ADAPTIVE HUMAN-ROBOT MOTION TRANSFER FOR COMPLETE BODY IMITATION

By

FRANCISCO VILLA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Computer Engineering

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2021

To my grandmother Doloros Mucek who taught me to think critically and opened my mind. And my mother and father who raised and supported me.

Acknowledgements

I would like to thank my supervising professor Dr. Manfred Huber for our long and interesting discussions, for his patience, and his wealth of knowledge. These discussions provided basis to conduct additional research and gave me many new in-sights. He provided me with an example to live up to. Patience is something Dr. Manfred Huber greatly excels at, and I am extremely thankful for it.

I wish to thank my committee members Dr. David Levine and Dr. Farhad Kamangar for attending my defense committee, being extremely flexible with their schedules, and asking enthusiastic questions. Without their assistance, I would not be progressing in my life.

Finally, I would like to express my deep gratitude to my grandmother who has encouraged and inspired me. To this day she still challenges my assumptions. She taught me to think of my past, present, short, mid, long-term future. I am extremely fortunate to be so blessed. I am also extremely grateful to my mother, and father for their encouragement and patience. Graduate school was a long intense ride.

December 16, 2021

ABSTRACT

ADAPTIVE HUMAN-ROBOT MOTION TRANSFER FOR COMPLETE BODYIMITATION

By

Francisco Villa, Masters

The University of Texas at Arlington, 2021

Supervising Professor: Manfred Huber

Programming robot systems to perform certain tasks is a big challenge especially if such programming is to be performed by persons who are not experts in robotics. For example, when programming a robot to serve as an exercise trainer, the person defining the motions might more naturally be a person in the exercise domain rather than a robotics expert. To address this, this thesis investigates programming by demonstration or teleoperation using full direct body motion. The goal is to reproduce gaits, gestures, and postures on a humanoid robot from observed human demonstrations. Fine motor movements such as movement of fingers will be excluded from this thesis' paradigm.

Mimicking said such movements is straight forward if human and robot dynamics and kinematics are the same. The robot can move exactly like how his human demonstrator moves. This, however, is never the case, thus leading to a multidimensional correspondence problem. In this thesis an approach is presented that attempts to resolve this by addressing four components.

First it addresses linking the degrees of freedom of two similar but different bodies, the human, and a humanoid robot. Second, it handles linking the image frame related to the observation of the demonstrator to the demonstrator structure to be able to track locations that have changed, using

3D computer vision with a human skeleton model. As such skeleton observations are usually noisy, a third process is aimed at of filtering out sensor noise and recognition errors, resulting in an observed motion trajectory. Lastly, to account for the differences between the human and robotic bodies both in terms of kinematics and dynamic stability, a modeling and learning framework, PILCO, is adapted to address mapping into an executable imitation that obeys the stability requirements and limitations of the humanoid robot system.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1 - INTRODUCTION

1.1 Learning from Demonstration

Over time there have been several trends in robot systems. For systems that operate in human environments and thus must be able to perform human-type tasks, one of these has traditionally been to make a robotic system as like a human body as possible, stemming from the intuition that it should be easier to program them, and the resulting capabilities should be as close to the ones of a human as possible. However, perfectly mimicking capabilities of a human might in the extreme require building robots with muscles and nerves instead of gears and motors, making it for the moment unachievable. However even if that technology was developed to the point like human muscles and nerves, other technologies, including battery technology needs to be developed that would be able to operate this system reliably and for similar periods of time. At the current state of progression of those three fields robots cannot be similar enough to directly mimic human movements directly. As a result, building a robot system that perfectly mimics the kinematic and dynamic properties of the human body is currently not achievable and thus most humanoid robots, while outwardly appearing human-like, have significant differences in terms of kinematics and in terms of dynamic properties. This has consequences in terms of their control as well as in the possibility to directly transfer human actions to the system with direct consequences for imitation and programming by demonstration.

Programming by demonstration (or imitation) is a very powerful approach to programming robots as it can be used by a domain expert in the field in which the system has to operate without the

need for that domain expert also to be an expert at the robot or robot programming. Instead, the robot can be programmed by just demonstrating the task in the intended setting. However, as bodies as well as the environmental settings are not identical, attempting to mimic another one must consider carefully what to reproduce. First off when considering preserving the movements of another, one must keep in mind that there are no two exact same bodies. Thus, objectives must be considered [1]. This leads to several questions that need to be addressed ant that offer different options depending on the task for which imitation is intended:

Which to imitate?

- If there are multiple demonstrations with different styles, there is a requirement to handle which demonstrations are appropriate.
    - An averaging strategy?
        - Positions, velocities, accelerations are all averaged
    - Should there be a categorical separation of demonstrations between expert and novice?
        - Categorical separation be completed by user or done by machine learning methods
    - A more statistical intuitive approach?
        - Gaussian process approach?
            - A Search of demonstrations of similar states and actions are higher weighted than outliers.

When to imitate?

- Does a specific arm motion make a normally stable posture unstable?
    - When is an appropriate time to reproduce this gesture?
        - Before stability is found
        - During stabilization
        - After stabilization
- Should the gesture be retained and the posture modified to maintain stability?
    - When should this occur?
        - Before
        - During
        - After

What to imitate?

- What segments of motions should be prioritized?
    - Should the user have control of this?
    - Should we rely on AI to guess what is important?

How to evaluate the human demonstration?

- Human Stability?
- Robot Stability?
- Which human demonstrations are expert and which are novice?

How to evaluate robot imitation?

- How much of the human demonstration exceeds the robotic joint constraints?
- How much of the human demonstration was persevered?

Correspondence Problem

- How to mimic movements of a body similarly but not exactly in a numerical way?

Considering these questions modulates what the imitation process looks like and tries to achieve. In some situations, imitation is aimed at capturing the outward appearance (such as in many arts such as ballet or performing arts) while sometimes it is aimed purely at capturing the functional results of the demonstration (such as in cleaning or care giving tasks). However, under all these settings it is a useful and very potent means of robot programming. “Monkey see, monkey do” is a very crude and direct saying to reinforce how powerful imitation learning is. In it observing a sequence of states leads to a sequence of actions that need to be completed to accomplish this goal. If a task is being observed, ideally several times, the robot target would attempt to replicate the task.

This thesis develops a set of tools to allow imitation in the context of external appearance-oriented tasks from human demonstration by a humanoid robot target while maintaining certain stability requirements that arise from the different dynamics of the target system. This thesis develops an application to extract the kinematic configuration for a NAO [2] robot that closely mimics a human demonstration from a 3D vision sensor while obeying the kinematic configuration of the robot. Using this as a starting point it then proposes a way to adapt the resulting kinematic task sequence to also obey dynamic stability constraints of the robot.

The remainder of this thesis initially discusses some related work before presenting background and used formalisms in Chapter 2. Chapter 3 then introduces the techniques used for motion mapping from 3D vision to the robot platform, for motion filtering to address observation noise, and for motion control adaptation to address kinematic and dynamic dissimilarities in the platform and achieve stability.

1.2 Related Work

To address the imitation problem, [4] identifies three important areas, gait generation, motion control, and motion filtering.

1.2.1 - Gait Generation

Gait generation is the formation or selection of a sequence of coordinated leg and body motions that propel a legged robot along a desired path [5]. There are a multitude of methods that were developed over the years to generate gaits.

A first method and one of the most common and direct is breaking down the robotic system into localized links and joints and solving trigonometric functions on a unit circle. As most humanoid robotic joints involve revolute joint positions further explanations will be provided in Chapter 2. To characterize the effects of trigonometric functions on a unit circle, Denavit-Hartenberg parameters were invented to simplify the kinematic calculations for robot mechanisms. This framework allows to efficiently characterize the kinematic configuration of a robot system and has been used frequently, including for the NAO robot used as a target in this thesis [6].

To achieve flexible gaits, there is a method called parametric gait synthesis which uses a combination of parametric functions that calculate walking motion[7]. This has been used in a wide range of applications where examples include windup toys or video game characters, that are commanded to walk in a particular predetermined predictable motion.

On the gait perception side, techniques from computer graphics have been used to characterize observed gaits. This has become of significant interest upon the arrival of usable 3D vision systems for gait capture. Since 2011, Microsoft released an affordable human motion capture data device and software development kit (SDK). This made obtaining this data affordable and correspondingly gait capture has gained much popularity over the last decade. The Kinect offered a method to record the gait from a demonstrator in real-time. It accomplished this by drawing a pseudo skeleton over a person and record the coordinates of the persons approximate joint locations. This appeared to solve the “correspondence problem” present in stereo vision system and resulted in representations of observed gait in terms of joint locations without the need to capture joint angles which are harder to extract from observations. On the flip side, if used not for computer graphics (displaying the skeleton) but rather to command or mimic corresponding motions, the absence of kinematic configurations poses the problem of mapping the skeleton onto

an actual robot mechanism that can only move or change configuration through the underlying joints.

First introduced by Emil Post's dissertation in 1946[8]. The post correspondence problem (PCP)is simply defined is an undecidable decision problem aimed at mapping one configuration onto a related system. As an undecidable system, any algorithm or step-by-step solution to addressing it has the problem that it is not always correct. Computer Vision, especially in the context of imitation or programming by demonstration experiences the PCP in the context of mapping observed behavior onto the kinematic and dynamic structure of the robot mechanism, making it relevant to the scope of this thesis. To oversimplify the general computer vision PCP definition, it is a mapping of two or more images taken at two different vantage points or reference points and being able to identify what are same objects. Most living organisms on earth do this quiet easily with their stereo vision, while computer system tend to encounter significantly more problems when attempting to do the same.

In the early days, computer vision approaches produced many inconsistencies in mapping out locations of objects in terms of individual pixel locations relative to the image. First there is a requirement for an origin from where all the local coordinates begin. Then the chicken and egg problem of the camera locations relative to all the distances of pixels within each frame must be addressed. Where am I? Where are you? Because of this it turns out that it is quite difficult to calculate exact distances from two cameras because of the need to determine consistent feature picks in the images from the cameras. Expert knowledge and other initial knowledge can assist with the calculations but in the many situations where this is not available or ambiguous this might not be sufficient. Here extra sensory equipment might provide a cheap method to solve this

dilemma and is the approach taken in most 3D vision systems through the use either of structured light or time of flight sensors.

There are many algorithms that attempt to solve the correspondence problem in the computer vision paradigm with varying degrees of success covered in the [9] paper. This thesis builds on the computer vision framework by calculating joint angles of a human demonstrator from extracted 3D locations using the Microsoft Kinect sensor. The Microsoft Kinect solves part of the correspondence problem in respect to the computer vision domain by using an infrared sensor to map out 3 dimensional coordinates to each pixel. Building on this, one of the contributions in this thesis is solving the subsequent mapping problem between human demonstrator joint locations and robotic platform joint angles of the NAO robot using the Gram-Schmidt process discussed in Chapter 2.

1.2.2 - Motion Control

Gait generation is aimed at producing the walking motion [7]. However, in most situations resulting kinematic gaits are not stable for the dynamics of a robot or are easily disturbed. Even in these situations the gait is kinematically viable for humans with similar weight proportions but often needs to be slightly adjusted for humanoid mechanisms with slightly different kinematics and dynamics.

Motion control's domain is focused on addressing actual control goals given the dynamics of the platform. It is often closely linked to gait generation but is more of a complementary domain. Motion control modifies gait generation to achieve a desired goal, often stability.

When stability is the focus, the center of mass and the lower body, specifically the ankles and knees, are of the utmost importance as they have the most impact on stability. In the paper, [10] which introduces a study of replication of humanoid motion on a bipedal robot platform, they excluded the ankle parameter and calculated trigonometric stable solutions in relation with the hips to simplify the problem as consideration of all joints in the body is highly complex and often intractable without either decomposition of the structure or simplification.

In other studies [10] and [11] focus on more of a complete analytical solution. This paper [10] does not describe how exactly they calculated the Center of Mass (COM) but in [11] they explicitly defined it. In real world applications computing each link may be important depending on the application and the cost to maintain stable velocities and weights carried by the platform if they are significant, thus justifying the additional cost of attempting to compute control solutions involving the complete body.

Most applications for the NAO robot specifically in [10] display in their results that general COM calculations are mostly sufficient. The torques, precisions, and velocities of the NAO platform exclude the need for precise calculations. Thus, this thesis, which is also aimed at the NAO root platform, will follow the result from [10] and calculate a generalized COM.

There are cases where motion control is not focused on function and the immersed paradigm of stability. For example, [13] displays a novel “fashion model” motion. Parameters of step length, waist yaw maximum height, maximum ankle yaw and pitch, and the corresponding velocities are the main focus here, concentrating primarily on the appearance of the gait rather than its stability. Then a hybrid approach is taken with least squared, and Zero Movement Point (ZMP) criteria being applied. The goal is to preserve as much as the demonstration as possible while maintaining a stable posture.

