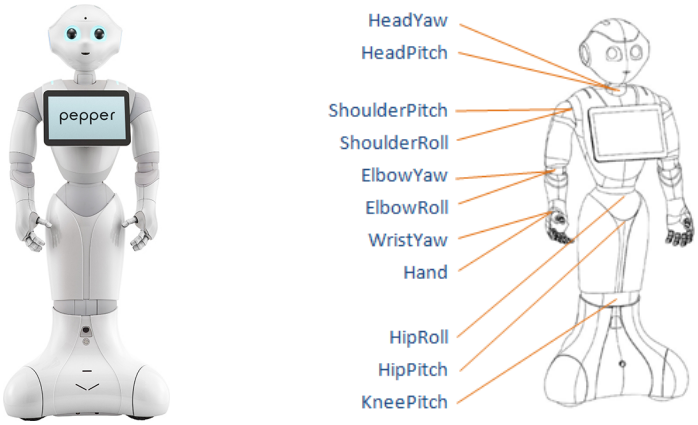
3.1.3. LSTM

Recurrented Neuronales Netz Kann Zustand halten Erkennt ab Gewissen Zeitpunkt Bewegung

3.2. Utilized Hardware and Software

3.2.1. Pepper

The humanoid-like robot Pepper (shown in Figure 3.1a) is 121cm tall, has a weight of about 28kg and has twenty degrees-of-freedom (DOF). It has an omnidirectional base with 3 DOF, a body with 3 DOF, two arms each with 5 DOF, two simple five-fingered hands with 1 DOF each, and a head with 2 DOF. One inertial measurement unit, which is in the base, provides information about acceleration and rotation in 3D space. Because Pepper was intended to interact with untrained users on a daily basis, extra precautions were taken to prevent damage to its environment and itself. For obstacle avoidance, multiple sensors are integrated around its base. It is equipped with three bump sensors: two in the front and one in the back; 2 sonar sensors: one in front and one behind, which measure the distance to potential obstacles; 6 pulsed laser line generators and sensors in the front and on the sides; and 2 infrared sensors on the left



(a) Pepper, Source: [22] (b) Pepper's Joints. Source: [27]

Figure 3.1.: The service-robot Pepper.

and right sides. If any of these sensors detects an obstacle, NAOqi will immediately stop the movement of the base to prevent accidents.

It has access to a variety of options to interact with users through multi-modal communication. It is capable of nonverbally signaling its state by using 3 groups of LEDs positioned in its eyes, around its ears and on its shoulders. Additionally, it is equipped with an Android tablet on its chest, which can be used for touch-based input or for displaying information. The manufacturer provides a toolkit to develop applications and hosts a platform for developers to distribute their software to other users. Both the tablet and the robot have two independent wireless connections, and there is no way to transfer data directly between the two devices. With 3 tactile sensors located on top of its head and one on the backside of each hand, the humanoid can also react to users touching it. To facilitate verbal communications, the robot can talk and play sounds through the speakers in its ears and listen to commands by using an array of 4 directional microphones that are located on top of its head. By determining the direction of the sound of a noise Pepper can orient itself to look in the direction of a user.

Pepper has two RGB cameras, one on its forehead pointing upwards in the direction of a potential user's head, and one in its mouth pointing downwards. Both have a native resolution of 640*480 and can provide either 640*480 images at 30 fps or 2560*1920 at 1 fps. Furthermore, one ASUS Xtion 3D sensor is placed behind the eyes and can be used for localization, navigation and distance measurements. Based on the camera images, the inbuilt NAOqi vision API provides some default functions like red ball and landmark detection.

With respect to computational resources, Pepper contains an Atom E3845 quad core processor running at 1.91 GHz, 4 GB DDR3 RAM, 8 GB flash memory, a micro SDHC with 16 GB of memory and an integrated Intel HD graphics card. The robot is running a proprietary operating system called NAOqi, which is based on the Gentoo

GNU/Linux distribution. Developers do not have root access and can use sudo only for shut down or reboot commands [27, 22].

3.2.2. Nao

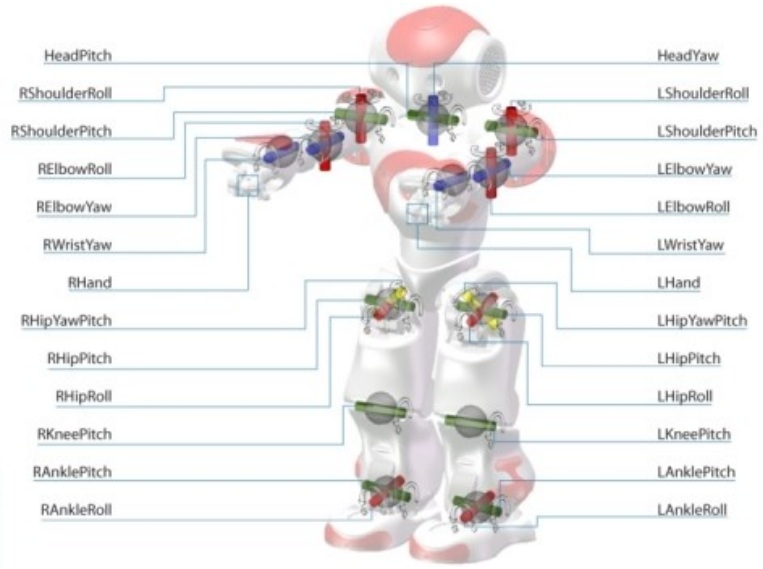


Figure 3.2.: Nao's Joints. Source: [27]

The humanoid robot Nao (shown in Figure 3.2) is 58cm tall, has a weight of about 4.3kg and has twenty-five DOF. It has two legs each with 5 DOF a hip with 1 DOF, two arms each with 5 DOF, two simple three-fingered hands with 1 DOF each, and a head with 2 DOF. Its inertial measurement provides information about acceleration and rotation in 3D space. Nao has two inbuilt RGB cameras, one positioned in its forehead pointing upwards and one in its mouth pointing downwards. The robot runs the same NAOqi operating system as the Pepper robot. The ALMemory module can be used to access the sensor readings of the machine [27].

3.2.3. NAOqi

The NaoQi OS is an operating system developed by Aldebaran to run on its robots and is based on the Gentoo GNU/Linux distribution. When started, the operating system runs the NAOqi executable which controls the robot and provides local and remote access to all capabilities of the robot, including actuators, sensors and network connection. The system allows for sequential, parallel and event-driven function execution [27, 24].

The NaoQi framework acts as a distributed environment and enables software modules which can either run on the same or on different machines, to communicate with each other. On start, NAOqi loads autoload.ini which is a preference file that defines which libraries should be loaded. Each library contains one or multiples modules which in turn contain one or multiple function. The resulting structure

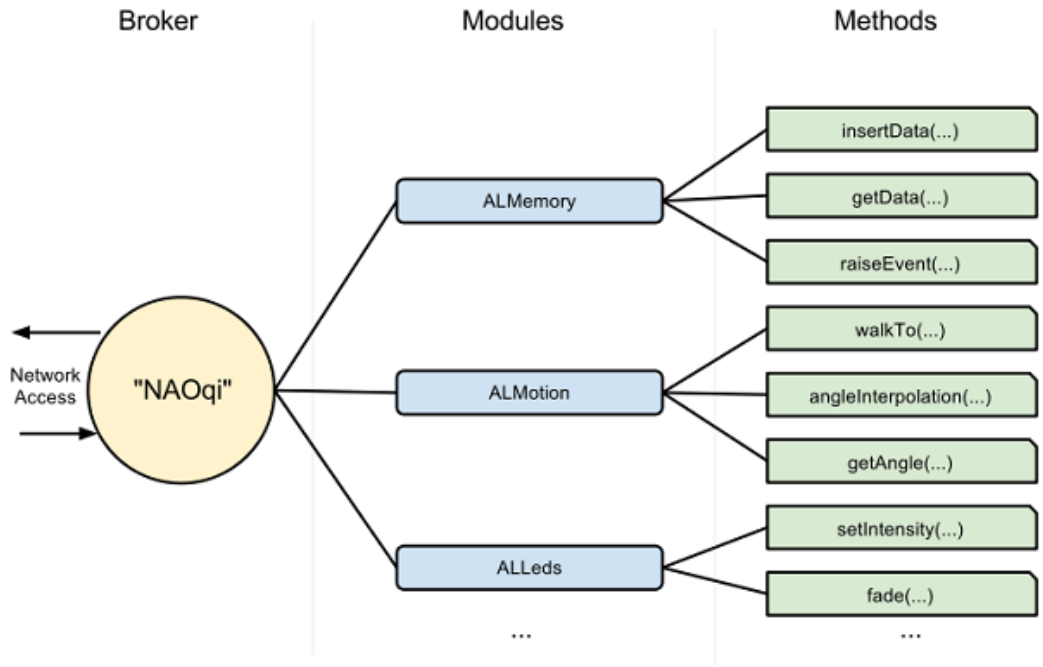


Figure 3.3.: NAOqi acting as a broker. Source: [27]

resembles a tree where the broker functions as a root with attached modules and modules with attached functions. During runtime, NAOqi acts as a broker (shown in Figure 3.3). It advertices the functions of the modules and offers lookup services to find the advertised services provided by the modules. For network capabilities, multiple brokers can be connected by specifiying the IP address and port number of the main broker [27, 22].

Two types of modules are supported by the framework. The first type consists of local modules that can only run on the robot. They run in the same process and can share variables and can call each other without serialization or networking. This allows for the fasted possible communication and is recommended for closed loop control. Local modules share the same broker process. The second type consists of remote modules that can run either on or outside the robot. They communicate with the other modules over network. Remote modules enable the use of extern computing, but are not capable of fast intermodule communication [27, 22]. NAOqi comes with a public API and a range of base modules which provide, among others, memory, communication, motion, audio, vision and sensor capabilities.

The NAOqi framework supports cross-platform development for Aldebaran's robots and can be run on Linux, Windows and Mac OS. Modules can be created with either Python or C++. Via API calls, access to the default NAOqi modules is provided. Modules created in Python are interpreted and can be flexibly run romotely or locally on the robot. Meanwhile, C++ modules need to be compiled to the target operating system before they can be run remotely or locally. If the developer wishes to run a

C++ module locally on the robot, he is required to compile the excutable for the NAOqi OS. [27, 22].

3.2.4. ROS

The open source Robotic Operating System (ROS) serves as a middleware framework for robotics software. It enjoys an active community and provides a wide range of libraries and tools that support the development process. These include an inter-process communication framework, tools for visualization and algorithms for perception, planning and localization. Although there is always one coordinating master node, other nodes can run on different machines and communicate with each other over the network. Because of this, ROS provides the means to extend NAOqi's inbuilt capabilities with a wide range of state-of-the-art libraries and algorithms to enable more advanced applications [25, 22].

