

Copyright  
by  
Steven Jens M. Jorgensen  
2017

**The Thesis committee for Steven Jens M. Jorgensen**

**Certifies that this is the approved version of the following thesis:**

**Human Detection, Gesture Recognition, and Policy  
Generation for Human-Aware Robots**

APPROVED BY

SUPERVISING COMMITTEE:

---

Luis Sentis, Supervisor

---

Andrea Thomaz

**Human Detection, Gesture Recognition, and Policy  
Generation for Human-Aware Robots**

by

**Steven Jens M. Jorgensen, B.S.E.**

**THESIS**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**Master of Science in Engineering**

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2017

Dedicated to my parents, siblings, and fiancée.

## Acknowledgments

First, I would like to thank my adviser, Dr. Luis Sentis, who has been very supportive and patient with my progress. I would also like to thank Dr. Andrea Thomaz for graciously agreeing to be the second reader for my master's thesis. I want to recognize and acknowledge NASA for their NASA Space Technology Research Fellowship (NSTRF) program for supporting my graduate studies. Finally, thank you to Kimberly Hambuchen from NASA Johnson Space Center for being my NSTRF mentor and giving me access to Valkyrie's MultiSense SL sensor.

I would like to thank my parents and fiancée for being patient with my absence from home as I pursue my graduate studies. I am motivated to continue working because of them. I also want to recognize the many friends and colleagues I have made in graduate school. They truly have been a constant source of support and enriched my experience.

Thank you to Chien-Liang Fok who helped me on many different occasions. Thank you for helping me transition to Austin, for inviting me to your family gatherings, and for your helpful mentorship in my first two years in the lab. Thank you to Gwendolyn Johnson for the useful discussion we had about Whole-Body Control and the prospects of being a member of the Human-Centered-Robotics Lab (HCRL). Her answers to my inquiry about controls and graduate school was very helpful in my first year.

I would like to thank my lab's senior members for their friendship and more. Thank you to Nicholas Paine for his wisdom and advice about graduate

school and work-life balance. Thank you to Donghyun Kim who has been a constant mentor, teacher, and friend in many aspects of robotics and life. Thank you to Ye Zhao for his teachings about LTL, graduate school and controls, and whose kindness and intelligence always amazes me. Thank you to Kwan-Suk Kim for always being open to talk about his research with Trikey as well as for teaching me Gaussian Mixture Models. Thank you to Gray Thomas whose insight and feedback often gives me a direction to improve my work. Thank you to Travis Llado for being an inspiration by showing me the standards of what a true mechatronics engineer constitutes. Thank you also to Kenan Isik for his friendship and his generosity with sharing his drinks, which caffeinated and motivate me in long nights.

I also want to thank the new lab members in 2016. Thank you to Orion Campbell for the open-minded intellectual discussions as well as our continuing joint work whenever possible. Thank you to Bingham He for our discussions regarding LTL control and general topics about culture and food. Thank you to Rachel Schlossman for the fun discussions, and for showing me what great work ethic looks like. Thank you also to Jaemin Lee for showing me his approach to research as well as our discussions about Whole Body Control as I always found my interactions with him intellectually rewarding and fun. Thank you to Junhyeok Ahn for the fun dinners, lunch, and coffee break, as well as his friendship, honesty, and work ethic demonstration. Thank you also to Minkyu Kim for his genuine kindness, friendship, and meaningful discussions during our commute to classes. Finally, thank you to Dorothy Jorgensen for the honest and fun discussions about NASA, graduate school, and life.

I also want to thank fellow graduate students Jack Hall, Prashant Rao,

and Taylor Niehues for their consistent availability every Wednesday afternoon for company. Thank you to Ajinkya Jain for the interesting and fun discussions about learning and manipulation and Alex Guitierrez for our computer science discussions. I always found the scientific and non-scientific discussion to be a constant source of delight, and I felt that being surrounded by great people was a good way to get motivated with research.

It's been a great and continuing pleasure to know and work with everyone. My time at graduate school would not have been the same. Thank you all!

# Human Detection, Gesture Recognition, and Policy Generation for Human-Aware Robots

Steven Jens M. Jorgensen, M.S.E.  
The University of Texas at Austin, 2017

Supervisor: Luis Sentis

For robots to be deployable in human occupied environments, the robots must have human-awareness and generate human-aware behaviors and policies. This thesis posits that a human-aware robot must be capable of (1) human detection and tracking, (2) human action or intent recognition and (3) intelligent, human-aware action generation. This work presents and evaluates a methodology for each stated capability.