1.2.3 – Motion Filtering Methods

While motion control is aimed at modifying or generating motions to address dynamics and function, motion filtering is observing generated trajectory paths and addressing their variability and noise from where upon constraints, dynamics, or time can be modified. Constraints such as motor torque, velocity, positions, gravity, mass, viscosity, etc. can be enhanced by filtering motions, especially when extracted from observations.

In the domain of animation, filtering methods are used frequently [14]. Filters can be applied to extract more reliable parameters such as start and goal coordinates as well as duration and then a trajectory will be calculated and animated. The motion filter is applied to smooth out the entire process.

While motion filtering is an effective process, it is expensive because of the calculations of past, present, and future trajectories [14]. Exhaustive filtering and optimization over entire trajectory sets is thus typically not viable for real time applications.

There is research that simplifies this paradigm [4]. Instead of applying complete kinematic and dynamic joint models, a full body model, lighting, and shadow observations for trajectories to the motion filter, the model could be simplified to a simple inverted pendulum when focusing on stability during trajectory execution. Thus, a complex 20+ degree of freedom (DOF) robot or figure can be reduced to a 3 DOF pendulum approximation when analyzing stance and gait stability.

Karl Mueceke's research [4] incorporates two types of filters, constrained Analytical Trajectory filtering (CATF) and COM-based motion Adaptation (CMA). The CATF's input accepts COM parameters and outputs a stable trajectory. The CMA accepts this stable COM and calculates the joint angles. At first glance, computing inverse kinematics from a stable COM is an intractable

problem for a humanoid system because of the infinite number of solutions. But Mueceke incorporated a constraint that generates single unique solutions. The initial raw reference input of joint angles that first was sent to the CATF also is also sent to CMA and solutions are biased towards the joint angle input.

While this thesis shares the goal of the last few papers described and many others, it will not make assumptions that the complete model of the system is pre-computed and instead employ reward-based optimization and thus trial and error methods. Typically, such methods are not viable for robots because of cost, time, and wear of parts as they often require thousands of trials to learn an effective policy. However, while in the domain of reinforced learning the most popular methods are model free to avoid model bias, there are techniques that are model based but can over time reduce model bias. Generally, use of these models reduces the learning time to achieve a sufficient policy and the work in this thesis will utilize such a model-based technique centered around a gaussian process model built from sample data.

CHAPTER 2 – BACKGROUND

The goal of this thesis is to develop tools to allow programming from demonstration by mapping observations from a Kinect 3D camera of a human performing motions to a NAO robot. For this, the differences in kinematic configurations between human and NAO as well as observation noise in the sensor and differences in stability constraint and body dynamic must be considered. To perform this mapping this thesis first extracts corresponding NAO kinematics from the observations requiring extracting corresponding joint angles for the NAO from the observation of the human skeleton. It then addresses irregularities resulting from sensor noise through filtering before finally addressing dynamic differences and stability through a reward-based stochastic optimization approach built on a data-driven system model.

This chapter covers the background of the utilized techniques that will be used in the next chapter when developing the details of the approach.

2.1 Kinematics

To address mapping of the observed joint positions of a human to joint angles in a. Robot we need to deal with system kinematics.

2.1.1 Unit Circle

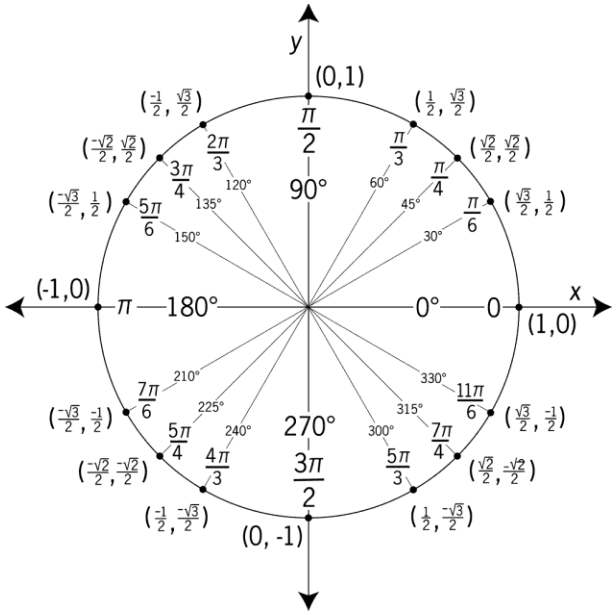


Figure 1: Unit Circle (From https://www.mometrix.com/academy/unit-circles-and-standard-position/)

The basis to all the kinematic calculations of humanoid robots starts with the trigonometric calculations on a unit circle. To compute this requires the trigonometric formulation. The information required are the locations that make up the edge of the circle and the θ or angle relative to the reference. The reference is typically the x vector pointing to the right. To calculate these positions, the calculations are given below.

$$X_{\text{Position}} = \ell * \cos(\Theta) \qquad Y_{\text{Position}} = \ell * \sin(\Theta)$$

$$(x, y) = (\ell * \cos(\Theta), \ell * \sin(\Theta))$$

If the position is known and the angle is required inverse trigonometric equations are required and supplied below.

$$\Theta = \cos^{-1}(X_{\text{Position}} / \ell) \textbf{ OR } \Theta = \sin^{-1}(Y_{\text{Position}} / \ell) \underline{\textbf{OR}} \ \Theta = \tan^{-1}(Y_{\text{Position}} / X_{\text{Position}})$$

2.1.2 Forward Kinematics

The goal of forward kinematics is to determine the locations of parts of the mechanism structure in space from the given joint angles in the kinematic chain:

$$\text{Given } L = (\ell_1, \ell_2, \ldots, \ell_N) \text{ and } J = (\Theta_1, \Theta_2, \ldots, \Theta_N)$$

$$\text{Find} \qquad P = (P_x, P_Y, P_Z)$$

Forward kinematics is derived from the trigonometric on the unit circle. It is formally defined as the mapping from joint space to cartesian space. The goal is to find the end destination when given joint angles and link length. A 2D method to imagine this is a mapping from unit circles multiplied by link length to cartesian space.

This process is repeated from the base frame defined in Cartesian space as the origin and is continued until the end of the "kinematic chain" to calculate the end of the chain.

$$\text{End destination location} = (P_x, P_Y, P_Z)$$

$$\text{Unit circle} = (\sin\Theta_1\cos\Theta_1, \sin\Theta_2\cos\Theta_2, \ldots, \sin\Theta_N\cos\Theta_N)$$

$$\text{Link length} = (\ell_1, \ell_2, \ldots, \ell_N)$$

2.1.3 Inverse Kinematics

Inverse kinematics addresses the reverse problem of determining joint angles that achieve a given location of a part of the robot mechanism:

$$\text{Given } P = (P_x, P_Y, P_Z) \text{ and } L = \ell_1, \ell_2, \ldots, \ell_N). \text{ Find } J = (\Theta_1, \Theta_2, \ldots, \Theta_N)$$

Inverse Kinematics aims at answering the question what my joint angles are given my desired end effector position and orientation. The trajectory of the end effector is directly influenced by the trajectory of the joint angles. Thus, inverse kinematics is defined as mapping cartesian space to joint space.

A solution to translate cartesian space to joint space is not arbitrary but for redundant kinematic mechanisms such as most humanoid skeletons there are an infinite number of joint configurations given an arbitrary trajectory. Thus, to make this problem tractable constraints must be imposed that differentiate solutions to make the problem analytically solvable. Typically, constraints are most energy efficient, shortest trajectory for all relevant motors, or similar optimization criteria.

2.1.4 Gram-Schmidt Process

The Gram-Schmidt process [15] is the process of orthonormalizing a set of vectors in inner product space. Orthonormalizing is a fusion of two words ortho "upright" and normalizing "to make normal". Thus, a set of orthonormal vectors consists of vectors that are perpendicular and have a length of 1. Each vector is upright relative to each other and have a normal length of 1.

Inner product space is a vector space that includes both complex and real vectors that combine two vectors into a single scalar value derived from the inner product. The purpose of inner product space is to provide a quick intuitive observation of two vectors with a single scalar value. For example, if the two vectors are perpendicular, their resulting scalar value will be zero. If the vectors are exact opposites they will result in a negative value, and the lengths would be multiplied. If they are pointing in the same direction their scalar value will be positive and lengths multiplied.

Vector $\vec{v}$ is "projected" orthogonally onto a line spanned by vector $\vec{u}$.

$$Proj_{\vec{u}}\vec{v} = \frac{\langle \vec{u}, \vec{v} \rangle}{\langle \vec{u}, \vec{u} \rangle}\vec{u}$$

The Gram-Schmidt process operator is as follows:

$$\mathbb{R}^1: u_1 = v_1$$

There are no other vectors in first dimension with a length or different direction, u_1 is the orthogonal vector.

$$\mathbb{R}^2: u_2 = v_2 - proj_{u1}(v_2)$$

Projection of $\vec{v}_2$ onto $\vec{u}_1$ is the rotation of v_2 on to u_1. The orthogonal vector is obtained by subtracting that projection from v_2.

$$\frac{\langle \vec{u}_1, \vec{v}_2 \rangle}{\langle \vec{u}_1, \vec{u}_1 \rangle}\vec{u}_1$$

Projection is the measure of how similar the two vectors are. Gram-Schmidt's method says it's the ratio of $\vec{v}_2$'s length and the rational similarity of $\vec{v}_2$and $\vec{u}_1$.

Similarity refers to the following:

If the dot product is within the domain of $||\vec{v}_2||*||\vec{u}_1|| > 0$, the angle between the two vectors is acute. Thus, categorized as "similar".

If the dot product equals zero, then categorized as not similar and perpendicular.

If the dot product is within the domain of $-(||\vec{v}_2||*||\vec{u}_1||)$, the angle between the two vectors is obtuse. Thus, categorized as "different".

The inner product maps length and angle into a scalar value. This is our first operation, the inner product between $\vec{v}_2$ and $\vec{u}_1$. A division by the length of $\vec{u}_1$ cancels out length from the scalar value derived from the inner product of $\vec{v}_2$ and $\vec{u}_1$. This gives the "ratio of $\vec{v}_2$'s length and the rational similarity of $\vec{v}_2$and $\vec{u}_1$."

Next, the scalar ratio is transferred to $\vec{u}_1$ effectively rotating it and scaling the length.

Finally, subtracted from $\vec{v}_2$ and scaled and rotated, $\vec{u}_1$ computes the orthogonal vector and normalizes it to give us the orthonormal vector.

This process can be repeated to $\mathbb{R}^n$ with the following equation:

$$u_m = v_m - \sum_{n=1}^{m-1} \left(\text{proj}_{u_n}(v_m)\right) \qquad e_m = \frac{u_m}{||u_m||}$$

2.2 Motion Filtering

Motion filtering is aimed mainly at eliminating noise and variation to obtain a better estimate of the true value in environments with observation noise, such as vision domains. The most used technique due to its formal foundation and relatively low computational complexity is the discrete Kalman filter [16].

2.2.1 Basic Discrete Kalman Filter

The Kalman filter's objective is to intuitively guess predictive values. It is an iterative mathematical process that uses a set of equations and consecutive data inputs to quickly estimate

the true value, position, velocity, etc. of the object being measured while the measured values contain unpredicted random error, uncertainty, or variation.

It is important to know and understand the inputs of the Kalman filter. In a spatial domain, data points are just points on a graph indicating a single exact location. The Kalman filter uses Gaussian probability distributions instead of points to be able to keep track of the possible true locations and not to have to commit to a specific estimate at an early point in time. The location relative to the center of this distribution reflects the likelihood of a point to be the true point with the maximum likelihood being at the center of the distribution. In this Gaussian, lines of equal probability form ellipsoids and areas within an ellipsoid correspond to a set of points that cover a certain likelihood that the true point is contained within that set. Thus, the distribution reflects the unpredicted random error, uncertainty, or variation. The Kalman filter deals with uncertainties and not absolutes and estimates the true distribution of the possible estimates for the true point. This is accomplished by having the states and inputs treated as Gaussian distributions. A mean and a corresponding covariance matrix can fully represent this distribution. Since these distributions are closed under the required fusion operations, this effectively models the estimate of data points as Gaussian distributions, i.e. a bell curve in which the highest probability for the correct data point is at the center of the curve. Ellipsoidal areas thus reflect probability regions, specially, with a 68% likelihood the correct prediction of the data point will be within 1 standard deviation and with 95% probability it would be within the domain of 2 standard deviations.