In a ROS system, functionalities are realized by running multiple processes called nodes in parallel. One application usually runs multiple nodes that are independent of each other. Nodes have two ways of communication with each other. The first way is the use of messages which represent typed data and follow the publisher/subscriber paradigm. If one node wants to send something, it can publish a message to a given topic for example "acceleration" or "camera". Nodes that are interested in receiving messages of a certain topic subscribe to it. Multiple nodes can publish and/or subscribe to the same topic. Individual nodes can publish and/or subscribe to multiple topics. To ensure loose coupling publisher and subscriber are usually not aware of each others existence, the communication over messages is asysnchronous. As a second way of communication, services can be used to achieve synchronous communication. The use of services follows the client/server paradigm and allows synchronous transactions. A service is defined by a string that represents the service name and two message types for request and answer. Only one node can advertise a service under a given name [25, 22].

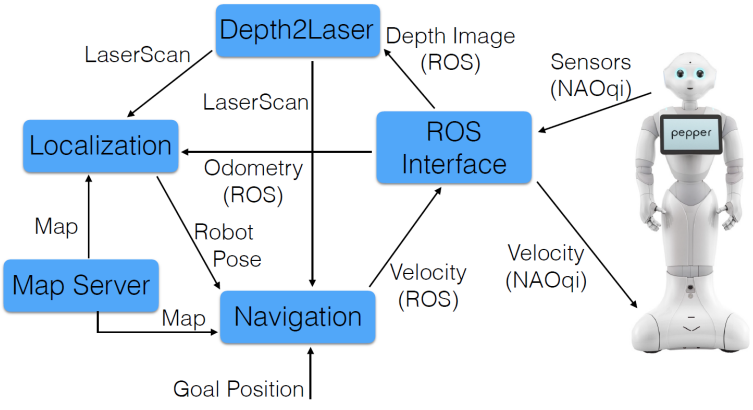


Figure 3.4.: NAOqi and ROS can communicate over an interface. The image shows how SLAM (simultaneous localization and mapping) can be realized by combining NAOqi's with ROS's libraries. Source: [22]

NAOqi and ROS can communicate over a ROS interface (shown in Figure 3.4) that is provided by the ROS community [10]. The interface runs as both a NAOqi module and a ROS node and allows to combine their functionalities. During runtime, it provides NAOqi's base functionalities as ROS services and topics by utilizing wrappers that handle the communication with the native NAOqi system. It uses standardized message types to enable the easy integration of Pepper with ROS. Therefore, it allows the use of common ROS packages and tools like OpenCv, GMapping and rviz. Also, it allows to port the developed code from and to other robots and to use simulation software like Gazebo [22].

3.2.5. Gazebo

Gazebo is a robotics simulator that is supported by the Open Source Robotics Foundation (OSRF). It allows for 3D robotics simulations which provide a graphical visualization, simulated sensor values and a physics engine. In combination with ROS, Gazebo is a useful tool to test algorithms, robot design and tests in a realistic environment. New models of robots can be created by defining bodyparts, joints and sensors. The simulation regards mass, friction and bounce factor of the body parts and the kinematic and dynamic relationships of the joints [17].

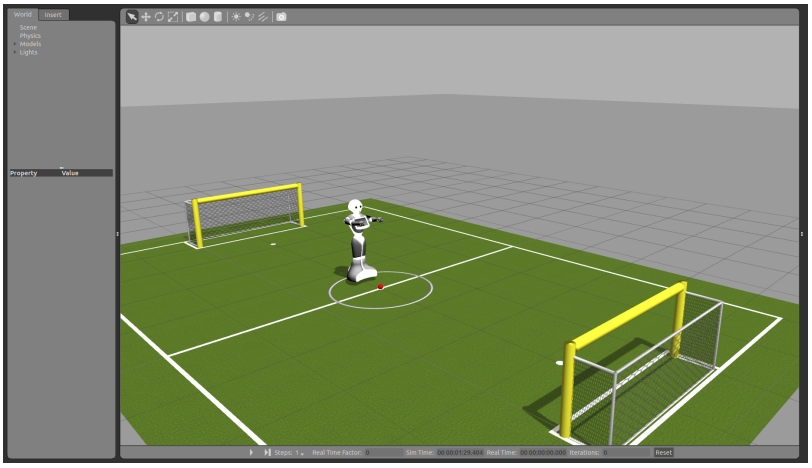


Figure 3.5.: Pepper simulated in Gazebo.

The ROS community offers a configuration file for Pepper [25]. It allows us to simulate and test the robot in a virtual environment. The environment does not support NAOqi calls and so the complete robot control must be written in ROS. Also, it is possible to use MoveIt! to control the robot's body. Gazebo provides us with simulated sensor values of our robot. Because the available model of Pepper has no official support, there are some issues. For example, the sensor values in the messages have a different order than on the real robot and some movements tend to make the robot fall over. Nontheless, it allows for some simple testing without wearing out the real robot. This is especially useful when we want to record multiple executions of a motion sequence in the pursuit of training an AI system.

4. Analysis and Design

In this section we start with formulating the task of sensor-based activity pattern recognition. Subsequently, we discuss what kinds of sensor data can be utilized to achieve reliable predictions of robot activity and take a look at the data provided by the robots Nao and Pepper. We discuss the required steps to leverage a robot's proprioceptive sensor data and propose a general architecture that allows a robot to recognize its own acitivities by analyzing heterogeneous, asynchronous streams of raw sensor data during runtime.

4.1. Sensor-Based Activity Pattern Recognition

We formulate the task of sensor-based activity pattern recognition inspired by similar approaches in the field of HAR [30]. Sensors act as a connection between the real world and the digital computer and allow us to observe a robot's physical state. During task execution, data streams generated by a robot's sensors can be leveraged to gain high-level insights about the activities that are performed by the machine. Assume a robot is executing a sequence of activities belonging to a predefined set A:

$$A = \{a_i\}_{i=1}^{n} \tag{4.1}$$

where n marks the number of activity types. We observe the sensor data that is generated over time. The observed sequence s contains m consecutive sensor readings r_i, $i \in \{1, \ldots, m\}$, that capture the state of the robot during a given period of time at equal intervals. Each of the m readings features l attributes.

$$s = (r_1, \ldots, r_m), \quad r_i \in \mathbb{R}^l \tag{4.2}$$

We aim at creating a model $\mathcal{M}$ that generates a sequence of predictions $\hat{A}$ about the performed activity at the time of each given reading r_i

$$\hat{A} = (\hat{a}_1 \ldots, \hat{a}_m) = \mathcal{M}(s), \quad \hat{a}_i \in A \tag{4.3}$$

where the actual performed activity sequence A^* is:

$$A^* = (a_1^*, \ldots, a_m^*), \quad a_i^* \in A \tag{4.4}$$

An appropriate model $\mathcal{M}$ minimizes the discrepancy between predicted sequence $\hat{A}$ and ground truth sequence A^*. This formulation makes multiple assumptions about the nature of the sensor readings generated by the robot. Mainly, it assumes that sensor readings are aggregated units that are sampled at equal intervals. On a real

robot, multiple sensors can generate readings asynchronously and at different rates. Moreover, on some systems, sensor data is only published on state changes. Therefore, it is necessary to synchronize the streams of sensor data and to generate an equal stream of synchronized readings.

In 4.3 we tackle these limitations by introducing the architecture of an online recognition pipeline that is capable of analyzing data from multiple, asynchronous sensors. Furthermore, rather than training our model directly on raw sensor data, a set of preprocessing functions Φ_j, $j \in 1, \ldots, k$, for each of the k sensors is utilized (see 4.3.1). The neural network introduced in 4.3.4 is optimized by minimizing the multinomial cross-entropy that functions as measurement for the discrepancy between prediction $\hat{A}$ and ground truth A^*.

4.2. Proprioceptive Sensor Data

During each deployment, proprioceptive sensors capture the state of a robot's body in the physical world over time. Heterogeneous sensors, such as accelerometer and gyroscopes, provide streams of activity information that can be utilized to recognize the actions that are performed by the machine. Common types of proprioceptive sensor data include acceleration, torque, velocity, electrical current, voltage, orientation, joint states and temperature. The individual types of data vary in significance based on the class of activity that is to be predicted. For example, acceleration and torque capture the forces that act on a robot's body at a given time and are suitable for detecting motion activity. Meanwhile, temperature can be seen as an indicator for long term engine activity by being dependent on the amount of heat that is generated during runtime.

4.2.1. Sensor Model

Formally, we assume that a robot features k sensors S_j, $j \in \{1, \ldots, k\}$, that give insight into its physical state. Each sensor S_j samples a signal $p_j(t)$ at a given rate f_j over time t. A reading of sensor S_j at time t_0 provides a d_j dimensional vector:

$$S_j(t_0) = (v_1, \ldots, v_{d_j}) \in \mathbb{R}^{d_j} \tag{4.5}$$

The combined use of readings from heterogeneous sensors can achieve more accurate predictions than mono-sensor approaches in the field of HAR [18, 21, 26]. While sensor-based HAR relies on wearable sensors that have to be fitted for the individual user, a robot's body is equipped with a variety of inbuilt sensors. Therefore, it is convenient to combine data from multiple sensors which facilitates the application of multimodal activity pattern recognition methods.

4.2.2. Sensor Data in NAOqi

Both the Pepper (3.2.1) and the Nao (3.2.2) robot feature the NAOqi operating system (3.2.3) which allows access to readings generated by their inbuilt sensors through the

ALMemory API. The ROS community provides an interface which allows the use of NAOqi functionalities via ROS topics and services (3.2.4). It does so by implementing a set of multiple wrappers that communicate with the ALMemory module. The basic version of the NAOqi/ROS API provides information about the joint configuration and inertial measurement unit (IMU) via designated topics. Additionally, we are interested in the electrical current that acts on the individual joints of the robot. Therefore, we implemented an additional ROS node that functions as a wrapper to communicate with NAOqi. This node samples information about the electrical current acting on the robot's joints at equal time intervals and publishes the information to a respective topic for further use.

	Sampling Rate (Hz)	
Sensor Data	Pepper	Nao
Joint States	50	20
IMU	10	20
Electrical Current	10	30

Table 4.1.: The sampling rates for the individual robot.

In our evaluation in Chapter 5 we utilize joint state, electrical current and IMU data of the robots Pepper and Nao. More specifically, we analyze joint states and electrical current for each of their joints. Pepper features 17 and Nao 20 joints. Additionally, we use acceleration, angular velocity and orientation information provided by the 3-axis IMU of the robots. Table 4.1 gives an overview about the sampling rates of the used sensor data. Because the sensor data is asychronously sampled at different rates, it is necessary to synchronize them for further analysis. In 4.3.2 we introduce an approach to synchronize and fuse multiple asynchronous streams of heterogenous sensor data to a combined stream.