In Chapter 2, a method for practical side-by-side human detection for the Valkyrie robot using the Multisense SL sensor is presented. An explanation of why current off-the-shelf techniques are not suitable and a depth-based algorithm using point cloud descriptors and a Random Forest classifier for detecting humans under occlusion, in close proximity, in varying sparsity, and in random poses on the Multisense SL sensor are presented.

In Chapter 3, action recognition of arm motion gestures is framed as a supervised learning problem. A popular technique for gesture representation with dynamic movement primitives (DMPs) and its classification using

Gaussian Mixture Models (GMMs) is explored. The approach is tested under various hypotheses to understand the intricacies of using DMPs for movement representation. The following findings are reported: (a) recognition rate is sensitive to the number of basis weights, (b) DMPs can be used to recognize two linear motions, (c) rhythmic gestures can be differentiated with the discrete formulation of DMPs, and (d) DMPs can represent static-type gestures.

In Chapter 4, a novel technique for (a) representing Human-Robot-Interaction as a dynamical system, and (b) using model predictive control to generate control policies is presented. The approach is motivated by using a scenario in which an Assistive Robot must be productive by bringing work to the human but must also be mindful of the human's workload. By modeling the interaction as a dynamical system, advances in control theory can be leveraged to generate useful control policies.

# Table of Contents

<b>Acknowledgments</b>	<b>v</b>
<b>Abstract</b>	<b>viii</b>
<b>List of Tables</b>	<b>xiii</b>
<b>List of Figures</b>	<b>xiv</b>
<b>List of Algorithms</b>	<b>xvii</b>
<b>Chapter 1. Introduction</b>	<b>1</b>
1.1 Thesis Contributions . . . . .	1
<b>Chapter 2. Human Detection with the MultiSense SL Sensor on Valkyrie</b>	<b>4</b>
2.1 Background and Motivation . . . . .	4
2.2 Required Capabilities . . . . .	8
2.3 Technical Approach Overview . . . . .	9
2.4 Feature Descriptions . . . . .	11
2.4.1 Local Surface Normal Calculations . . . . .	11
2.4.2 FPFH Cluster feature . . . . .	13
2.4.3 Other Features . . . . .	15
2.5 Random Forest Classification Learning . . . . .	21
2.6 Results and Discussion . . . . .	23
2.6.1 Learning Curves, Precision/Recall, and Calculation Time	23
2.6.2 Empirical Results . . . . .	26
2.7 Evaluation and Future work . . . . .	28

<b>Chapter 3. Action Recognition: Gesture Recognition with DMPs and GMMs</b>	<b>31</b>
3.1 Related Works . . . . .	35
3.2 Technical Background . . . . .	37
3.2.1 Dynamic Movement Primitives (DMPs) for Gesture Representation . . . . .	37
3.2.2 Gaussian Mixture Models (GMMs) for Gesture Recognition	39
3.3 Experiment Methodology for DMP and GMMs . . . . .	40
3.3.1 Gesture Data Gathering . . . . .	40
3.3.2 Gesture Feature Representation . . . . .	41
3.3.3 GMM Supervised Classification . . . . .	43
3.3.4 GMM Unsupervised Classification . . . . .	44
3.4 Experiment and Results . . . . .	45
3.4.1 Unsupervised GMM Performance . . . . .	45
3.4.2 Supervised GMM Performance . . . . .	46
3.5 Discussion . . . . .	49
3.6 Future Work . . . . .	52
<b>Chapter 4. Action Generation: Decision Making with Model Predictive Control</b>	<b>54</b>
4.1 Related Works . . . . .	58
4.2 HRI as Linear Dynamical Systems . . . . .	59
4.3 Policy Generation via Model Predictive Control . . . . .	59
4.4 Assistant Robot Scenario as a Linear Dynamical System . . . . .	61
4.4.1 World State and Actions . . . . .	61
4.4.2 State Transition Matrix . . . . .	61
4.5 World Constraints formulated as Mixed-Integer Constraints for Model Predictive Control . . . . .	64
4.5.1 Scenario Constraints . . . . .	65
4.6 Results . . . . .	69
4.7 Discussion . . . . .	69
<b>Chapter 5. Summary and Future Outlook</b>	<b>71</b>