An intuitive perspective, rotating the curve to view it from a bird's eye view the probability regions of the bell curve in 2 dimensions look like circular/ellipse shapes with identical centers at the mean. Distributing the intuitive perspective upon this bird's eye view ellipse, the observer would see 3

pseudo ellipses, each one standard deviation larger than the previous and representing increasing probability regions as in the following figure:

Note: **Not to scale**

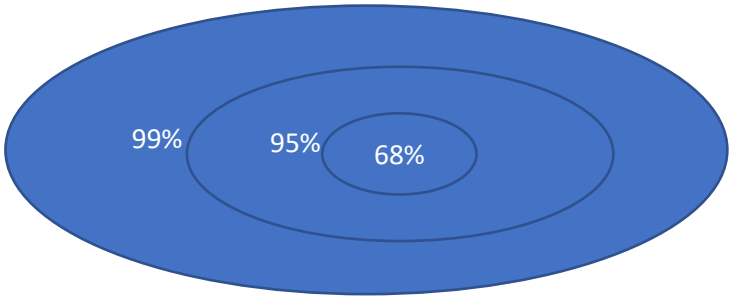


Figure 2: Kalman filter uses Gaussian probability distributions

Kalman Gain

The next important aspect of the Kalman filter is the Kalman gain and how it is derived. The Kalman gain represents the amount of focus to put on the predictive model and the sensor/input readings. It depends on the calculated error between the prediction calculations and the sensor error. Below is the example operation of the filter in a 1-dimensional case to simplify.

$$E_{estimate_t} = \frac{E_{measured} * E_{estimate_{t-1}}}{E_{measured} + E_{estimate_{t-1}}}, where\ E_{measured}\ is\ "tuned"$$

Kalman Gain is computed:

$$Kalman\ Gain = \frac{E_{estimate}}{E_{estimate} + E_{measured}}, where\ 0 \leq Kalman\ Gain\ \leq 1$$

Kalman Gain is applied:

$$Estimate_t = Estimate_{t_{-1}} + KG[Measured - Estimate_{t_{-1}}]$$

$E_{Measured}$ depends on the measurement noise where it varies from application to application. As for the Kalman gain, when it approaches 1 the prediction model is ignored and the system completely relies on the sensor input. On the other hand, if the Kalman gain approaches 0, the prediction is accurate and should be focused on over sensor data, thus the sensor data is ignored.

Intuitively, if the Kalman gain is approaching 1, a large amount of focus will be applied to the measured sensor data and the current estimate will be updated based on the sensor data minus the expected sensor data from the previous estimate to obtain a new estimate of the true value.

Below is the overall process of the Kalman filter.

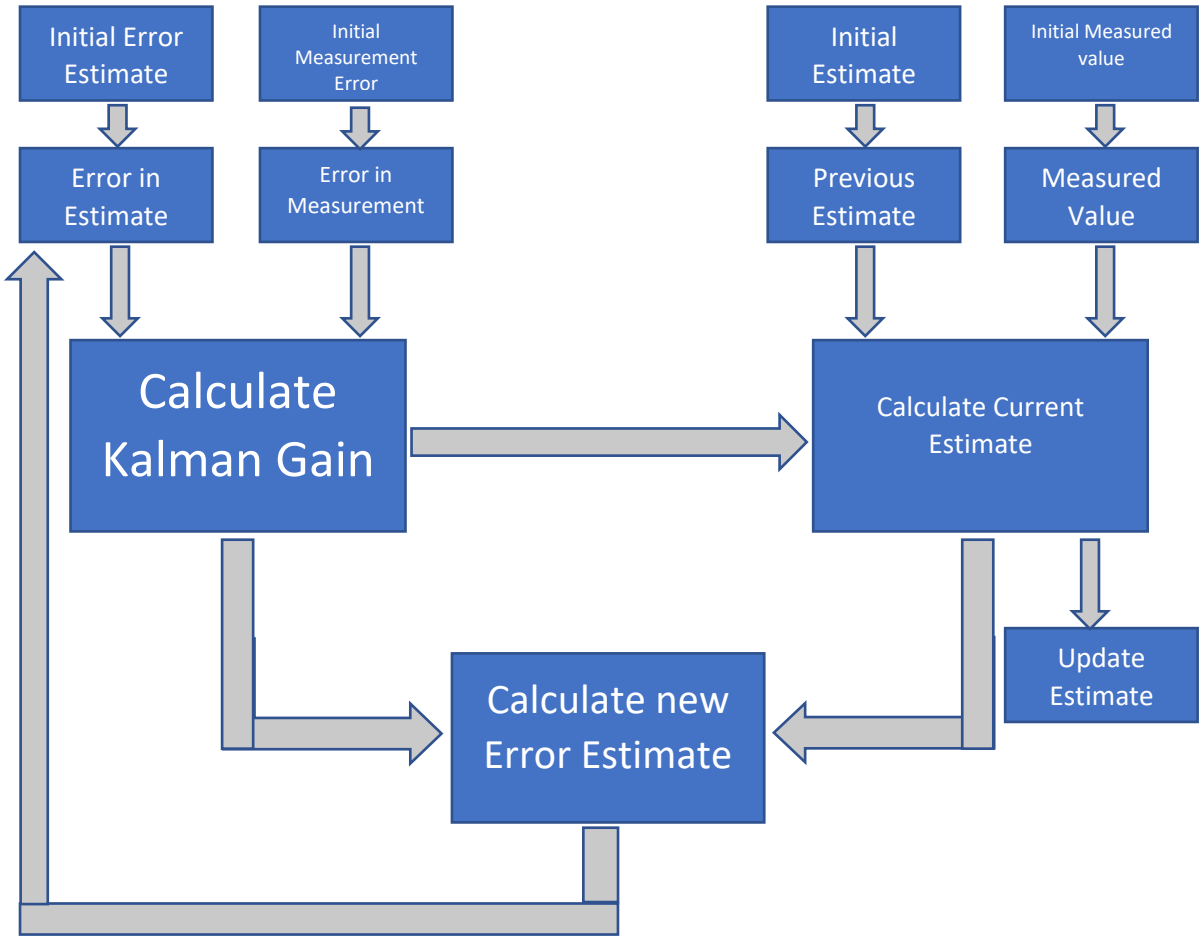


Figure 3: Kalman Flow Chart

2.3 Reward-Based Motion Optimization

To adapt movement to the dynamics of a different platform while optimizing second criterion this thesis builds on a previous framework, PILCO, that avoids requiring an explicit model while limiting the need for experimentally collected data. For this, it builds a model in the form of a Gaussian process model and then utilizes a reward-based optimization framework to derive model-based optimal control given the used cost criteria.

2.3.1 PILCO – Probabilistic Inference from Learning Control

Marc Peter Deisenroth published his dissertation November 20, 2010, and later revised it on August 4, 2019 [3], introducing a control optimization framework that reduced the amount of data required. His work is revolutionary in robots and efficient learning. Most traditional reinforcement learning methods often require a large amount of data and trials to learn how to complete a task. These tasks are broken down to simple segments learned separately then concatenated all together. Model-free reinforcement learning in this form is not practical for robotic applications because the amount of time to learn these tasks and in particular the number of actual operations before knowing the solution and thus at risk to produce system failures is often too large to be practical.

Marc Peter Deisenroth developed a method that increased data efficiency that does not require prior expert knowledge in terms of demonstrations or differential equations. Instead, a general policy-search framework for data-efficient learning from scratch was developed. He goes in detail how this is accomplished in his dissertation and publications.

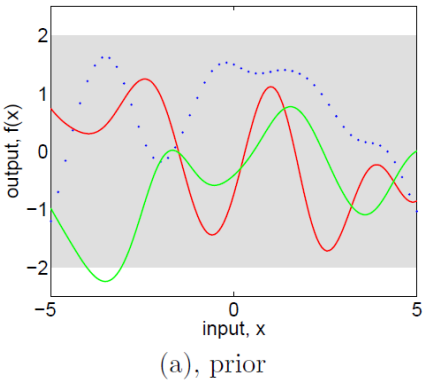
In the following, this thesis will introduce the three main points of focus, the Gaussian process (GP), hyperparameter optimization, and cost function optimization. The GP is the method that

forms the probabilistic model that is the foundation of the model-based optimization framework. Hyperparameters are used that precondition the data to make the GP effective. The cost functions represent the optimization criterion and thus are ultimately used to determine what actions and states are “good” and which are “bad”. The determination of these desired states allows “reinforcement learning” to occur.

2.3.2 PILCO - Gaussian Process

Like many other supervised machine learning methods, the goal of Gaussian process is it attempts to statistically predict the hidden function given a data set. Gaussian processes are a way to learn a distribution of functions from a dataset where the distribution is represented captured in terms of Gaussian distributions representing the function value range over different samples. Bayesian modeling of functions is how the Gaussian process attempts to predict the hidden function of a data set.

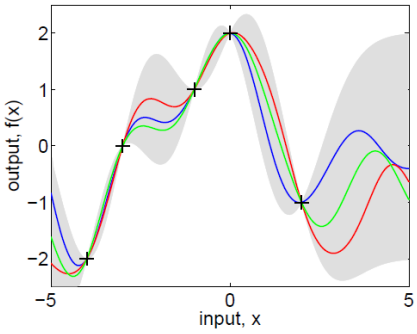
In a Gaussian process, the goal function is estimated from a set of observed functions. Starting from a prior function, observations of functions are integrated to estimate new means and covariances at different points in the input space.



(a), prior

Figure 4: Gaussian Prior (From http://www.gaussianprocess.org/gpml/chapters/RW.pdf)

Above is the prior which is in this case is a constant function with value 0 with a high covariance. This prior distribution of functions is represented as the grey region, representing the set of functions that stay within this region. The red and green lines are two synthetic functions generated and evaluated at many points. The dotted blue function is the hidden true function. Prior to applying the GP, the confidence area, displayed as the shaded grey area, shows the normalized standard deviation of 2σ of the Gaussians at each point. The two generated functions here have no correlation with hidden function. Starting from the prior, the GP observes multiple noisy versions of the underlying function and accordingly modifies the gaussian estimates to better capture the possible true functions considering the underlying Gaussian noise assumptions



(b), posterior

Figure 5: Gaussian Posterior (From http://www.gaussianprocess.org/gpml/chapters/RW.pdf)

The posterior example above displays the prior's conditioned outcome drawn from 5 observations, 3 of which are indicated by the colored lines. It can be seen that in the posterior the uncertainty region has moved and shrunk depending on the variation in the sample functions at that point. The synthetic functions allow the GP's estimate to coverage upon the observed data points and confidence areas are more certain where data is available.

A simplistic way of explaining the methodology of Gaussian process regression is that it generates multiple plausible functions given a data set and formulates a Gaussian distribution over those synthetically generated plausible functions. Because of how this is accomplished, the Gaussian process suffers from reduced model bias because the extra layer of mathematical abstraction. It is crucial to note that while outlier functions still have an impact upon the Gaussian process, it is significantly reduced. While impact is diminished it still influences estimating the underlying function. The following shows this interpretation by showing sets of possible functions.

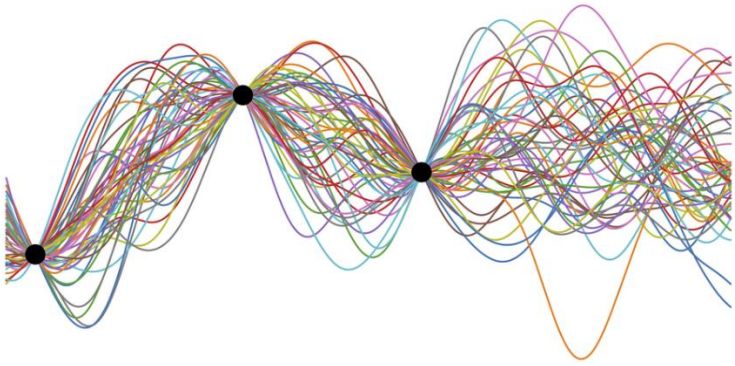


Figure 6: Synthetic Function Generation (From https://www.aidanscannell.com/post/gaussian-process-regression/)

The black dots are the data points that each of the synthetic functions converge on. The colored lines are the synthetically generated functions. For large data sets (n > 10,000), generation of each of these plausible functions and the required computation and storage of an order $O(N^3)$ is not

practical for modern computer systems (Gaussian Processes for Machine Learning) and thus representing the distribution as a Gaussian process is significantly more efficient.

Because of these synthetic function generations, the Gaussian process is ideal for scarce data sets, and converges upon a good estimate function quickly due to its data efficiency. Data efficiency is accomplished though extrapolating additional data or hypotheses from the data set that presents more data without the requirement of larger data sets.

2.3.3 PILCO - GP – Hyperparameters

There are situations where generated functions are not converging es expected due to sporadic nature or sparseness of the data set. Thus, there are 3 hyperparameters used as a mechanism to tune. These hyperparameters are integral to the Gaussian process' ability to predict the hidden function that is actively estimated. Hyperparameters are tuned on the training set.