4.3. Online Activity Pattern Recognition

We introduce a general approach for activity pattern recognition in robotics. The proposed architecture features a 4-step pipeline (shown in Figure 4.1) that is capable of generating predictions by utilizing heterogeneous, asynchronous streams of proprioceptive sensor data during runtime. The raw data from each sensor (4.2.1) is preprocessed separately (4.3.1) and is then fused to a combined synchronous data stream (4.3.2). Afterwards, the combined readings are scaled and standardized (4.3.3) and subsequently passed to a LSTM based neural network which predicts the current activity that is performed by the robot (4.3.4). Both scaler and network are intended to be trained offline with prerecorded annotated activity sequences as shown in Section 4.4. In the following, we discuss the individual pipeline steps in detail with regards to the implementation of our system.

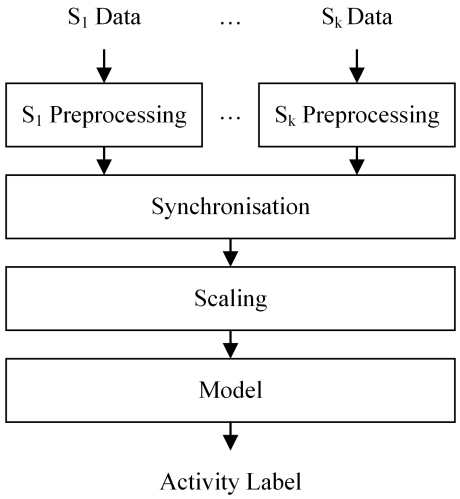


Figure 4.1.: The proposed activity recognition pipeline. It predicts the current activity performed by the robot by leveraging data streams from multiple proprioceptive sensors.

4.3.1. Preprocessing

The recognition pipeline receives one stream of raw sensor data from each of its k sensors. The featured LSTM based neural network is capable of generating and selecting suitable features for pattern recognition autonomously during training. However, it can be favorable to perform sensor specific transformations before learning the model. This can reduce the amount of required training samples by injecting certain expert knowledge into the model. Because of this, each sensor S_j is related to a preprocessing function Φ_j that is implemented in a separate module. Thereby, the raw sensor data is transformed to a d'_j dimensional feature vector.

$$\Phi_j(S_j(t_0)) = (v'_1, \ldots, v'_{d'_j}) \in \mathbb{R}^{d'_j} \tag{4.6}$$

For our evaluation in Chapter 5, we implemented one ROS node for each of the topics which provide us with raw sensor data. Joint states and electrical current data is being scaled to unit space based on the individual sensor specifications. In addition, a filter is applied to the raw acceleration data generated by the inertial measurement unit to generate an additional separation of low frequency gravitational acceleration from high frequency activity acceleration. A similar approach to acceleration data filtering was used in [3].

4.3.2. Synchronization

In general, the pipeline functions on a robot which samples its k inbuilt sensors at different rates asynchronously. Therefore, we deal with asynchronous streams of heterogenous sensor data. The utilized recognition module assumes all sensors to be

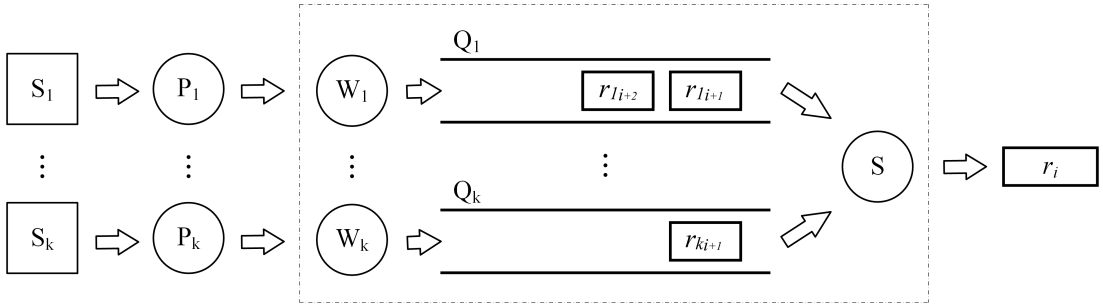


Figure 4.2.: Synchronization Module.

sampled synchronously at a given rate f. Therefore, it is necessary to synchronize the individual streams of preprocessed sensor data for further use.

The synchronization module fuses the different streams of preprocessed sensor readings to one combined stream. First an initial start time t_0 is being determined. Subsequently, for each $t_i = t_0 + i * T$, $(T = 1/f)$ one combined measurement is being interpolated as follows: The module observes each data stream in parallel. For each sensor S_j one reading r_{j_m} (the m-th reading of S_j) is kept in a buffer together with the timestamp of its creation $t_{r_{j_m}}$. When a new reading $r_{j_{m+1}}$ at time $t_{r_{j_{m+1}}}$ arrives, the condition $t_{r_{j_m}} \leq t_i < t_{r_{j_{m+1}}}$ is being checked. If the conditions are not met, the synchronization module updates its buffer with $r_{j_{m+1}}$ and keeps on listening to the stream until another reading fullfills the requirement. If the conditions are met, the buffer is updated likewise and a linear interpolation between r_j and $r_{j_{m+1}}$ is performed to determine

$$r_{j_i} = \frac{r_{j_m} * (t_{r_{j_{m+1}}} - t_i) + r_{j_{m+1}} * (t_i - t_{r_{j_m}})}{t_{r_{j_{m+1}}} - t_{r_{j_m}}} \tag{4.7}$$

where r_{j_i} is the representative vector for sensor S_j at time t_i that will be utilized for activity prediction. Afterwards, the module buffers r_{j_i} in a queue and continues to determine $r_{j_{(i+1)}}$ analogously. After one vector r_{j_i} for each Sensor S_j has been determined for time t_i, it deques the vectors, concatenates them and passes the combined and synchronized data to the next pipeline step.

Our implementation of the synchronization module is shown in Figure 4.2. For each sensor S_j, $j \in \{1, \ldots, k\}$, a worker W_j analyzes the stream of data published by preprocessing node P_j. For a time t_i, worker W_j interpolates a representive vector as described above and puts it into queue Q_j. Subsequently it continues to interpolate an entry for t_{i+1}. After one representative vector for each sensor for time t_i has been put into the queue, synchronizer S dequeues the individual vectors and fuses them to a combined reading r_i. This reading is then published to a topic for further processing.

4.3.3. Scaling

The scaling module subtracts the mean from the individual features contained in the synchronized readings and scales them to zero mean unit variance. This pipeline step reduces the numerical difficulties in the training process and avoids that features in a greater numeric range have a negative impact on the optimization process. In

our implementation we utilize the StandardScaler implementation of the scikit-learn library which we train offline with prerecorded activity sequences (4.4). Afterwards, the scaler is saved and later loaded during the initialization of the online recognition pipeline.

4.3.4. Model

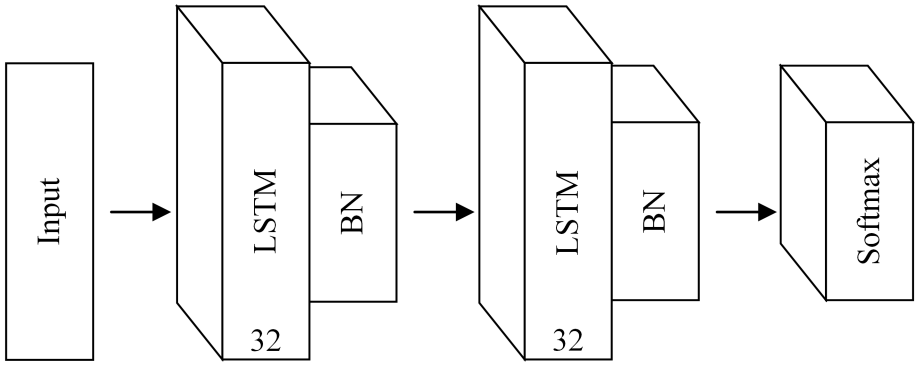


Figure 4.3.: The utilized Neural Network.

The model receives a stream of synchronized and scaled equidistant sensor readings from the previous pipeline steps. This stream matches the requirements for the activity pattern recognition formulation described in Section 4.1. Each reading describes the physical state of the robot at a given point in time. The model analyzes the synchronized stream of heterogenous sensor data and outputs activity labels that describe the activity that is currently performed by the robot. Depending on the activity patterns that are intended to be recognized, a suitable model can be selected. The selected model is trained offline with annotated activity sequences as described in Section 4.4.

Figure 4.3 visualizes the LSTM based neural network we utilized in our implementation. The robot's physical state is described by an input matrix containing multiple consecutive sensor readings. This matrix is prepared by a small buffer that proceeds the network. In case of the Pepper robot, we use 5 readings consecutive containing 50 features each. This input then goes through two layers of LSTM consisting of 32 neurons each. Afterwards, a softmax layer classifies each sequence into one of 7 classes (see 5.2.2). Each LSTM layer is followed by a batch normalization layer and is regularized by l_1 and l_2 regularizers each with coefficient 0.05. The categorical cross-entropy function is used to calculate the loss and Adam is the used optimizer. The python code that describes our model can be found in A.2.

4.4. Learning Activity Patterns

This section describes the training process for scaler and model which are used for the pipeline. During training deployments, the robot performs the set of activities that is

intended to be recognized. The data generated by the robot's proprioceptive sensors is captured as well as annotations generated by the control script. Both are being stored in a database and are later combined for training. During training scaler and model are prepared for the online deployment and saved for later use. Afterwards, during the initialization of the recognition pipeline, pretrained scaler and model are loaded and recognize the acitivies of the robot during normal deployment.

4.4.1. Data Collection

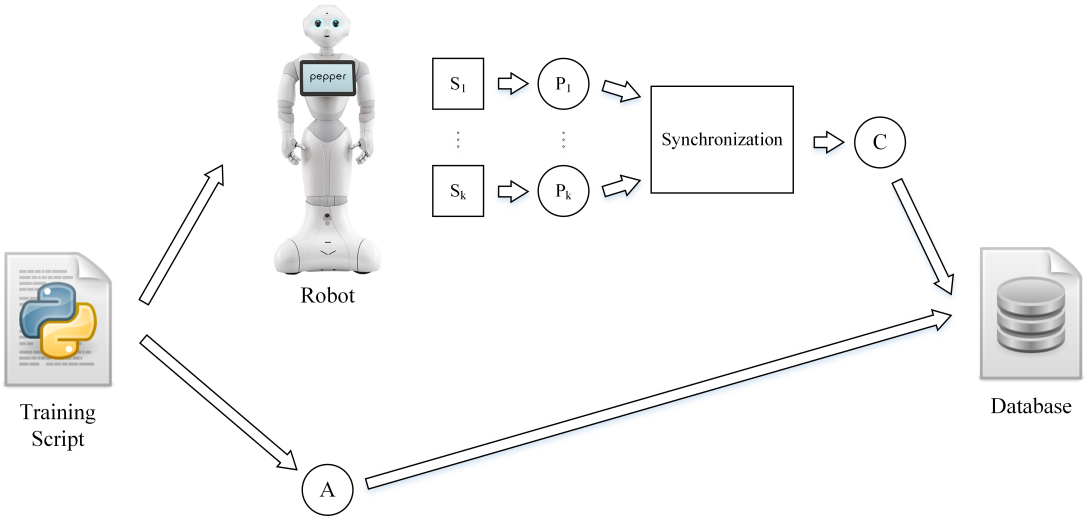


Figure 4.4.: The recording process.

For the training of scaler and model a training script is executed that performs two tasks in parallel (shown in Figure 4.4). Firstly, it lets the robot perform representative sequences of activities that are to be recognized. Secondly, it publishes annotation information about the current action that is being performed. While the robot executes the individual activities of the sequence its inbuilt proprioceptive sensors S_i collect data about the physical state of its body. The resulting data then goes through the preprocessing and synchronization stage of the pipeline. The raw readings from sensor S_i are preprocessed by node P_i and subsequently fused to one combined synchronous stream. This stream of activity information is then collected by a ROS node C which stores the data in a lightweight SQLite database for later use. In parallel, another node A collects the annotations which are published by the training script and stores them in the database. Both the combined reading created by the synchronization module and the annotations sent by the training script are associated with timestamps. Every execution of the training script results in a new database. After recording, we run a converter script which reads the individual databases, fuses sensor readings with annotations based on their respective timestamps and stores the resulting tables as pandas DataFrames afterwards.

During the data collection a human supervisor is required to verify the correct execution of the training script. During our evaluation in 5.2.2 we encountered the

problem that the Pepper robot sometimes ignored single commands of the training script. For example, the robot was commanded to execute a forward movement, but it kept standing still. The training script still sent the respective annotation and because of this, data which should be annotated as "standing" was annotated as "forward". This corrupted the training data and led to undesired training results.

4.4.2. Training Process

The data collected during the previous training deployments is used to learn scaler and model for the recognition pipeline. After the training process, scaler and model are being serialized and stored. During the initialization of the pipeline, they are then loaded and analyze the stream of readings provided by the synchronization layer.

In our pipeline implementation, we use the StandardScaler provided by the sklearn library, which subtracts the mean and scales the data to zero mean unit variance (see 4.3.3). We fit the scaler on the training data and store it as a pkl object. For the training of the neural network, the deep learning framework Keras is utilized. After training the models as described in 4.3.4, it is then stored for later use. The code for the training of scaler and model is executed in a Jupyter Notebook. This allows us to experiment with the parameters conveniently and to visualize our results directly.

5. Evaluation

In this section, we evaluate the proposed activity pattern recognition approaches in a simulation as well as on the real robots. In our first experiment, we train two models to recognize various one-arm and two-arm motions performed by a Pepper robot in the robotics simulator Gazebo. In the second experiment, the proposed pipeline (see Section 4.3) is utilized to recognize a set of base movements executed by the real robots Pepper and Nao. We show that a multimodal approach to robot activity pattern recognition that utilizes data from heterogenous sensors can lead to increased classification accuracy. Finally, we analyze the behavior of the model from the previous experiment in case of unexpected behavior. Here, a human interferes with the forward movement of the Pepper robot by pulling its base backwards.

5.1. Recognizing simulated Arm Motions

In the first experiment, we analyze series of joint state readings in order to recognize a set of arm based motions performed by a Pepper robot in the simulator Gazebo. We compare the recognition accuracy of two different models and analyze how many sensor readings they need to recognize the different motions. The first model is an LSTM based neural network. It contains an LSTM layer with 128 neurons, followed by a batch normalization layer which is then followed by a softmax layer to determine the classification. l_1 and l_2 regulizers are used each with coefficient 0.05. The Adam optimizer is used for training and minimizes the categorical cross-entropy function. The second model is a Nearest Neighbor based approach that uses the Hausdorff distance (3.1.1) to determine the similarity between trajectories. Some of the results of this experiment will be published in [15].

5.1.1. Recorded Data

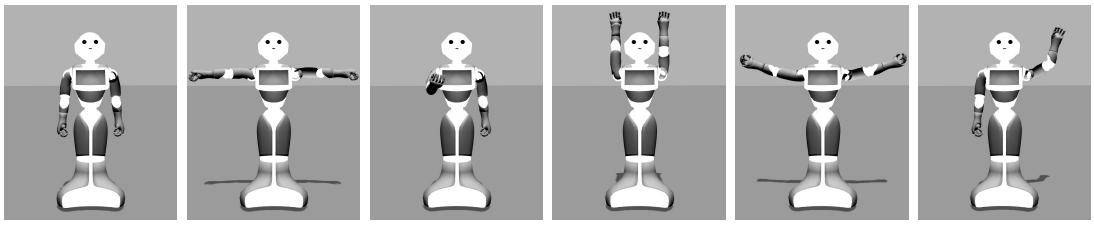


Figure 5.1.: The leftmost image shows the starting position for each motion (standing). The other images show end positions for five of the motions.

We evaluate the two models with recordings of 20 simulated arm-based motions (some of them shown in 5.1 together with the starting position). For every motion, the robot begins its movement from the starting position. By using the robotics simulator Gazebo and the control software MoveIt!, we collected 50 samples for each of the 20 individual motions. Each sample captures a 7-second-long motion sequence of the robot and contains 10 joint states readings per second. Every reading contains a 20 dimensional real valued vector. We scale each angle to unit space based on the robots specifications. In total, 700 samples were used for the training process and 300 for the testing of the models.

Internally, we handle the data as a 3 dimensional tensor. The first dimension represents the number of the sample, the second dimension represents the timesteps of the sequences (1 to 70) and the third dimension is the 20 dimensional joint angle vector recorded at the specific time step. For the nearest neighbor model, one centroid is calculated for each of the 20 individual motions by averaging the representative class samples of the training set. During the classification process, the nearest neighbor for each test sample is determined by utilizing the Hausdorff distance. We trained the neural network in two different ways. First, we trained a length dependent model by training separate networks for each sequence length in the range from 1 to 70. Subsequently, if we for example determine the class for a partial recording containing the first 30 readings out of a test sequence, we used the neural network that was trained on the first 30 readings of each training sample. Secondly, we trained a mixed length LSTM that can classify any sequence containing 1 to 70 joint state readings. We did so by taking the first n readings of each samples, $n \in \{1, \ldots, 70\}$, and filling the missing $70 - n$ readings of the sequence with zero vectors.

5.1.2. Experimental Results

We analyzed the recognition accuracy of the different classification methods when using only the first n measurements of the recorded samples, $n \in \{1, \ldots, 70\}$. The results of this experiment are visualized in Figure 5.2. The nearest neighbor approach and length dependent LSTM perform similarly in the beginning and can recognize most activities with only a few readings. The mixed length LSTM achieves similar prediction accuracy after around 10 readings. A few of the motions follow identical trajectories until around the 40th measurement and are non-separable up to this point. The rest of the motions get classified correctly at this point. Soon after the 40th measurement, the nearest neighbor approach recognizes all samples correctly. The length dependent approach requires a few measurements more than the nearest neighbor model to reach 100% accuracy. The mixed length LSTM needs about 10 measurements more than the nearest neighbor based model to achieve the same result. The spikes in the plot of length dependent LSTM are caused by the random initialization of the training process. The mixed length LSTM needs to encode more information in its parameters and is therefore less expressive than the length dependent LSTMs.

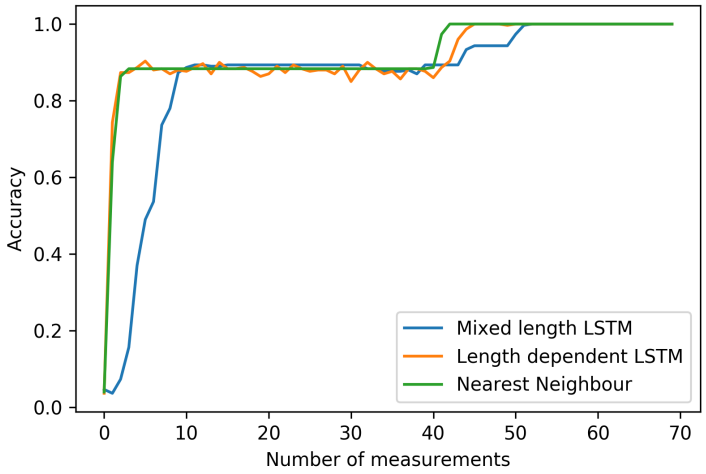


Figure 5.2.: Accuracy over number of observed measurements.

5.2. Recognizing Base Movements

In this Section, the activity recognition pipeline proposed in Section 4.3 is evaluated. We learn to recognize a set of base motions of the Pepper and Nao robot by utilizing synchronized joint state, electrical current and IMU data (see 4.2.2). For both robots, a set of 7 different activities, namely standing, moving forward, moving backward, moving left, moving right, rotating clockwise and rotating counterclockwise, is recognized. First, a script is executed to collect annotated data from the robots performing a movement sequence in repetition (shown in Figure 5.3). Multiple recordings of robot activity are then utilized in order to train a scaler and model for the pipeline as discussed in Section 4.4. We evaluate the recognition accuracy of the pipeline on the two robots and discuss our findings.

5.2.1. Recorded Data

We train scaler and model in order to recognize the different movement activities contained in the sequence shown in Figure 5.3. In this sequence, the robot performs a full clockwise rotation, moves forward, moves right, performs a full counterclockwise rotation, moves backwards and finally moves left to its initial position. During each directional movement, Pepper and Nao move 2m and 1m respectively. After each of the individual movements, the robots stand still for 2.5 seconds.

Before the beginning of the recording process, ROS is started. Afterwards, the preprocessing and synchronization stage of the recognition pipeline are initialized to provide access to one combined stream of sensor data containing information about joint states, electrical current and IMU. For this experiment, the synchronization module interpolates combined sensor readings at 10Hz. The training script and recording nodes are started to begin the collection process (see 4.4.1). While running the script lets the robot perform repetitions of the described activity sequence. The

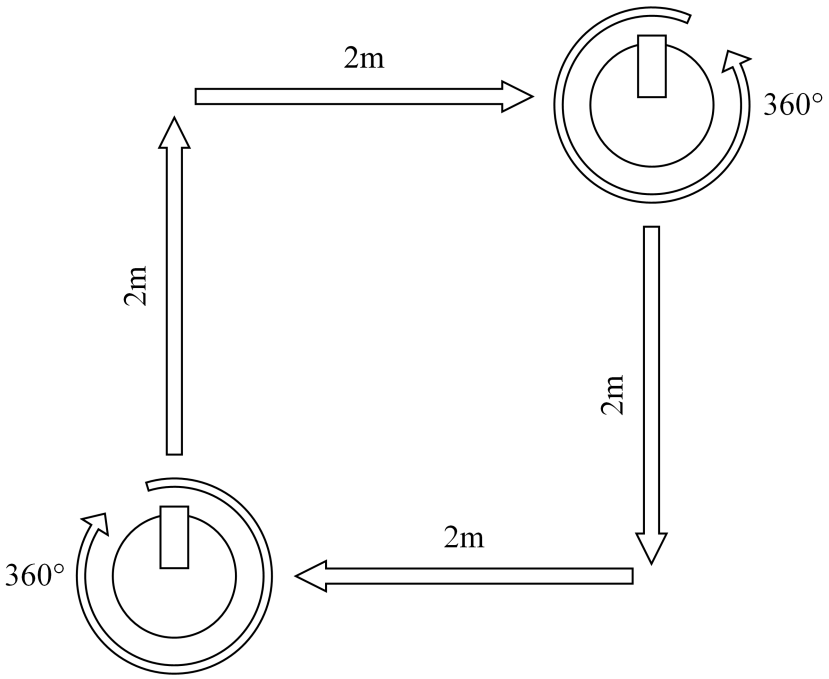


Figure 5.3.: The recording sequence from Pepper. Nao only moves 1m during each movement.