The process of tuning inputs to generate meaningful functions is like a lens bending light to a focal point. Lengthscale and signal variance hyperparameters refer the order of magnitude relative to length and indirectly amplitude or size respectively. To keep the generating functions meaningful, these parameters require to be tuned precisely so the lens, i.e. the Gaussian process, reveal the focal point, i.e. the hidden function. While length scale and signal variance are the tuning parameters for the lens, noise variance is a parameter to describe the blemishes of the lens. Thus, giving a way to describe measurement noise into the Gaussian process.

PILCO - GP – Hyperparameters – Lengthscale

Lengthscale describes an order of magnitude in respect to length. The desire is to focus on one specific order of magnitude. Basically, they limit how fast a function can change over the input and thus represent a property akin to smoothness. Lengthscale is also very useful to avoid model overfitting. Short lengthscales promote overfitting while longer reduce it. While tuning this parameter it is recommended to select longer over shorter, especially if only small datasets are available.

Lenthscale depends on scaled closeness of the inputs where,

$$d = ||x - x'||$$

and the length scale,

$$d/\ell = ||x - x'||/\ell$$

The figure below shows examples with 3 different lengthscales from small (left) to large (right).

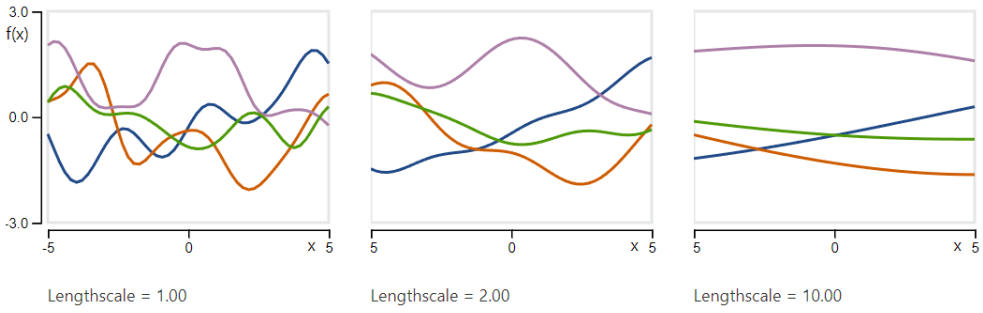


Figure 7: Lengthscales (From https://infallible-thompson-49de36.netlify.app/)

A risk with longer lengthscales is that they could distort the inputs enough to smooth out all the features of the generating functions and make it impossible to estimate the hidden function. Longer lengthscales map the input data to narrow ranges causing the data to be strongly correlated, while

shorter keep most of the features. A lengthscale of 1 keeps all the features and increasing the value causes smooth distortion throughout.

Data consistency is the key. If data has all differing order of magnitude lengthscales then increasing the lengthscale to map the data to narrow ranges is the objective. For example, mapping time vs meters and time vs millimeters increasing the lengthscale by a factor of two to three would be ideal.

PILCO - GP – Hyperparameters – Signal Variance

Signal variance describes fluctuation of the input data. Indirectly, signal variance influences amplitude.

$$Amptitude = \sqrt{\sigma^2}$$

The figure below shows examples with different signal variance from small (left) to large (right).

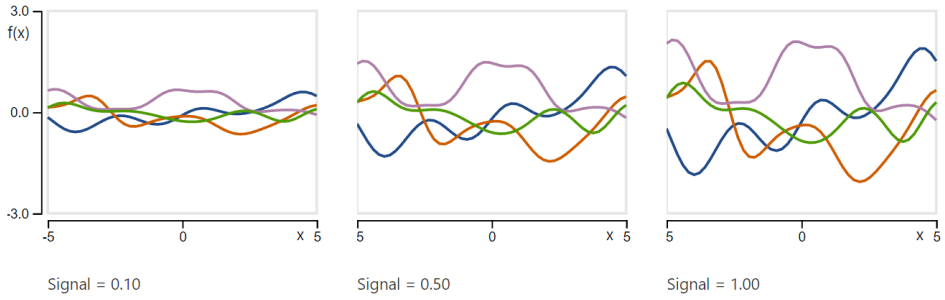


Figure 8: Signal Variance (From https://infallible-thompson-49de36.netlify.app/)

As with the lengthscales the variance must be consistent for the Gaussian process to accurately estimate the latent function. There are two key goals while tuning signal variance.

First, prior knowledge of the observed function values is extremely helpful and can assist in initializing the value correctly.

Second, knowledge of the signal to noise (S/N) ratio is useful. If the noise is higher than the signal, meaning S/N < 1, then there is no point applying the Gaussian process.

PILCO - GP – Hyperparameters – Noise Variance

Noise is a separate, typically unwanted signal that comes from the environment or even the system. It distorts the data's true positions. Having a good estimate of the noise through the noise variance parameter has two benefits for the Gaussian process.

First, if the hyperparameter closely approximates the true noise variance then it will be effectively accounted for and be at least in part "filtered" out.

Second, this hyperparameter indirectly simplifies the mean functions and indirectly assists with the lengthscale function by avoiding model overfit. This hyperparameter is useful in cases where there is very little noise but complex latent functions.

The following figure shows examples with 3 different noise variance parameters from small (left) to large (right).

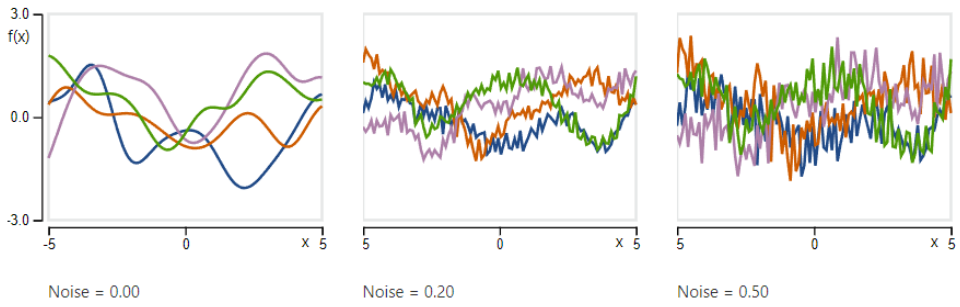


Figure 9: Noise (From https://infallible-thompson-49de36.netlify.app/)

The hyperparameters are parameters that set up the Gaussian process to be more effective by manipulating the data to be more consistent. While there are methods to optimize these parameters, it is beneficial to initialize them to approximately correct values with prior knowledge so they will avoid incorrect local minima as only limited amounts of data are available.

2.3.4 PILCO – Cost Function

For the learning to occur a cost function must be selected that is specific to the task at hand and encode the important aspects of the task. In this thesis, two loss functions have been selected, namely one utilizing saturated loss and one using a double hinge loss which are added together. The task is defined to maintain stability in an inverted pendulum with an arbitrary Zero Movement Point (ZMP) (encoded as a double hinge loss) while following a directed demonstration (encoded as a saturated loss). No knowledge of the demonstration to be followed nor the dynamics of the pendulum will be encoded beforehand but rather both will have to be inferred from observed trajectories and controls.

The saturated loss function is defined below,

$$c(x) = 1 - e^{\left(-\frac{1}{2a^2} d\left(x, x_{target}\right)^2\right)}$$

This saturated loss maps similarity to demonstration to cost as seen below.

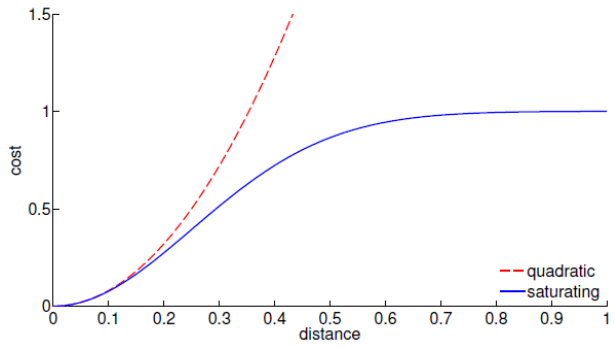


Figure 10: Saturation Loss Function (From https://deisenroth.cc/pdf/thesis.pdf)

This immediate cost function does not explicitly encode any knowledge of the task. The sole purpose is to penalize the geometric distance of the current state to the target state. $D(x,x_{target})$, which returns the geometric distance from current position to target state. The main benefit of saturated loss is it saturates at 1 and thus does not grow towards infinity for larger distances. It is perfect for small immediate adjustments in certain ranges.

The double hinge loss function models a preference region with increasing cost as the system leaves this region. In this thesis this will be used to provide stability-related costs where the preferred region is when the COM is within the support triangle for statically stable motion while cost increases as the system leans beyond this region. The following shows the base form of the double hinge loss function:

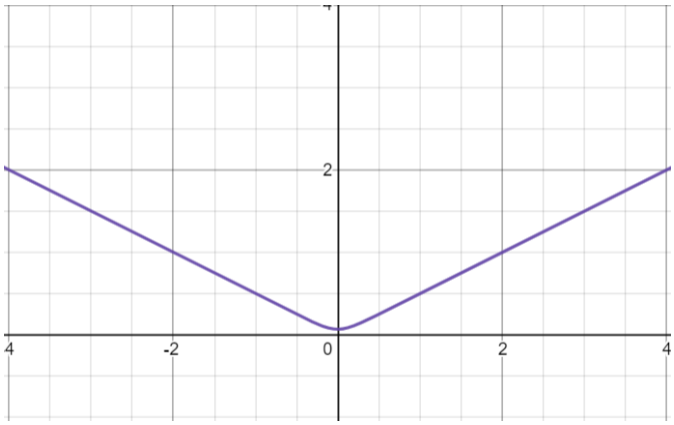


Figure 11: **Double Hinge Loss (From https://www.desmos.com/calculator)**

For a point from a Gaussian distribution will incur an expected cost under a double hinge loss function of

$$c(x) = a(\frac{2b}{2erfb + c + b})$$

Where the parameters a, b, and c as well as the underlying function *erf*b are:

Slope of double hinge:

$$a$$

Corner points:

$$b = [b_1\ b_2]$$

From the Gaussian process, Covariance matrix:

$$s$$

And the law of facility of errors:

$$c = (\sqrt{\frac{s}{\pi}})e^{-sb^2}$$

Finally the Gauss Error function:

$$\operatorname{erf} b = \frac{2}{\sqrt{\pi}} \int_0^{b/\sqrt{2s}} e^{-t^2} dt$$

Which is also shown in the following figure:

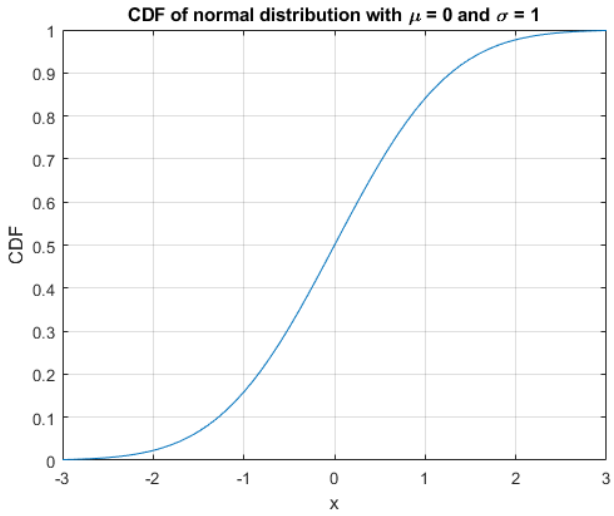


Figure 12**: Gaussian Error Function (From https://www.mathworks.com/help/matlab/ref/erf.html)**

The point of the double hinge function is to encode b with the ZMP calculated by the location of the feet thus the function will be dynamic based on the foot placement. This function does not saturate, allowing to impose priority of stability over imitation as represented by the saturation function.

CHAPTER 3 - APPROACH AND IMPLEMENTATION

This chapter will explain the process of the work completed in this thesis. Chapter 3.1 will be the Acquisition of the data and how that is accomplished. Due to limitations with the used Kinect V2 sensor, this will take place on a windows Computer. Chapter 3.2 will describe the preprocessing of the data received from the Kinect on the Linux computer. Chapter 3.3 will elucidate the Kalman filter method. The final part of the chapter in Section 3.4 will bring to light the learning process to adapt to the different dynamics by addressing the tradeoff between demonstration and stabilization.

3.1 – Acquisition of the Kinect Data

The first step of adaptive human-robot motion transfer for complete body imitation starts by extracting 3D coordinates from the Kinect V2. The device comes with a software development kit (SDK) [17]. This greatly assists in accessing the Kinect's sensor readings and Microsoft's proprietary software. The Kinect V2 can recognize human shapes with a certain degree of accuracy and is able to synthetically extract and draw a skeleton upon the human shape and display this via their Kinect V2 studio.