Pepper robot performs 5 repetitions and Nao 2 repetitions during each recording. The generated stream of combined sensor data is collected together with the annotation information published by the training script. Both are then stored in a database. In total, 10 executions of the script were collected for each robot. Each recording process results in one database with training data. The duration of one individual recording for the Pepper robot is a little above 5 min and for the Nao robot around 4 min.

During the recording process, it is crucial that a human observer verifies the correct execution of the activity sequence. In our experiments, the robots sometimes ignored single commands given by the control script. This corrupted the training data because it led to wrongly annotated data. For example, the robot kept standing still, but the respective sensor readings were annotated as moving forward. Therefore, it was necessary to repeat the respective recording after the robot skipped one command.

After the collection of sensor data and annotations, a converter script which labels the individual readings based on their timestamps is executed . The script outputs a pandas DataFrame containing annotated training data which is then serialized and stored for later use. Appendix A.3 visualizes a part of the resulting training data recorded from the Pepper robot as a heatmap. On it, the moving and standing parts can be differentiated by eyesight.

5.2.2. Evaluation Pepper

We recognize a set of base movements executed by the Pepper robot. Each of the 10 previously collected recordings contains a little above 3000 annotated readings of

combined sensor data where each reading consists out of 50 attributes (for an overview see A.3). 7 recordings are used as training and the remaining 3 as test set. The neural network is trained with overlapping windows containing 5 consecutive sensor readings each. In the evaluation process, we train with different combinations of sensor data. Therefore, we filter out the attributes provided by a single sensor. For example, if we say we train with electrical current data, we only look at the attributes that describe the current that acts on the individual joints. During each training process, the model is optimized over 20 epochs with a batch size of 100. For more information about model and scaler, refer to Section 4.3.

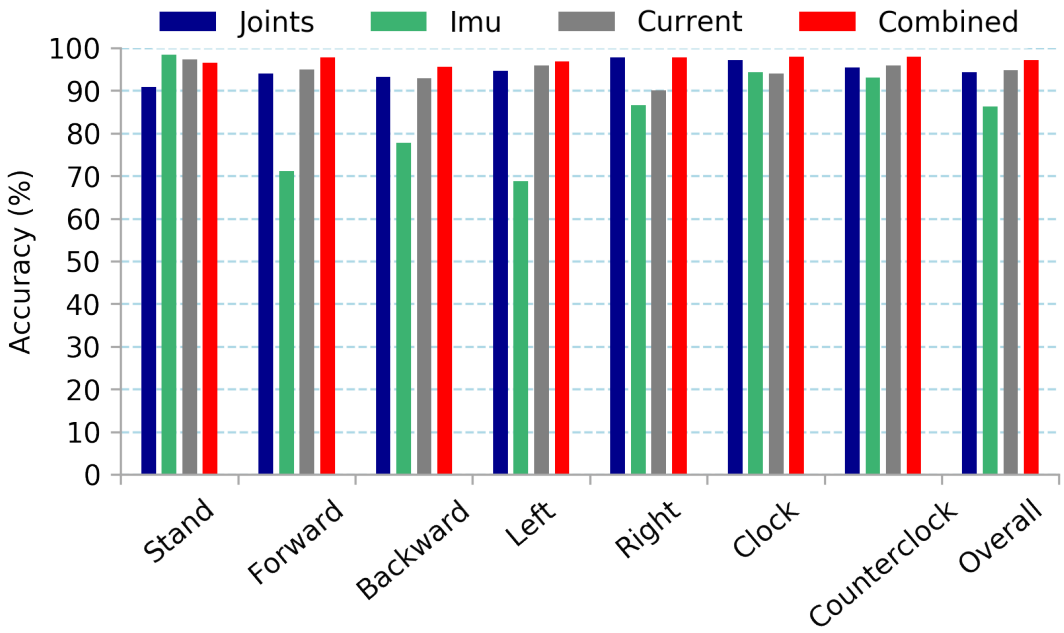
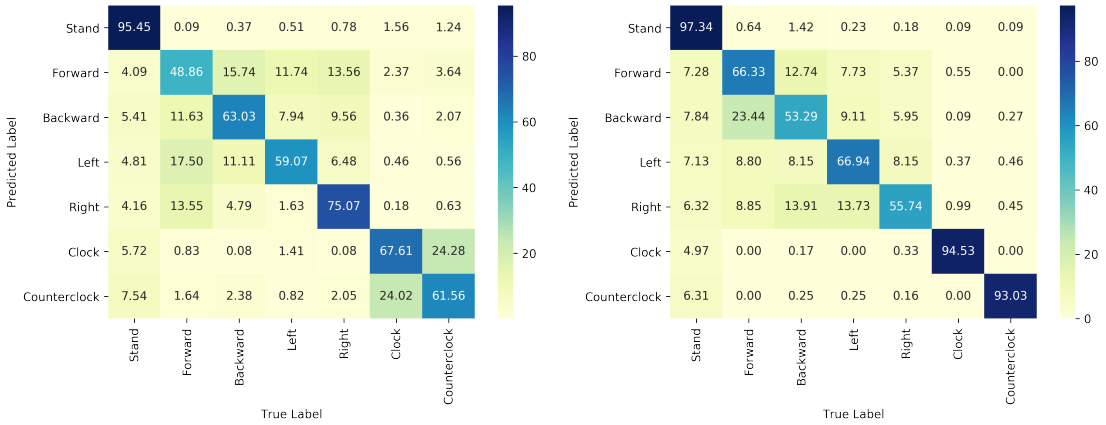


Figure 5.4.: The recognition accuracy achieved on Pepper's base movements when using different sensor data. The multimodal model achieves higher accuracy than the single sensor models.

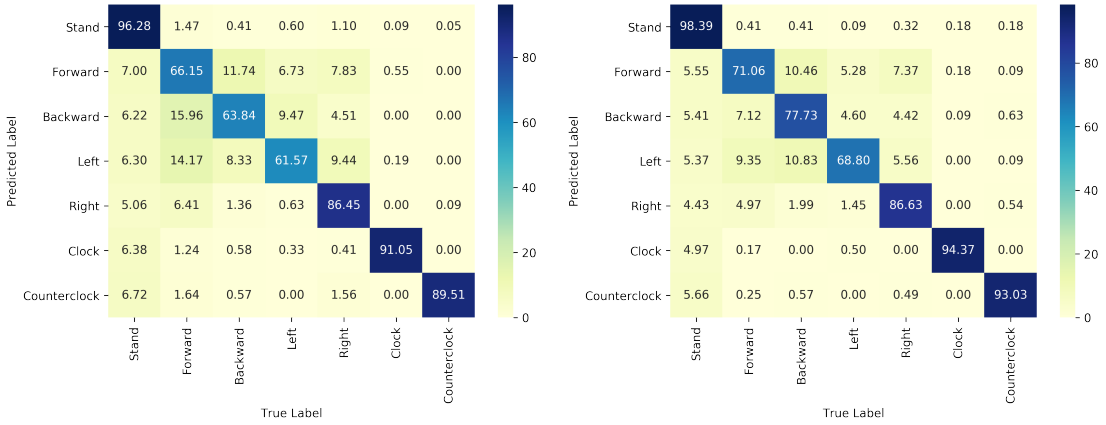
First, we train on the complete activity sequences in various configurations. We learn single-sensor models that only use joint states, IMU and electrical current data respectively and one model that uses combined data. In the following, we will call them joint, IMU, current and combined model respectively. The per-class and overall accuracies achieved by the trained models are visualized in Figure 5.4. The combined model achieves an overall accuracy of 97.12%, which is higher than the ones achieved by the joint (94.32%), IMU (86.24%) and current (94.78%) model. While the combined model outperforms the others in terms of overall accuracy, there are differences in the individual class accuracies. The joint and current model perform similarly except when it comes to recognizing the standing and right moving robot. The IMU model achieves a lower overall accuracy than the other models, but achieves the highest accuracy for the standing activity. It is worth noticing that the IMU model achieves good results on the rotations while achieving worse results on the

directional movements. In the following, we take a closer look at the behaviour of the neural network when using different parts of the IMU readings.



(a) Acceleration

(b) Angular Velocity

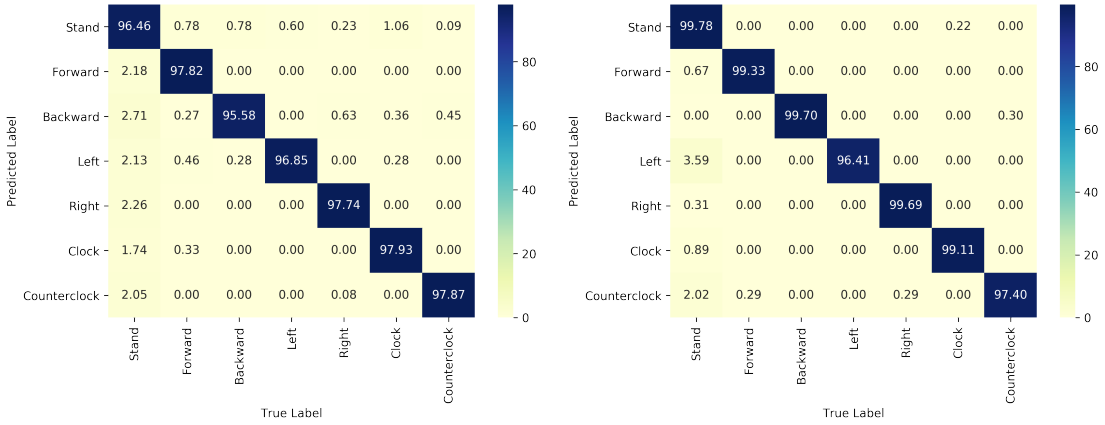


(c) Orientation

(d) Combined IMU

Figure 5.5.: The above confusion matrices give insight into models that are trained on partial and combined IMU data. Acceleration alone seems insufficient to determine the rotation motions reliably. The use of combined IMU data outperforms the other models.

Pepper's IMU provides 3 types of proprioceptive sensor data, namely acceleration, angular velocity and orientation. Figure 5.5 shows the confusion matrices of models that are trained with acceleration, angular velocity, orientation and combined IMU data respectively. The model that is only trained with acceleration data can't differentiate clearly between clockwise and counterclockwise rotations of the robot. The per-class accuracy of the combined IMU model is higher than the ones of the other models on every class. One exception to this is the angular velocity model which achieves an equal accuracy on the counterclockwise rotation and a slightly higher accuracy on the clockwise rotation. This suggests that a combination of heterogenous sensor data can lead to more accurate recognition results.