To be able to interface to the robot platform, Linux has a well-known platform called Robotic Operating System (ROS) [18]. It provides tools, libraries, and capabilities to develop robotic applications and interface to robot simulations and robot hardware. In this paradigm, ROS will be used as a messaging system where each process can know the location of each other and

communicate with each other. In addition, catkin will be used to adapt their use of macros to cross compile different packages whether in C, C++, Python, etc. using catkin make command.

Computer system setup required Ineracting Windows and Linux computers to utilize Kiect V2 and ROS

Integration between the two systems is required for this project. At the time of the development of this project ROS-Windows did not have a lot of support thus it was disregarded as an option. The initial framework for the integration of the Kinect V2 and ROS was derived from https://github.com/msr-peng/kinect_v2_skeleton_tracking. The open-source algorithm is a plugin that contains the Kinetic V2 libraries, ROS library, and other various files. The Windows plugin copies the position and orientation of each joint the kinetic is currently tracking and sends it via the ROS serial Windows extension to the wider ROS network with the ROS master on the Linux computer. The Kinect V2 orientation information was disregarded because it contained the orientation of the joint relative to the base frame, and this was not useful information for this project and will have to be computed relative to their appropriate respective joints.

The Windows plugin algorithm can track up to 6 different people and transfer their joint information. Possible but due to the limitations of the Kinect V2 infrared sensors it would be extremely difficult to produce accurate readings [19]. The algorithm first initializes the data, by

creating a pointer to the location of the Kinect data structure, pointer to current frame, and pointer to each body that is tracked.

After the pointers are assigned, it attempts to establish a connection with ROS on the Linux computer. The idea is to first establish a connection with the Kinect V2 device and then to establish a connection with the Linux computer. Once completed, ROS topics user_0 to user_5 will be broadcasted via ethernet connection between the two devices. These are the body structures that can be tracked.

The algorithm structures each possible person as an object “body0” to “body5”. If the body was “tracked” its Boolean value will be marked as true. Each body object housed an array of joints, and each joint again a Boolean tracked, where each joint tracked will be propagated with position and orientation. Each joint will be related to a name based on the index; for example, base spine “joint” ID is 0 and this system is applied to all the other joints. If the joint was tracked, then the position and orientation information would be propagated into this object structure. If the body or joint was not tracked, then that part of the structure would be omitted.

The table below displays the structure of the Body represented in the algorithm.

Figure 13: Body Data Structure

Body	Body [0]	…	Body [5]
Body Tracked	True or False	…	True or False
Joint	Joint [0] … Joint [19]	…	Joint [0] … Joint [19]
Joint Tracked	True or False	…	True or False
Joint Name	"Base Spine"- "Shoulder Spine"	…	"Base Spine"- "Shoulder Spine"
Position	Pos [0] Pos [1] Pos [2]	…	Pos [0] Pos [1] Pos [2]
Orientation	Or [0] Or [1] Or [2] Or [3]	…	Or [0] Or [1] Or [2] Or [3]

The information recorded is "published" via ROS protocol standards as a geometry_msgs/ TransformStamped Message. The following table contains the structure of the message.

Message Type	Message
geometry_msgs	TransformStamped Message
std_msgs	Header header
string	child_frame_id
geometry_msgs	Transform transform

Figure 14: TransformStamped Message

The following contains the header information of each message.

Message	
Header	header
uint32	seq
struct	time stamp
string	frame_id

Figure 15: Header of TransformStamped

In the table below is the structure for the position as a 3D vector (Vector3) and the orientation represented in Quaternions.

Message	
Transform	transform
geometry_msgs	Vector3
geometry_msgs	Quaternion

Figure 16: 3D Vector Data Structure

The following displays the Vector3 “3D-Vector”, and Orientation represented in Quaternions.

Message	
Vector3	
float64	x
float64	y
float64	x

Message	
Orientation	
float64	x
float64	y
float64	x
float64	w

Once all the data is published and sent over the ethernet connection there is a sleep for 50ms because of the framerate of the Kinect V2 plus the processing time required. Pointers are then reset, including the pointers of the body structure and Kinect V2 structures in case of interruption between frames.

3.2 – Preprocessing the Data

The Kinect V2 provides the crucial position data on which all the calculations will be based. But it does not provide the directions of the vectors that are required to translate these into joint angles in the NAO robot's kinematic structure. What is required is the direction from the elbow to the right hand and the direction from elbow to shoulder. This will provide the basis for the inverse kinematics calculations.

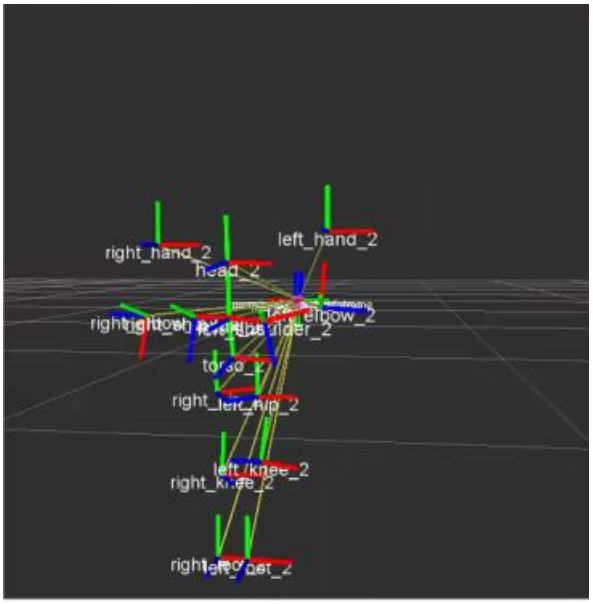


Figure 17: The orientation of each "joint" of the NAO robot's structure is displayed as its 3 basis vectors.

3.2.1 – Differences in Kinematics of NAO vs Human

The main difference between the human kinematics and the human are the arrangement of the NAO joints in the human ball joints (shoulder, wrist, and hip), as well as the more limited NAO hip and human hip Z rotation. Figure 6 displays the Z axis plane. Human hip rotation is approximately the Z plane rotation. However, the NAO hip the case is not. The rotation directly effects yaw and pitch. As shown in Figure 18 “NAO Joint scheme”, the NAO HIP indicated by the blue cylinder, is rotated 45 degrees from the x axis plane.

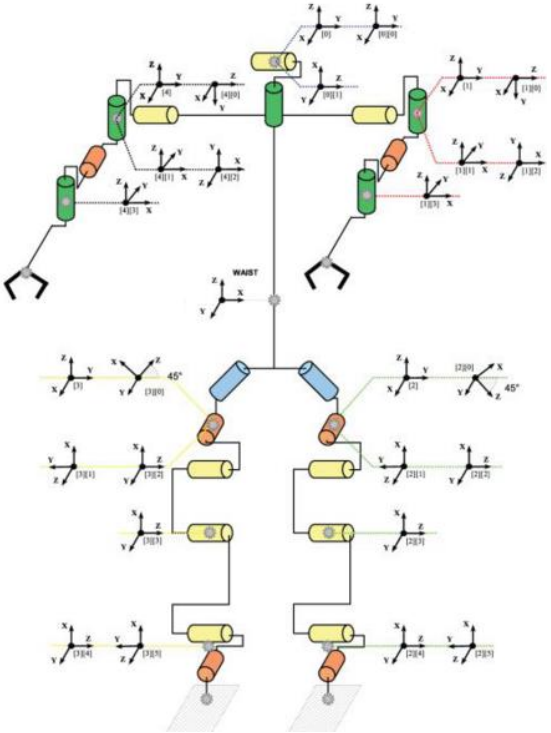


Figure 18: Human Joint Scheme

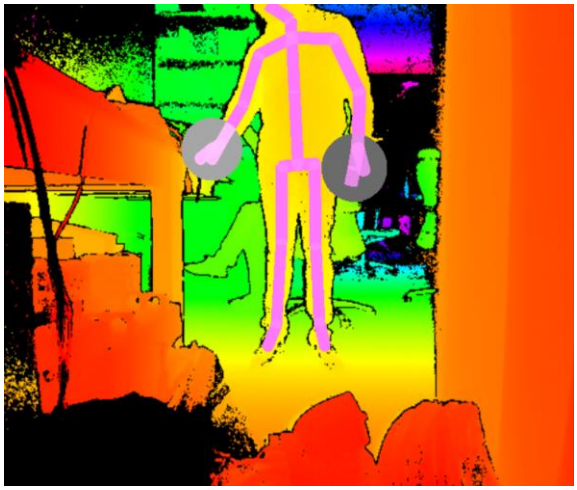


Figure 19: NAO Joint Scheme

The second step as mentioned will be the preprocessing through the inverse kinematics, transforming from human joint positions to NAO robot joint angles while preserving the appearance of the human demonstration. There are several methods for how this can be accomplished but the data from the Kinect V2 is structured as vectors, favoring an algebraic method. Assuming the length of the limbs will remain constant, the method of using vector math and Gram-Schmidt method will be used and explained.

3.2.2 – Compute the Vectors

Initially, the algorithm computes all the vectors with useful orientation information for the NAO joint rotations from the human skeleton observations. All vectors will be computed such that the head of the vector is the direction it is pointing towards, and the tail is the trailing end as indicated below.

All the vectors computed as such are thus:

$$V = head - tail$$

$$\vec{v}[0] = pos.x(head) - pos.x(tail)$$

$$\vec{v}[1] = pos.y(head) - pos.y(tail)$$

$$\vec{v}[2] = pos.z(head) - pos.z(tail)$$

The following table shows the important orientation-relevant vectors extracted:

	Vectors	
Left Hip and Right Hip	$\overrightarrow{LH} = LH - RH$	$\overrightarrow{RH} = RH - LH$
Left Upper Leg and Right Upper Leg	$\overrightarrow{LUL} = LK - LH$	$\overrightarrow{RUL} = RK - RH$
Left Lower Leg and Right Lower Leg	$\overrightarrow{LLL} = LA - LK$	$\overrightarrow{RLL} = RA - RK$
Left Upper Arm and Right Upper Arm	$\overrightarrow{LUA} = LE - LS$	$\overrightarrow{RUA} = RE - RS$
Left Lower Arm and Right Lower Arm	$\overrightarrow{LLA} = LHa - LE$	$\overrightarrow{RLA} = Rha - RE$
Up Back and Down Back	$\overrightarrow{UB} = SS - BL$	$\overrightarrow{DB} = BL - SS$
Left Shoulder and Right Shoulder	$\overrightarrow{LS} = LS - SS$	$\overrightarrow{RS} = RS - SS$
Left Foot and Right Foot	$\overrightarrow{LF} = LF - LA$	$\overrightarrow{RF} = RF - RA$

Figure 20: Orientation Table

Abbreviations

LH Left hip RH Right Hip

LK Left Knee RK Right Knee

LA Left Ankle RA Right Ankle

LE Left Elbow RE Right Elbow

LS Left Shoulder RS Right Shoulder

Lha Left Hand Rha Right Hand

LF	Left Foot	RF	Right Foot
BL	Base Link	SS	Shoulder Spine
UB	Up Back	DB	Down Back

3.2.3 – Compute the Synthetic Vectors

Calculation of the NAO hip will be more involved. The objective is to create a synthetic NAO hip vector. The purpose is to calculate the rotation effect this joint will incur on the rest of the leg. Rotational effects will incur error in all rotational planes if not calculated correctly. Thus, the Gram-Schmidt method is used to calculate the orthogonal basis of the two vectors, then normalization is applied. The desire is to have two vectors, namely the hip vector and up back vector which points straight up towards the head, to be perpendicular. When those two vectors are subtracted it will give the 45-degree NAO hip vector:

$$\overrightarrow{RNAO}_{hip} = \frac{GS\left(\overrightarrow{RH}, \overrightarrow{UB}\right)}{|GS\left(\overrightarrow{RH}, \overrightarrow{UB}\right)|} - \frac{\overrightarrow{RH}}{|\overrightarrow{RH}|}$$

$$\overrightarrow{LNAO}_{hip} = \frac{GS\left(\overrightarrow{LH}, \overrightarrow{UB}\right)}{|GS\left(\overrightarrow{LH}, \overrightarrow{UB}\right)|} - \frac{\overrightarrow{LH}}{|\overrightarrow{LH}|}$$

Where Gram-Schmidt Method is,

$$GS(\vec{v}_1, \vec{v}_2) = \vec{v}_2 - ((\frac{\vec{v}_1}{|\vec{v}_1|} \cdot \vec{v}_2) \cdot \frac{\vec{v}_1}{|\vec{v}_1|})$$

Where the length of |v| is defined,

$$|v| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

Next the computation of other synthetic vectors is required to compute additional angles needed in the next section.

Resulting Vector	Calculation
$\overrightarrow{LS} \perp \overrightarrow{LUA}$	$GS(LS, LUA)$
$\overrightarrow{RS} \perp \overrightarrow{RUA}$	$GS(RS, RUA)$
$\overrightarrow{LH} \perp \overrightarrow{LNAO}_{hip}$	$\overrightarrow{LH} \otimes \overrightarrow{LNAO}_{hip}$
$\overrightarrow{RH} \perp \overrightarrow{RNAO}_{hip}$	$\overrightarrow{RH} \otimes \overrightarrow{RNAO}_{hip}$
$\overrightarrow{LLeg}_{norm}$	See Below
$\overrightarrow{RLeg}_{norm}$	See Below
$\overrightarrow{LLeg}_{norm} \perp \overrightarrow{LNAO}_{hip}$	$\overrightarrow{LLeg}_{norm} \otimes \overrightarrow{LNAO}_{hip}$
$\overrightarrow{RLeg}_{norm} \perp \overrightarrow{RNAO}_{hip}$	$\overrightarrow{RLeg}_{norm} \otimes \overrightarrow{RNAO}_{hip}$
$\overrightarrow{LNAO}_{hip} \perp \overrightarrow{\overrightarrow{LLeg}_{norm} \perp \overrightarrow{LNAO}_{hip}}$	$\overrightarrow{LNAO}_{hip} \otimes \overrightarrow{\overrightarrow{LLeg}_{norm} \otimes \overrightarrow{LNAO}_{hip}}$
$\overrightarrow{RNAO}_{hip} \perp \overrightarrow{(\overrightarrow{RLeg}_{norm} \perp \overrightarrow{RNAO}_{hip})}$	$\overrightarrow{RNAO}_{hip} \otimes \overrightarrow{(\overrightarrow{RLeg}_{norm} \otimes \overrightarrow{RNAO}_{hip})}$

Figure 21: Synthetic Vector Table

While most of the vector calculations were reliable, the leg normal was not stable because any of the leg normal vectors taking the cross product of the upper leg with the lower leg or foot with the lower leg sometimes would be exactly 180 degrees, thus not allowing to extract a unique vector orthogonal to both and creating not a number (NaN) because of the calculation. Thus, an algorithm had to be developed to test the stability of the Leg normal vector. The algorithm simply would take the length of all the normal vectors which are the following:

$Foot \perp Lower\ Leg$
$Upper\ Leg \perp Lower\ Leg$
$Upper\ Leg + \ Lower\ Leg \perp Foot$
$Foot + \ Lower\ Leg \perp Foot$

Any length less than .0001 was discarded

1. Compute the Θ between normal vector and corresponding hip vector. If Θ was greater than 90-degrees, it was discarded.
    a. If left leg then left hip vector was used and the same applied for the right leg.
2. Longest length vector would be chosen to be the "best normal vector"

The longest length vector was chosen because it would be the "most stable" and the logic was which normal vector to choose if all of them fit within the constraints implemented. Upon reviewing the vector lengths there were typically some of a very short length of less than .001. Prior to the implementation of this algorithm the leg of the NAO virtual model would behave extremely strange. After this algorithm was implemented, it was significant more stable.

The selection of additional synthetic vectors for this was required for the situations where the leg was complete straight (Θ = 180), and the foot would appear to be straight with the leg. These were measured upon first developing the algorithm. The general idea is to generate a vector that pointed in the correct y-axis plane for the hip joint that was also meaningful.

3.2.4 – Compute NAO Leg Vector

To refresh from earlier, the NAO hip is the largest kinematic difference between human and NAO. Because of this there is a requirement to account of this difference. Under the simulation, the leg vectors are independent from one another; however, the real NAO platform's hip yaw has only one motor. In that case, implementation of only one side would be necessary.

Using the calculations above, the NAO basis, i.e. the 3 basis vectors of the 45-degree vector of the hip yaw joint can be represented as:

$$NAO_{basis} = \begin{bmatrix} \overrightarrow{NAO}_{hip_x} & \overrightarrow{\overrightarrow{NAO}_{hip} \perp (\overrightarrow{Leg}_{norm} \perp \overrightarrow{NAO}_{hip})}\_x & \overrightarrow{\overrightarrow{Leg}_{norm} \perp \overrightarrow{NAO}_{hip}}\_x \\ \overrightarrow{NAO}_{hip_y} & \overrightarrow{\overrightarrow{NAO}_{hip} \perp (\overrightarrow{Leg}_{norm} \perp \overrightarrow{NAO}_{hip})}\_y & \overrightarrow{\overrightarrow{Leg}_{norm} \perp \overrightarrow{NAO}_{hip}}\_y \\ \overrightarrow{NAO}_{hip_z} & \overrightarrow{\overrightarrow{NAO}_{hip} \perp (\overrightarrow{Leg}_{norm} \perp \overrightarrow{NAO}_{hip})}\_z & \overrightarrow{\overrightarrow{Leg}_{norm} \perp \overrightarrow{NAO}_{hip}}\_z \end{bmatrix}$$

Then the NAO x-axis rotation matrix is used rotate the NAO leg vector.

$$NAO_{rot} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(yaw) & -\sin(yaw) \\ 0 & \sin(yaw) & \cos(yaw) \end{bmatrix}$$

Finally, the setup of all that is required to calculate the NAO leg has been completed:

$$NAO_{leg} = NAO_{basis} * NAO_{rot} * NAO_{basis}^{T} * UpperLeg$$

This matrix multiplication is typically used to transform one coordinate domain to another, and this situation calls for it. The change of domain from the Kinect V2 input to the NAO modified leg is then performed and the yaw rotation is applied. Following this the domain is reverted and finally the input leg is applied to it and the NAO leg is computed.

What this calculation exactly does in this case is to negate the offset the NAO hip rotation would apply. A very simple example, if the demonstrator would rotate his leg in place, without rotation

of the pitch or roll of the hip, the NAO would appear to kick as the offsets were so vast. Direct z-axis or yaw rotation of the leg would infer a x-axis or pitch and y-axis roll offset. While just a pitch hip rotation would not incur any noticeable offset. But a roll rotation would incur an offset. Thus the correction is necessary to achieve correct inference of the joint angles.

Calculation of the NAO leg vector was one of the last challenging parts of the mimicry. There were some noticeable differences that can be attributed to the sensor error because of small difference between the foot and the ground. If the feet where not in the correct positions for the infrared sensors to accurately determine their coordinates, the foot vector would noticeably twitch. To remedy this, a filter would have to be integrated for this application which will be described in a later section.

3.2.5 – Compute the Joint Angles

As we now have the coordinate axes of each of the joints, the dot product is what is required to compute the angles between each vector thus giving us the joint angle we require. This is defined as displayed below.

$$\theta = \cos^{-1}\left(\frac{\vec{v}_1 \cdot \vec{v}_2}{|v_1| * |v_2|}\right)$$

Before the calculation of the joint angles, it is important to note the standard starting joint angles for the NAO robot, i.e. the angels considered to be 0. Figure 8 displays such a configuration for all but the elbow angles.

The following will explicitly define the calculations of the joint angles:

Θ	Left	Right
Elbow	$\cos^{-1}\left(\frac{\overrightarrow{LUA}\cdot\overrightarrow{LLA}}{\lvert\overrightarrow{LUA}\rvert * \lvert\overrightarrow{LLA}\rvert}\right)$	$\cos^{-1}\left(\frac{\overrightarrow{RUA}\cdot\overrightarrow{RLA}}{\lvert\overrightarrow{RUA}\rvert * \lvert\overrightarrow{RLA}\rvert}\right)$
Shoulder Roll	$\cos^{-1}\left(\frac{\overrightarrow{LUA}\cdot\overrightarrow{LS}}{\lvert\overrightarrow{LUA}\rvert * \lvert\overrightarrow{LS}\rvert}\right)-\frac{\pi}{2}$	$\cos^{-1}\left(\frac{\overrightarrow{RUA}\cdot\overrightarrow{RS}}{\lvert\overrightarrow{RUA}\rvert * \lvert\overrightarrow{RS}\rvert}\right)-\frac{\pi}{2}$
Shoulder Pitch	$\cos^{-1}\left(\frac{\overrightarrow{DB}\cdot GS(\overrightarrow{LS},\overrightarrow{LUA})}{\lvert\overrightarrow{DB}\rvert * \lvert GS(\overrightarrow{LS},\overrightarrow{LUA})\rvert}\right)-\frac{\pi}{2}$	$\cos^{-1}\left(\frac{\overrightarrow{DB}\cdot GS(\overrightarrow{RS},\overrightarrow{RUA})}{\lvert\overrightarrow{DB}\rvert * GS(\overrightarrow{RS},\overrightarrow{RUA})}\right)-\frac{\pi}{2}$
Knee	$\cos^{-1}\left(\frac{\overrightarrow{LUL}\cdot\overrightarrow{LLL}}{\lvert\overrightarrow{LUL}\rvert * \lvert\overrightarrow{LLL}\rvert}\right)$	$\cos^{-1}\left(\frac{\overrightarrow{RUL}\cdot\overrightarrow{RLL}}{\lvert\overrightarrow{RUL}\rvert * \lvert\overrightarrow{RLL}\rvert}\right)$
Hip Pitch	$\cos^{-1}\left(\frac{\overrightarrow{LNAO}_{leg}\cdot(\overrightarrow{LLeg}_{norm}\otimes\overrightarrow{LNAO}_{hip})}{\lvert\overrightarrow{LNAO}_{leg}\rvert * \lvert(LLeg_{norm}\otimes\overrightarrow{LNAO}_{hip})\rvert}\right)+\frac{\pi}{2}$	$\cos^{-1}\left(\frac{\overrightarrow{RNAO}_{leg}\cdot(\overrightarrow{RLeg}_{norm}\otimes\overrightarrow{RNAO}_{hip})}{\lvert\overrightarrow{RNAO}_{leg}\rvert * \lvert(\overrightarrow{RLeg}_{norm}\otimes\overrightarrow{RNAO}_{hip})\rvert}\right)-\frac{\pi}{2}$
Hip Roll	$\cos^{-1}\left(\frac{\overrightarrow{LNAO}_{leg}\cdot\overrightarrow{LH}}{\lvert\overrightarrow{LNAO}_{leg}\rvert * \lvert\overrightarrow{LH}\rvert}\right)+\frac{\pi}{2}$	$\cos^{-1}\left(\frac{\overrightarrow{RNAO}_{leg}\cdot\overrightarrow{RH}}{\lvert\overrightarrow{RNAO}_{leg}\rvert * \lvert\overrightarrow{RH}\rvert}\right)-\frac{\pi}{2}$
Hip Yaw	$-\cos^{-1}\left(\frac{GS(\overrightarrow{LNAO}_{hip},\overrightarrow{LH})\cdot GS(\overrightarrow{LNAO}_{hip},LLeg_{norm})}{\lvert GS(\overrightarrow{LNAO}_{hip},\overrightarrow{LH})\rvert * \lvert GS(\overrightarrow{LNAO}_{hip},LLeg_{norm})\rvert}\right)$	$-\cos^{-1}\left(\frac{GS(\overrightarrow{RNAO}_{hip},\overrightarrow{RH})\cdot GS(\overrightarrow{RNAO}_{hip},RLeg_{norm})}{\lvert GS(\overrightarrow{RNAO}_{hip},\overrightarrow{RH})\rvert * \lvert GS(\overrightarrow{RNAO}_{hip},RLeg_{norm})\rvert}\right)$

Figure 22: Joint Angle Calculation Table

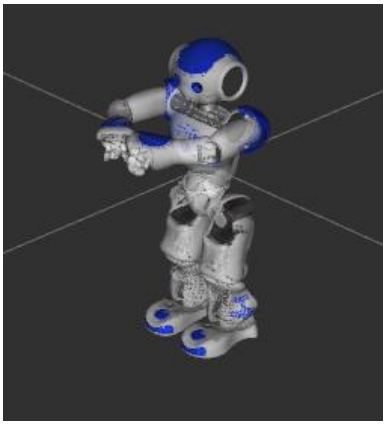


Figure 23: Rviz representation of the NAO

The elbow and knee angles are straightforward calculations by taking the dot product and applying the inverse cosine to calculate the joint angle. However, the shoulder pitch and roll require an offset of 90-degrees because of the "reference" vector used to measure the correct axis relative to the NAO. The reference vector for the shoulder pitch was the down back and a vector that would be measured in the x-axis rotation plane. For the hip pitch and roll the reference vector depending on left or right side would point in front of or behind the NAO robot and thus the offsets would require to be added or subtracted. The NAO leg vector would successfully negate the influence that the NAO hip would have imposed and is vital for the calculations. As for the hip yaw calculation, it required measuring the difference between a vector that was not a part of the leg and in this case was the hip vector and measuring rotation angle difference.