(a) With Transitions (b) Without Transitions

Figure 5.6.: Left the confusion matrix when trained and evaluated on the full recordings from Pepper. On the right we removed the first and last 0.5s of each activity.

Looking at the confusion matrices from the different IMU models and the previous combined model that utilizes all 50 attributes (see Figure 5.6a), we see that the models have some difficulties with making a clear differentiation between the standing and moving robot. One reason for this is the unclear start of an activity. The training script sends the annotation information immediately before sending the movement blocking command to the robot. While both are sent almost instantaneous, it takes about 50-200ms before the robot starts to move. This leads to an unclear cut between the activities. Also, during the transitions, the different activities might seem similar to each other. To substantiate this theory, we compare the multimodal model from our previous analysis with another model that is trained and tested without transitions. For this purpose, we removed the first and last 0.5s of each activity contained in the training and test sequences. As shown in Figure 5.6, it is easier to differentiate between the classes when one ignores the transition phases.

5.2.3. Evaluation Nao

Analogously to the previous experiment with the Pepper robot, we recognized the same set of base movements executed by the Nao robot. Here, each of the 10 collected recordings contains around 2400 annotated readings of combined sensor data where each reading consists of 65 attributes (for an overview see A.4). 7 recordings are used as training and the remaining 3 as test set. Likewise to the previous experiment, the neural network trained with overlapping windows containing 5 consecutive sensor readings each. We trained models that use different combinations of attributes. As in the Pepper experiment, the models were optimized over 20 epochs with a batch size of 100.

During this experiment, we encountered some mechanical issues which led the Nao robot to uneven movement behaviour and sometimes even to fall. For example, the robot sometimes just kept swinging sideways for a while when issued a sideward movement command. During this time, even a human observer had difficulties classfying the direction of the motion.

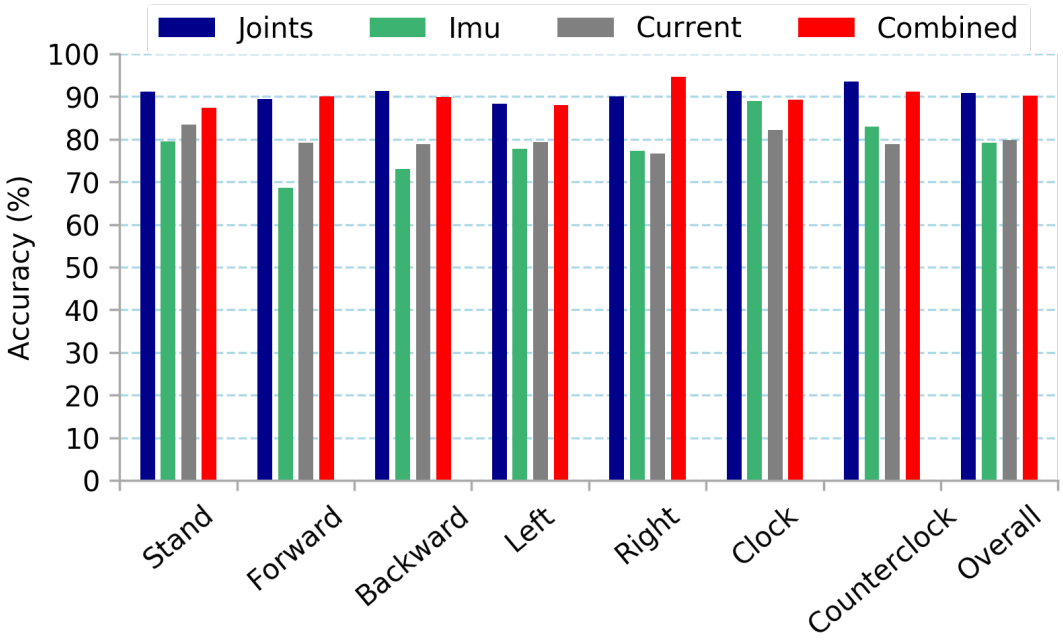


Figure 5.7.: The accuracy recognizing achieved on Nao's base movements when using different sensor data.

Again, we first trained on the complete activity sequences in various configuration. We learned single-sensor models that only use joint states, IMU and electrical current data respectively and one model that uses combined data. The per-class and overall accuracies achieved by the trained models are visualized in Figure 5.7. The joint model achieved the highest overall accuracy with 90.76% followed by the combined (90.19%), current (79.75%) and IMU (79.24%) model. The joint models achieved the highest per-class accuracy on all classes except for the forward and right movement, where the combined models achieved the highest accuracy. The combined model seems to have difficulties extracting additional information out of the additional IMU and current data. A reason for this could be the higher variance in the training data when compared to the Pepper data.

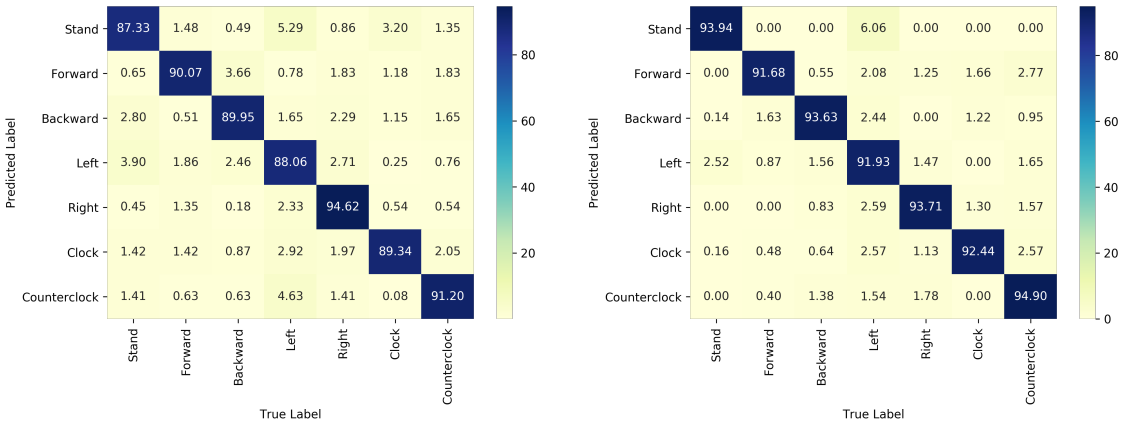
The current model achieved a higher per-class accuracy on the forward and backward movement than the IMU model. Meanwhile, the IMU model achieved a higher per-class accuracy on clockwise and counterclockwise rotations. We trained a model which combines current and IMU data to benefit from the strength of both types. This model achieved an overall accuracy of 84.75% and therefore outperforms the unimodal models by 5 percent points. Furthermore, it achieved a higher per-class accuracy on all classes except standing where the current model is better by a small margin.

The individual per-class and overall accuracies are listed in Table 5.1. This again suggests that a model which combines data from heterogenous sensor can achieve better recognition accuracy than unimodal approaches.

	IMU	Current	IMU + Current
Stand	79.51	83.46	83.07
Forward	68.53	79.24	86.05
Backward	72.97	78.93	79.64
Left	77.71	79.36	82.95
Right	77.20	76.63	80.20
Clock	88.87	82.23	91.07
Counterclock	82.98	78.85	87.59
Overall	79.24	79.75	84.75

Table 5.1.: Accuracy of IMU and current and combined model.

Similarly to the experiment with the Pepper robot, the confusion matrix (see Figure 5.8a) of the model that uses all 65 attributes shows that it has some difficulties making a clear differentiation between the standing and moving robot. Furthermore, it is more difficult to distinguish between the individual movements compared to the Pepper robot. Again, a new multimodal model was trained and tested by removing the first and last 0.5s of each activity contained in the training and test sequences. The new model achieved higher overall accuracy (93.18%) and higher per-class accuracy for all motions except the right movement.



(a) With Transitions (b) Without Transitions

Figure 5.8.: Left the confusion matrix when trained and evaluated on the full recordings from Nao. On the right we removed the first and last 0.5s of each activity.

5.3. Detecting human interference

In this experiment, we analyze the output of the recognition pipeline in case of unexpected robot behaviour. Here, a human interferes with the forward movement of the Pepper robot by dragging its base backwards. We show that the pipeline provides an output which can be used to detect a problem in the activity execution. This can allow a robot to detect deviations in the expected activity execution on its own and report it to a remote human operator.

5.3.1. Recorded Data

The Pepper robot executes a simple control script. It first stands still for 5 seconds, then moves 3m forward and concludes the activity by standing still for another 5 seconds. A simple control script sends the corresponding commands to the robot and publishes annotation information in parallel. The synchronization layer of the robot publishes combined sensor readings which contain joint state, IMU and electrical current information. Both sensor data and annotation information is collected and stored in a database. Later, the readings are labeled with the annotation information to reflect the state of the control. The human watches the robot while it stands still first. After the machine has executed about half of its forward movement, he grabs the base of the robot in a way that is not detected by the NAOqi operating system and pulls it backwards.

5.3.2. Experimental Results

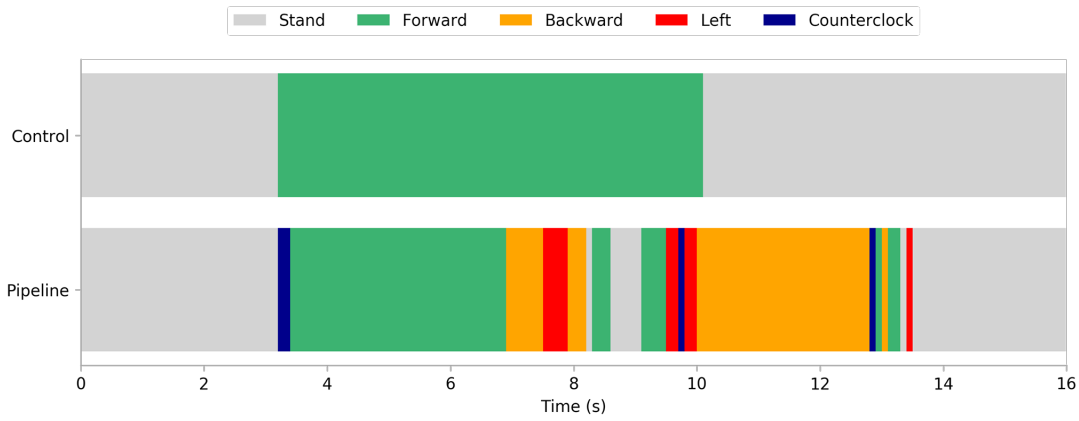


Figure 5.9.: The robot's activity over time as seen by the control and pipeline.