3.2.6 – Discrete Simple Kalman Filter

As indicated previously, sensor noise in the detection of the foot location sometimes causes erratic behavior. To address this a motion filter is applied to the corresponding joints. The Kalman filter was selected here for its low computation cost and ease to implement. In Chapter 2, details of the Kalman filter were introduced. Within this chapter practical application will be explicit described.

The system and sensor models of the Kalman filter represents the way that the system evolves over time and how the sensor readings relate to the system state.

The A-Matrix is referred to as the "System Dynamics Matrix". What this implies is that this matrix indicates the change in the system state over a time step dt. dt could be setup dynamically to account for lag or set as a constant value based on what the refresh rate of the sensor in question is. In this application, it was decided to set it to current time and each time current time was updated

the difference would be calculated from the previous time. Thus, the dynamic option was chosen to represent constant velocity motion for the joint and thus set to:

$$A = \begin{bmatrix} 1 & dt & 0 \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix}$$

The C-Matrix alludes to as the "Output Matrix". Is represents the relation between the actual sensor value and the system state. In this case, the observation is the joint angle derived from the Kinect sensor readings, resulting in the matrix below. The Q and R values which represent the associated uncertainties will be described later.

$$C = [1 \quad 0 \quad 0]$$

The Q-Matrix is describes as the "Process Noise Covariance Matrix". This matrix represents the amount of "noise" the user believes the model has and thus characterizes the amount the system deviates from the behavior indicated by the A matrix. This matrix will generally be a constant and not be set dynamically nor would any calculation be applied to find an optimal Q value. A standard initial recommend value would be used of .05 in this case.

$$Q = \begin{bmatrix} .05 & .05 & 0 \\ .05 & .05 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The R matrix is cited as "Measurement Noise Covariance Matrix". It characterizes the noise in the sensor reading and since the senor reading in this case is a scalar, the R matrix here is a scalar value. This value introduces the amount of error the user believes the sensor has. This value will be constant and not be set to dynamically change at any time. Nor were there any calculations to find an optimal R value. A standard initial recommendation value of 5 will be used here.

$$R = 5$$

The P matrix is the “Estimate Error Covariance”. This matrix represents the resulting uncertainty in the filter estimate resulting from the integration of the sensor readings. Typically, the estimate error covariance matrix would be a lot lower than the noise of the sensor. In this case, it would also be expected that the system error covariance is lower than the sensor variance as indicated by the chosen values for Q and R. The P matrix has to be initialized with constant values representing the uncertainty in the initial guess and are then updated with the calculations as a part of the Kalman process. The initial values used here are:

$$P = \begin{bmatrix} .1 & .1 & .1 \\ .1 & 10000 & 10 \\ .1 & 10 & 100 \end{bmatrix}$$

The Kalman filter is initialized with the initial values using the matrices describe. Time is initialized to 0 and our initial state is a 3-vector filled with ones. Then a Boolean check variable would be set to true to imply that the initial states are set. A NaN check was implemented to not break the matrix multiplication. The Kinect V2 will send NaN if a position was not tracked, and this would occur frequently. If this case would happen it was instructed to use the previous state value as an observation and not update the filter until a real number would be sent in.

Next the Kalman calculations would be implemented. Predicted state estimate would be applied. This would give the predicted change in state:

$$x' = A * x$$

Predicted Error Covariance is updated:

$$P = A * P * A^T + Q$$

Kalman Gain computed:

$$K = P * \frac{C^T}{C * P * C^T + R}$$

The final prediction is computed by integrating the Kinect-based observation:

$$x' = x' + K(x - C * x')$$

And the corresponding Process Error Estimate is calculated:

$$P = (I - K * C) * P$$

The Kalman filter would be applied to each of the joints. As mentioned in the previous section, this implementation of the Kalman filter noticeably cleared the foot twitch and other joint twitches within the NAO model. However, it also reduces the ability of the imitation to track very fast changes in movements caused by strong acceleration events due to the constant velocity system model. In the future we will implement proper calculations of the Q and R values but for this application the recommended values were satisfactory.

3.3 – Modeling Human Demonstration and Stability Using an Inverted Pendulum

A background was provided on the learning method selected. An optimized Gaussian Process method was selected for many positive attributes. The method is used in sparse data environments, leads to reduced model bias, incorporates a reinforcement learning facet to minimize the need for system knowledge, and can estimate the hidden function at a rapid rate with relatively accurate results. The open-source software provided by the creator Marc Peter Deisenroth made it a perfect application to select as it is easy to adapt to the problem at hand.

The reason why the Gaussian Process thrives in sparse data environments is because of the nature of the algorithm. It generates user defined synthetic functions to guess effectively mathematically what the hidden function is. Learning is flexible and can change over time. These changes could be the environment, friction from the motors, or any other unknown variable. It links theory to practical applications.

In this section, the cost functions and the pseudo dynamic environment function insights will be explained. Why certain values were chosen and why each cost function was chosen is also explained. The main goal of this application is to learn to adapt to the dynamic constraints of the robot platform and to closely imitate the demonstration while maintain a stable posture.

3.3.1 – Setup

As the simplified stability model models the mechanism as an inverted pendulum with the COM at the tip and an angular stability region which represents the conditions under which the COM is within the support polygon, two inverted pendulums would be set up, one representing the demonstration and one the imitating agent. As we are here interested in control commands for higher level position control, simplified equations of motion are applied to these pendulums where the demonstrator pendulum is driven by a pre-coded sequence corresponding to a recorded demonstration and the imitator pendulum follows a position controller, thus allowing direct velocity control of the joint. The goal is to simulate the demonstration (blue) which will be the human viewed by the Kinect V2 and the actual platform NAO robot (red). For the initial testing stages this system was not directly integrated with the joint angle system and real demonstrations. Rather, an oscillation function will be used to simulate a demonstration. The state of the Gaussian process thus includes both the angles and angular velocities of the demonstration and of the

imitation with the objective of determining controls for the imitation pendulum to maintain stability while following the demonstration.

Initial angle states are set as zero. Which is the straight up and measured in radians. And cost functions inputs will be in radians. Figure 24 shows an example of the graphical representation of the simulation of the demonstration and imitation pendulums.

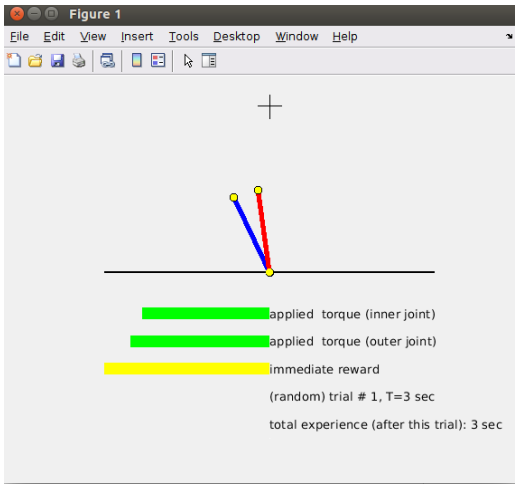


Figure 24: Matlab representation of the two pendulums

3.3.2 – Loss Function

To learn the imitation strategy, a cost function has to be provided that represents both the closeness between the demonstrator and the imitator state and the degree of risk in terms of stability to the imitator. Thus, a requirement to make the simulation possible is a loss function to penalize not following the displayed demonstration. Saturated loss was selected to penalized up to a maximum of 1 loss based on the difference between the two angles.

$$c(x) = 1 - e^{\left(-\frac{1}{2a^2} d\left(x, x_{target}\right)^2\right)}$$

The target was set to zero meaning that the pendulum angle difference between the demonstration and the current state should be zero if not a penalty will incur. The maximum loss of 1 is on purpose because of the use of a second cost function representing stability violations and in particular to ensure that this second function will dominate the first whenever stability would be critically violated even if maintaining stability would require dramatic deviations in angles from the demonstration. The whole reason to use two cost functions is to easily fine tune and understand what is being adjusted.

The "a" variable is used to adjust how fast the exponential function initially increases. Recommended value is the distance between current state and target state divided by 10.

$$a = \frac{d(x, x_{target})}{10}$$

The second cost function is a double hinge that provides 0 error if the system is still far from the edges of the stability area and then increases towards infinity as it moves towards the critical boundary where the system would fall.

$$c(x) = a(\frac{2b}{2erfb + c + b})$$

The variables of importance are the "a" and "b". The "b" would be where the slope of would initially start at, thus representing the interval in which the system is securely stable. The "a" will be the steepness of the slope, indicating how fast the risk of falling increases. This gives the ability to encode our ZMP angles into our cost function. The slope is a linear increase of the cost and was set to 3 to penalize quickly if the bounds were breached. The "b" value was set $\pi/8$ and $-\pi/8$.

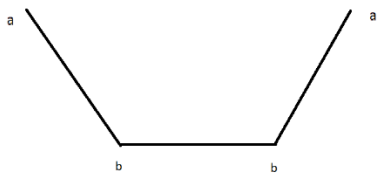


Figure 25**: Double hinge loss**

3.3.3 – Pseudo Dynamics – Demonstration to be learned

The demonstration that was designed is an oscillating sine wave function that will occur penalties if followed perfectly as its amplitude exceeds the limits of the stable region. This was by design to learn an oscillation with a truncated amplitude that incurs fewer penalties by preserving stability throughout.

The following displays the positions the demonstration pendulum will be oscillating at.

$$\theta_{Demo} = .5 * \sin(t)$$

The velocity of the demonstration pendulum is thus,

$$\theta_{Demo} = 2.5\dot{} * \cos(t * 5)$$

The difference in the position of the learning pendulum and the demonstration is,

$$\theta_{learn} = f(t) - \theta_{Demo}(t-1)$$

And the velocity difference of the demonstration and the learning pendulum is,

$$\dot{\theta}_{Learn} = \frac{1}{dt}(f(t) - \dot{\theta}_{Demo}(t-1)$$

The f(t) is the policy applied which is effectively a position command. The Gaussian process will be derived from initially from a set of random control trajectories and applied to learn a policy that

attempts to minimize the difference to the demonstration while staying within the stability region. The loss hinge loss function will incur a penalty to follow this demonstration because the oscillation will exceed the ZMP bounds that was setup. Thus, it is expected to see the pendulum learn to oscillate with the approximate function that will effectively cut the tops of the sine wave to avoid instability in the extremes but otherwise closely follow the sine wave.

The following shows the learning progress in terms of the predicted cost by the GP and the actual cost during learning. The system starts initially with a random policy, generating a set of 10 largely random executions to form an initial Gaussian process model. After this initial phase, in each subsequent trial the GP is used to update the policy before a set of 10 additional trajectories are generated with the resulting policy, and the GP is updated accordingly. Figure 26 shows the performance after Trial #2. Trial #1 is disregarded because the initial policy implementation was random. This figure shows the cost and cost predictions over time with the blue curve representing the mean of the GP prediction, the blue intervals indicating the corresponding confidence intervals, and the red curve showing the actual cost incurred when executing the policy learned in this trial.

This figure shows that initially, based on largely random trials, the GP predicts a relatively high cost throughout the execution with very high confidence intervals. Also, it shows that this corresponds to the initial policy which also produces large costs due to the fact that it has not yet learned that following the demonstration will reduce cost. The initial drop, however, shows that some learning has taken place where the system avoids staying at the expected cost of a pure random policy.

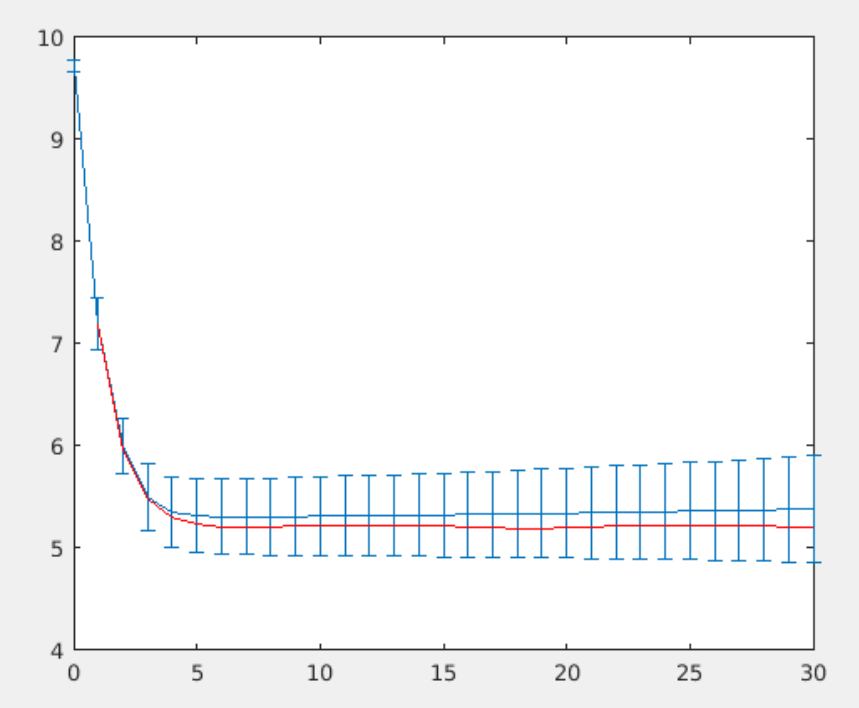


Figure 26: PILCO Trial #2

After 33 to 35 additional trials, which corresponds to a total of about one minute of actual policy execution time, the results in Figure 27, Figure 28 and Figure 29 show that the system over time was able to successfully learn to reduce the cost, resulting in the anticipated behavior.