We compare the information from the control layer with the recognition results of the pipeline that analyzes the prerecorded synchronized sensor data. In this experiment, the pipeline uses a scaler and model that were both trained with the 10 Pepper recordings from 5.2.2. Figure 5.9 shows the robot's current activity as seen by the control and recognition pipeline over time. The pipeline recognizes the

standing activity correctly. At the transition between standing and forward movement, it makes two wrong predictions (200ms) and then recognizes the forward movement correctly. As discussed in 5.2.2 and 5.2.3, the neural network has issues recognizing state transitions because of their ambiguity. At around second 7, the human observer interferes with the forward movement of the robot by pulling its base backwards. The control does not detect the interference and thinks the robot is still moving forward. Later, the control issues a standing command and thinks the machine is standing still while in reality it is still moved by the human. The pipeline recognizes the change of the robot's body activity and classifies it first as backward, then left and then backward again. Then it swings between standing, moving forward, moving left and counterclockwise, followed by a long pull backwards. The human leaves the robot in a standing position at around second 13. The model is noisy during the transition and then recognizes the standing robot. The noisiness in the recognitions reflects the way the robot is manipulated by the human.

In a system implementation, the output of the recognition pipeline can be compared to what the robot's control is doing. If control and pipeline do not agree on the same activity for a certain amount of time (e.g. 0.5s), an observation system can detect the unexpected behavior of the robot. This finding can then be used to allow the machine to communicate its problem to a remote operator who then can check on the machine.

6. Conclusion

6.1. Summary

A robot's proprioceptive sensors provide rich information about the state of the machine and the actions it performs. By leveraging streams of low-level sensor readings, one can recognize patterns that tell about the activities the machine is performing in the world. While the control layer captures what a robot is intended to do, sensors give direct insight into what its physical body is doing. Therefore, a robot can achieve a basic awareness of its own activities by analyzing the data streams provided by its proprioceptive sensors. The machine can then utilize this capability in order to verify its own activity execution or to narrate its actions to external users. Also, it might allow the machine to recognize and recover from unexpected system states.

In this work, we introduced a general activity recognition pipeline inspired by HAR methods to the robotics domain. The proposed pipeline analyzes multiple streams of asynchronous sensor data to recognize activity patterns and to classify the type of action the machine is performing. By combining readings from heterogenous sensors, one can achieve more accurate recognition results than single-sensor approaches. We discussed the pipeline architecture in detail and evaluated it in multiple experiments. Our approach can be transferred to other robots with proprioceptive sensing capabilities.

In our evaluation, we first recognized a set of one and two-arm based motions executed by a simulated Pepper robot in order to verify the suitability of a Hausdorff distance based nearest neighbor algorithm and an LSTM neural network for activity pattern recognition. We then decided to utilize the LSTM based approach for our further analysis. Next, the pipeline was evaluated by recognizing a set of 7 different base movements of the service robot Pepper and humanoid Nao. This experiment showed that models which combine readings from multiple sensors can achieve higher accuracy than single sensor approaches. In our final experiment, we analyzed the model behavior in case of unexpected events. A human interrupted the forward movement of the Pepper robot by pulling its base backwards and the model sucessfully detected the unexpected system state.

6.2. Future Work

As a potential future research direction, we want to recognize more complex activities that go beyond simple arm and base movement. Especially the application of plan detection, where the robot executes a defined sequence of activities, is interesting

to us. Here, we want to analyze how the current activity that a robot executes fits into a more meaningful context. Another research question is to create more detailed descriptions of a robot's actions. For example, one could recognize parallel activities such as moving forward while raising the left hand.

During our work with the Nao robot, we encountered some mechanical issues because of its engines calibration. This led the robot to uneven movement behavior, sometimes even to fall. It would be desirable to repeat this experiment on a Nao robot that moves more evenly. Also, the fact that the combined model that utilizes joint states, IMU and electrical current data achieved worse recognition accuracy than the model that only utilizes joint states motivates further investigation. Furthermore, we need to analyze the resource consumptions of our models regarding their deployment on a robot.

Furthermore, it is of interest to create models that are capable of analyzing sequences of proprioceptive sensor readings that can capture the complete deployment. During our experiments we worked with sequences of 5 consecutive sensor readings. The reason for this small number mainly lies in the types of activities that were recognized in the experiments. We would like to derive compact descriptions about the robot's activities directly out of low-level sensor logs. A starting point would be to detect if the robot had to drive around an obstacle instead of taking the direct path to a goal.

Bibliography

[1] Halim Alwi, Christopher Edwards, and Chee Pin Tan. *Fault detection and fault-tolerant control using sliding modes*. Springer Science & Business Media, 2011.

[2] Akin Avci et al. "Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey". In: *Architecture of computing systems (ARCS), 2010 23rd international conference on*. VDE. 2010, pp. 1–10.

[3] Akram Bayat, Marc Pomplun, and Duc A Tran. "A study on human activity recognition using accelerometer data from smartphones". In: *Procedia Computer Science* 34 (2014), pp. 450–457.

[4] Yoshua Bengio. "Deep learning of representations: Looking forward". In: *International Conference on Statistical Language and Speech Processing*. Springer. 2013, pp. 1–37.

[5] Gerald Bieber, Marian Haescher, and Matthias Vahl. "Sensor requirements for activity recognition on smart watches". In: *Proceedings of the 6th International Conference on PErvasive Technologies Related to Assistive Environments*. ACM. 2013, p. 67.

[6] Saisakul Chernbumroong et al. "Elderly activities recognition and classification for applications in assisted living". In: *Expert Systems with Applications* 40.5 (2013), pp. 1662–1674.

[7] Maria Cornacchia et al. "A survey on activity detection and classification using wearable sensors". In: *IEEE Sensors Journal* 17.2 (2017), pp. 386–403.

[8] Thomas Cover and Peter Hart. "Nearest neighbor pattern classification". In: *IEEE transactions on information theory* 13.1 (1967), pp. 21–27.

[9] Can Erdogan and Manuela Veloso. "Action selection via learning behavior patterns in multi-robot domains". In: *Proc. International Joint Conference on Artificial Intelligence*. 2011, pp. 192–197.

[10] Open Source Robotics Foundation. *Pepper*. 2017. URL: `http://wiki.ros.org/pepper`.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[12] Sami Haddadin, Alessandro De Luca, and Alin Albu-Schäffer. "Robot collisions: A survey on detection, isolation, and identification". In: *IEEE Transactions on Robotics* 33.6 (2017), pp. 1292–1312.

[13] Yi He and Ye Li. "Physical activity recognition utilizing the built-in kinematic sensors of a smartphone". In: *International Journal of Distributed Sensor Networks* 9.4 (2013), p. 481580.

[14] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[15] Michiel de Jong et al. "Towards a Robust Interactive and Learning Social Robot". In: *Accepted for AAMAS* (2018).

[16] Eliahu Khalastchi and Meir Kalech. "On Fault Detection and Diagnosis in Robotic Systems". In: *ACM Computing Surveys (CSUR)* 51.1 (2018), p. 9.

[17] Nathan Koenig and Andrew Howard. "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator". In: *In IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2004, pp. 2149–2154.

[18] Kui Liu et al. "Fusion of inertial and depth sensor data for robust hand gesture recognition". In: *IEEE Sensors Journal* 14.6 (2014), pp. 1898–1903.

[19] C. Mandery et al. "The KIT whole-body human motion database". In: *Proc. Int. Conf. Advanced Robotics (ICAR)*. July 2015, pp. 329–336. DOI: 10.1109/ICAR.2015.7251476.

[20] Ferda Ofli et al. "Berkeley MHAD: A comprehensive multimodal human action database". In: *Applications of Computer Vision (WACV), 2013 IEEE Workshop on*. IEEE. 2013, pp. 53–60.

[21] Francisco Javier Ordóñez and Daniel Roggen. "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition". In: *Sensors* 16.1 (2016), p. 115.

[22] Vittorio Perera et al. "Setting Up Pepper For Autonomous Navigation And Personalized Interaction With Users". In: *CoRR* abs/1704.04797 (2017).

[23] Ola Pettersson. "Execution monitoring in robotics: A survey". In: *Robotics and Autonomous Systems* 53.2 (2005), pp. 73–88.

[24] E. Pot et al. "Choregraphe: a graphical tool for humanoid robot programming". In: *Proc. RO-MAN 2009 - The 18th IEEE Int. Symp. Robot and Human Interactive Communication*. Sept. 2009, pp. 46–51. DOI: 10.1109/ROMAN.2009.5326209.

[25] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA Workshop on Open Source Software*. 2009.

[26] Valentin Radu et al. "Towards multimodal deep learning for activity recognition on mobile devices". In: *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*. ACM. 2016, pp. 185–188.

[27] SoftBank Robotics. *Pepper - Developer Guide*. 2017. URL: http://doc.aldebaran.com/2-5/family/pepper_technical/index_dev_pepper.html.

[28] Raúl Rojas. “Neural Networks-A Systematic Introduction Springer-Verlag”. In: *New York* (1996).

[29] Muhammad Shoaib et al. “A survey of online activity recognition using mobile phones”. In: *Sensors* 15.1 (2015), pp. 2059–2085.

[30] Jindong Wang et al. “Deep learning for sensor-based activity recognition: A survey”. In: *arXiv preprint arXiv:1707.03502* (2017).

[31] Jianbo Yang et al. “Deep Convolutional Neural Networks on Multichannel Time Series for Human Activity Recognition.” In: *IJCAI*. 2015, pp. 3995–4001.

[32] Tong Zhang et al. “Fall detection by embedding an accelerometer in cellphone and using KFD algorithm”. In: *International Journal of Computer Science and Network Security* 6.10 (2006), pp. 277–284.

[33] Zhongtang Zhao et al. “Cross-people mobile-phone based activity recognition”. In: *IJCAI*. Vol. 11. 2011, pp. 2545–250.

A. Appendix

A.1. Joint Conversion

Real	Simulated
HeadYaw	HeadPitch
HeadPitch	HeadYaw
LShoulderPitch	HipPitch
LShoulderRoll	HipRoll
LElbowYaw	KneePitch
LElbowRoll	LElbowRoll
LWristYaw	LElbowYaw
LHand	LHand
HipRoll	LShoulderPitch
HipPitch	LShoulderRoll
KneePitch	LWristYaw
RShoulderPitch	RElbowRoll
RShoulderRoll	RElbowYaw
RElbowYaw	RHand
RElbowRoll	RShoulderPitch
RWristYaw	RShoulderRoll
RHand	RWristYaw
WheelFL	WheelB
WheelFR	WheelFL
WheelB	WheelFR

Table A.1.: Order of joint states vector for the simulated and real Pepper

Each joint state message published by the Pepper robot contains an array with 20 floating point values that relate to the individual joints of the robot. While switching from the Gazebo version of Pepper to the real robot we noticed that the order in which the arrays describe the joint states is different. Table A.1 shows the order in which the joints are described on the real and simulated robot. Because of this, we implemented a converter that transforms the readings from the simulation to the format of the real robot.