These figures show that the overall cost incurred as well as the cost predicted have dropped by more than 2 orders of magnitude, indicating that the system has learned to follow the demonstration and to avoid instability. Looking at the actual cost (red line), it can be seen that it initially spikes before reducing close to 0 with largely periodic small spikes of around 0.04 in Trials #33 and #34 and even smaller very periodic spikes of 0.02 after Trial #35. While in Trials #33 and #34 these later spikes are still somewhat asymmetric and of different magnitude, after trial #35 their periodic character can be clearly seen.

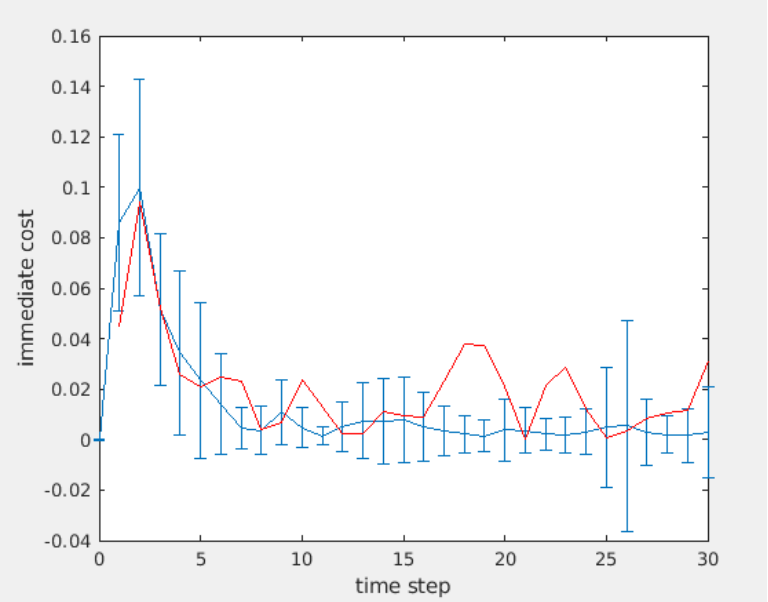


Figure 27: PILCO Trial #33

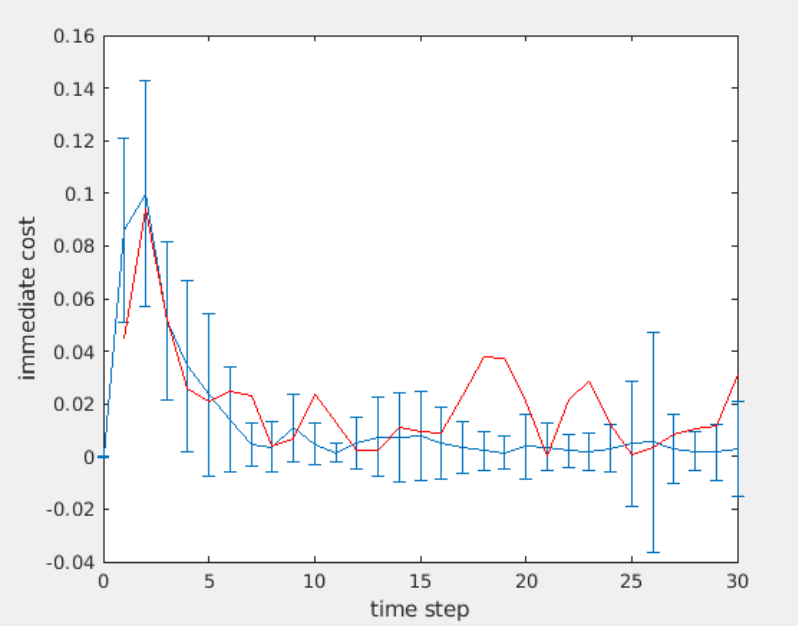


Figure 28: PILCO Trial # 34

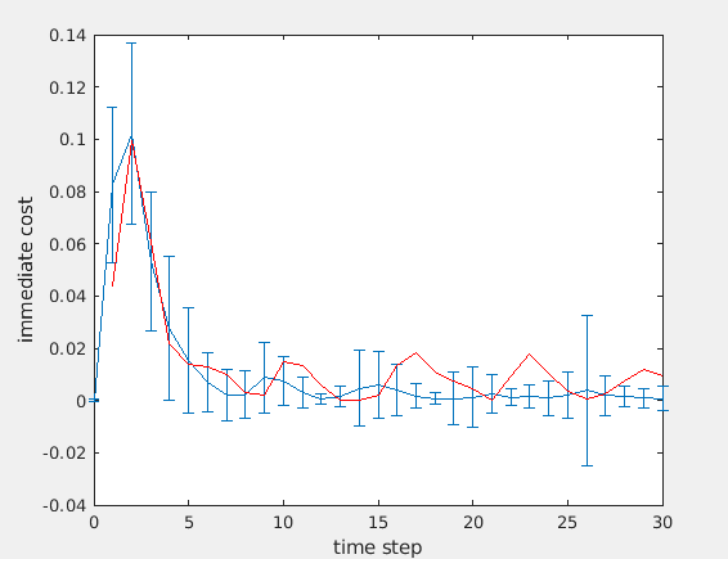


Figure 29: PILCO Trial # 35

The reason for this behavior is that as the demonstration starts with a speed of 2.5 while the imitator starts at a speed of 0, the two pendulums have to initially diverge before the imitator can catch up with the demonstration. Once that happens, the imitator successfully tracks the demonstration, as shown by the cost going to 0, but has to incur a cost by stopping short every time the demonstration leaves the imitator's own stable region. Once the system has reached this point in learning, it can within a very small number of trials optimize this deviation from the demonstration in the extremes to maintain stability, leading to the periodic cost fluctuations seen in Trial #35 in which the policy is close to converged.

As displayed for these PILCO trials, the cost is rapidly converging close to a zero mean with mean deviations at points where perfect imitation would violate stability. This demonstrates the approach's progress in learning a policy to follow the presented demonstration while avoiding becoming unstable. As indicated, the initial spike is the adjustment to the velocity of the initially stationary position of the imitator to the velocity of the presented demonstration. These trials display in simulation that an arbitrary demonstration can be learned. However, further research

needs to be conducted to determine whether this framework can learn a stationary policy that can successfully imitate any given demonstration without the need for additional trials for new demonstrations.

CHAPTER 4 - CONCLUSION

This thesis explored a method of translating human motions observed by a 3D camera to robot motion and provides supporting simulated data that the method explored is effective. It used existing technologies such as the Microsoft Kinect V2 to greatly assist in solving the correspondence problem for the observation of human behavior. To obtain the observations, it applied existing acquisition methods to gather data from the Kinect V2. Using this skeleton data, it then developed a framework to translate the observed skeleton to the NAO kinematics using trigonometric equations. The Gram-Schmidt process greatly assisted in the ease of the calculations. To address sensor noise resulting in joint "twitching", a Kalman Filter was integrated and through this implementation significant improvements in the reliability and accuracy of the acquired data was achieved. To further adapt the kinematically translated motions to address the dynamic and stability differences, an approach that simplifies the stability problem for the NAO robot to an inverted pendulum control problem and then applies a modified version of PILCO, a model-based policy search method based on cost optimization, that is adjusted to learn imitation tasks with stability constraints is proposed and applied in simulation. The thesis explored multiple learning methods and chose PILCO as an effective tool to learn a stable arbitrary demonstration due to its efficient use of data. Experiments with simple demonstration tasks show the ability of this system to learn control strategies that trade off imitation accuracy against stability considerations in an effective and efficient manner without the need for excessive amounts of system operation.

While this thesis introduces a set of viable techniques and demonstrates their potential, it is only the beginning of the research. In future work, we plan full integration of the two paradigms, PILCO and body imitation, into a complete imitation system for humanoid robots. For this, we plan to utilize Gazebo [20] physics simulation with the final goal of moving the learned imitation policy onto a real NAP robot. Gazebo simulates a real-life environment which will allow testing out the complete framework for adaptive human-robot motion transfer for complete body imitation.

REFERENCES

[1] Alissandrakis, C.L. Nehaniv, K. Dautenhahn, Imitation with ALICE: learning to imitate corresponding actions across dissimilar embodiments. IEEE Trans. Syst. Man Cybern. Syst.Hum. 32(4), 482–496 (2002)

[2] Marc Peter Deisenroth, Andrew McHutchon, Joe Hall, and Carl Edward Rasmussen. PILCO Code Documentation v0.9, July 4, 2013

[3] Marc P. Deisenroth and Carl E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In Proceedings of the International Conference on Machine Learning, pages 465 - 472, New York, NY, USA, June 2011. ACM.

[4] Muecke K, Hong D (2008) Investigation of an analytical motion filter for humanoid robots. In: Proceedings of the fifth international conference on ubiquitous robots and ambient intelligence

[5] David Wettergreen and Chuck Thorpe (1992) Gait Generation for Legged Robots. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems, July 1992

[6] Kofinas, N., Orfanoudakis, E. & Lagoudakis, M.G. Complete Analytical Forward and Inverse Kinematics for the NAO Humanoid Robot. J Intell Robot Syst 77, 251–264 (2015). https://doi.org/10.1007/s10846-013-0015-4

[7] Muecke, Karl, and Dennis Hong. Constrained Analytical Trajectory Filter for stabilizing humanoid robot motions. Intelligent Service Robotics 4, no. 3: 203-218. 2011

[8] E. L. Post (1946). "A variant of a recursively unsolvable problem" (PDF). Bull. Amer. Math. Soc. 52 (4): 264–269. doi:10.1090/s0002-9904-1946-08555-9.

[9] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision, 47(1/2/3):7-42, April-June 2002.

[10] Y. Kondo, S. Yamamoto and Y. Takahashi, "Real-Time Posture Imitation of Biped Humanoid Robot Based on Particle Filter with Simple Joint Control for Standing Stabilization," 2016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS), 2016, pp. 130-135, doi: 10.1109/SCIS-ISIS.2016.0039.

[11] Muscolo, Giovanni & Caldwell, Darwin & Cannella, Ferdinando. (2017). Calculation of the Center of Mass Position of Each Link of Multibody Biped Robots. Applied Sciences. 7. 10.3390/app7070724.

[12] Graf C., Röfer T. (2012) A Center of Mass Observing 3D-LIPM Gait for the RoboCup Standard Platform League Humanoid. In: Röfer T., Mayer N.M., Savage J., Saranlı U. (eds) RoboCup 2011: Robot Soccer World Cup XV. RoboCup 2011. Lecture Notes in Computer Science, vol 7416. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-32060-6_9

[13] K. Harada et al., "Toward human-like walking pattern generator," 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009, pp. 1071-1077, doi: 10.1109/IROS.2009.5354557.

[14] Andrew Witkin and Michael Kass. 1988. Spacetime constraints. In Proceedings of the 15th annual conference on Computer graphics and interactive techniques (SIGGRAPH '88). Association for Computing Machinery, New York, NY, USA, 159–168. DOI:https://doi.org/10.1145/54852.378507

[15] Cheney, Ward; Kincaid, David (2009). Linear Algebra: Theory and Applications. Sudbury, Ma: Jones and Bartlett. pp. 544, 558. ISBN 978-0-7637-5020-6.

[16] Kalman, R. E. 1960. “A New Approach to Linear Filtering and Prediction Problems,” Transaction of the ASME—Journal of Basic Engineering, pp. 35-45 (March 1960).

[17] Citation: Geerse DJ, Coolen BH, Roerdink M (2015) Kinematic Validation of a Multi-Kinect v2 Instrumented 10-Meter Walkway for Quantitative Gait Assessments. PLoS ONE 10(10): e0139913. https://doi.org/10.1371/journal.pone.0139913

[18] Stanford Artificial Intelligence Laboratory et al. (2018). Robotic Operating System. Retrieved from https://www.ros.org

[19] Nguyen, Chuong & Izadi, Shahram & Lovell, David. (2012). Modeling Kinect Sensor Noise for Improved 3D Reconstruction and Tracking. 10.1109/3DIMPVT.2012.84.

[20] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), 2004, pp. 2149-2154 vol.3, doi: 10.1109/IROS.2004.1389727.