A.2. Model Code

```
embedding_vecor_length = 50 # Pepper generates 50 attributes
n_classes = 7 # There are 7 activities to predict
window_size = 5 # 5 readings are used for classification

epochs = 20
batch_size = 100

reg = L1L2(l1=0.05, l2=0.05)
model = Sequential()

# first LSTM layer
model.add(LSTM(32, return_sequences=True,
               input_shape=(window_size, embedding_vecor_length),
               batch_size=batch_size, bias_regularizer=reg))
model.add(BatchNormalization())

# second LSTM layer
model.add(LSTM(32, bias_regularizer=reg))
model.add(BatchNormalization())

# softmax output layer
model.add(Dense(output_dim=n_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])

model.fit(train_X, train_Y, validation_data=(test_X, test_Y),
          epochs=epochs, batch_size=batch_size, validation_split=0.25)
```

Listing A.1: The code for the neural network.

The above code describes the neural network used in our implementation of the activity recognition pipeline (4.3). We used the deep learning framework Keras together with a TensorFlow backend for our experiments. The length of the embedding vector varies depending on the number of sensor attributes that are used for the activity recognition.

A.3. Collected Data Pepper

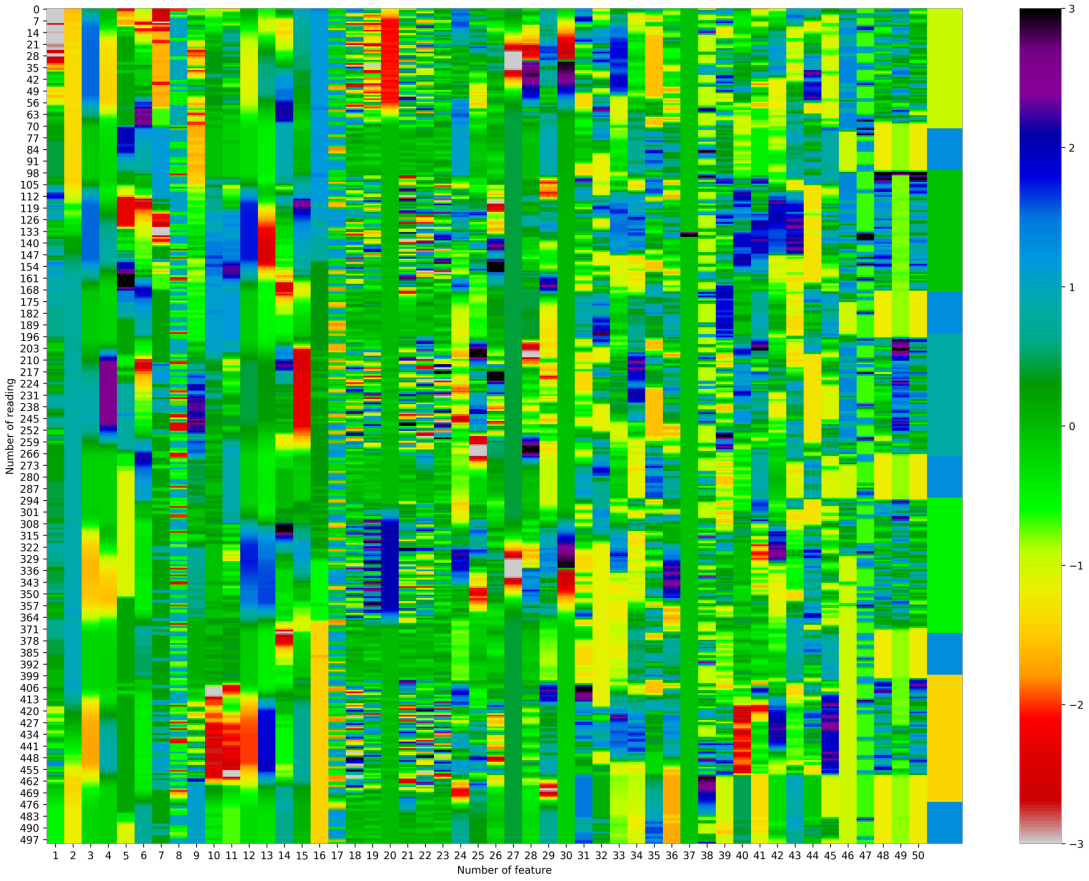


Figure A.1.: A visualization of the recorded data from the Pepper robot. The rightmost column indicates the performed activity.

ID	Attribute Name	ID	Attribute Name
1	j_HeadYaw	26	BaseLinearAccelerationLowZ
2	j_HeadPitch	27	BaseOrientationW
3	j_LShoulderPitch	28	BaseOrientationX
4	j_LShoulderRoll	29	BaseOrientationY
5	j_LElbowYaw	30	BaseOrientationZ
6	j_LElbowRoll	31	c_HeadPitch
7	j_LWristYaw	32	c_HeadYaw
8	j_LHand	33	c_LShoulderPitch
9	j_HipRoll	34	c_LShoulderRoll
10	j_HipPitch	35	c_LElbowYaw
11	j_KneePitch	36	c_LElbowRoll
12	j_RShoulderPitch	37	c_LWristYaw
13	j_RShoulderRoll	38	c_LHand
14	j_RElbowYaw	39	c_HipRoll
15	j_RElbowRoll	40	c_HipPitch
16	j_RWristYaw	41	c_KneePitch
17	j_RHand	42	c_RShoulderPitch
18	BaseAngularVelocityX	43	c_RShoulderRoll
19	BaseAngularVelocityY	44	c_RElbowYaw
20	BaseAngularVelocityZ	45	c_RElbowRoll
21	BaseLinearAccelerationHighX	46	c_RWristYaw
22	BaseLinearAccelerationHighY	47	c_RHand
23	BaseLinearAccelerationHighZ	48	c_WheelFR
24	BaseLinearAccelerationLowX	49	c_WheelB
25	BaseLinearAccelerationLowY	50	c_WheelFL

Table A.2.: The attributes that were used for the Pepper robot.

A.4. Collected Data Nao

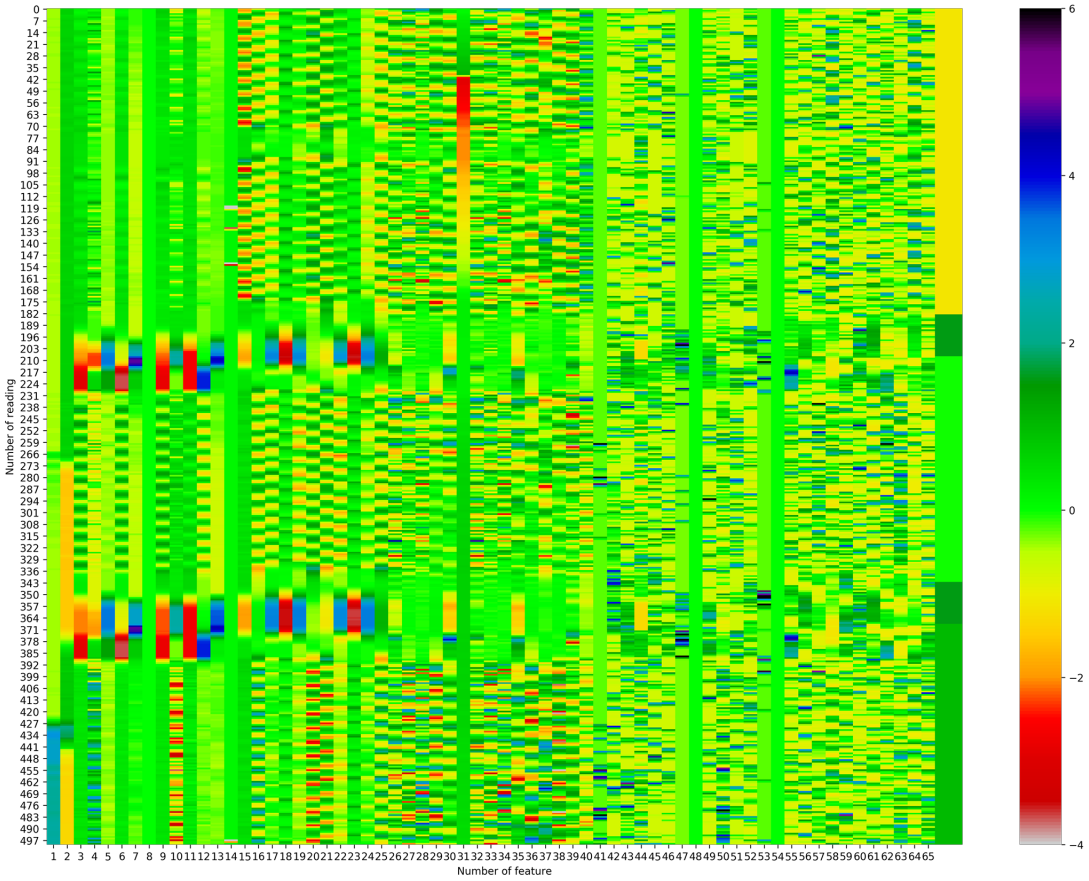


Figure A.2.: A visualization of the recorded data from the Nao robot. The rightmost column indicates the performed activity.

ID	Attribute Name	ID	Attribute Name
1	j_HeadYaw	34	BaseLinearAccelerationHighZ
2	j_HeadPitch	35	BaseLinearAccelerationLowX
3	j_LShoulderPitch	36	BaseLinearAccelerationLowY
4	j_LShoulderRoll	37	BaseLinearAccelerationLowZ
5	j_LElbowYaw	38	GyroscopeX
6	j_LElbowRoll	39	GyroscopeY
7	j_LWristYaw	40	GyroscopeZ
8	j_LHand	41	c_HeadYaw
9	j_RShoulderPitch	42	c_HeadPitch
10	j_RShoulderRoll	43	c_LShoulderPitch
11	j_RElbowYaw	44	c_LShoulderRoll
12	j_RElbowRoll	45	c_LElbowYaw
13	j_RWristYaw	46	c_LElbowRoll
14	j_RHand	47	c_LWristYaw
15	j_LHipYawPitch	48	c_LHand
16	j_LHipRoll	49	c_RShoulderPitch
17	j_LHipPitch	50	c_RShoulderRoll
18	j_LKneePitch	51	c_RElbowYaw
19	j_LAnklePitch	52	c_RElbowRoll
20	j_LAnkleRoll	53	c_RWristYaw
21	j_RHipRoll	54	c_RHand
22	j_RHipPitch	55	c_LHipYawPitch
23	j_RKneePitch	56	c_LHipRoll
24	j_RAnklePitch	57	c_LHipPitch
25	j_RAnkleRoll	58	c_LKneePitch
26	AccelerometerX	59	c_LAnklePitch
27	AccelerometerY	60	c_LAnkleRoll
28	AccelerometerZ	61	c_RHipRoll
29	AngleX	62	c_RHipPitch
30	AngleY	63	c_RKneePitch
31	AngleZ	64	c_RAnklePitch
32	BaseLinearAccelerationHighX	65	c_RAnkleRoll
33	BaseLinearAccelerationHighY		

Table A.3.: The attributes that were used for the Nao robot.