<b>Bibliography</b>	<b>73</b>
<b>Vita</b>	<b>82</b>

## List of Tables

2.1	Table summarizing the features used for human segmentation	17
2.2	Table summarizing the classifier’s precision and recall performance for each label. The average labeling accuracy is the classifier’s performance on a k-fold cross-validation set. The recall performance for classifying a candidate human cluster is bolded to indicate that human clusters are extracted at a high rate even if non-human clusters are classified as humans. . . . .	26
3.1	DMP Learning Parameters . . . . .	38
3.2	Unsupervised GMM . . . . .	45
3.3	Supervised GMM on all data sets . . . . .	46
3.4	Supervised GMM on Cross Validation data set . . . . .	46

## List of Figures

2.1	Multisense SL RGB-D distortion: The raw camera image (top left) shows a human placing his hand in front of the Multisense SL sensor. The yellow box indicates that the human’s hand is visible both in the raw camera image and the lidar point cloud data. However, the orange box shows that the RGB-D point cloud data is heavily distorted. . . . .	5
2.2	Multisense SL lidar data sparsity: Due to the scanning pattern of the Multisense SL’s hokuyo sensor (center image), the lidar data introduces sparsity as lidar speed, and distance and angle from the sensor increases (top image). . . . .	6
2.3	Multisense SL Overhead Visibility: The green region shows where both RGB-D and lidar point cloud data are visible, the blue region shows where lidar point cloud data is visible, and the yellow region is an approximation of where side-by-side collaboration occurs. . . . .	7
2.4	Visibility comparison between RGB-D point cloud data versus lidar data from Valkyrie’s Multisense SL sensor . . . . .	9
2.5	Visualizing how Human Points are extracted from Lidar data: (a) The original point cloud data is sub-sampled by voxelizing nearby points as one voxel. (b) The points are separated into $m$ layers after extracting the local normals for each voxel point. (c) Euclidean clustering is used to cluster points together for each layer. (d) Features of each clustered points are extracted and classified as “Candidate Humans” (green) or not. Then, “Candidate Humans” are clustered again and semantic classification is performed. A bounding box is returned for all classified human points. . . . .	12
2.6	Extracting the angular difference between two local normals. This image is borrowed from [49]. . . . .	13
2.7	An illustration of a 33-bin FPFH histogram with each bin having a width of $2\pi/11$ . . . . .	16
2.8	Random Forest Classifier: The candidate human cluster classification error rate vs number of training examples. The plot for the training error and cross-validation (CV) error are shown. . . . .	25
2.9	Calculation Time vs Number of Voxels . . . . .	27

2.10	Human Detection Empirical Results. Green points indicate candidate human points and the purple bounding box indicates the region where a human is detected. Each sub-figure demonstrates the algorithm’s detection capability. See text for details.	30
3.1	The eight types of demonstrated gestures are shown. The sub-figures indicate (a) a static gesture, (b)-(f) five discrete gestures, and (g) and (h) are two rhythmic gestures. The gestures were made using a Kinect that recognized the x-y-z position of the AR marker held by the demonstrator. Each gesture demonstration is plotted as a single color. The static gesture, (a), is a demonstration where the marker never moves. (b) and (d) are discrete letter-type gestures which is used in existing DMP literature to show movement recognition [23]. (c) is a triangle shape gesture to test the ability of the DMP to recognize gestures with almost equal starting and ending positions. (e) and (f) are linear gestures with different starting and ending positions to test if DMPs can discriminate between two spatially different discrete motions. Finally, (g) and (h) represent a continuous circular and waving motion respectively. For each sub-figure, each colored trajectory represents the trajectory of a single demonstration.	34
3.2	The similarity matrix of all the gestures is visualized as a colormap. Each cell represents the similarity between any two gestures where colors closer to 1 indicates high similarity and those below 0 have minimum similarity.	42
3.3	Spatially Different Demonstrations	46
3.4	Linear Discrete Motion Gestures can be differentiated when $K$ is high such that the DMP’s attractor dynamics move faster than the actual demonstration making the forcing function non-zero.	49
3.5	Recognizing static gestures is possible by setting the goal position away from the user and using features such as arm angle relative to the body of the user.	50
4.1	Assistive HRI Scenario: An assistive robot must bring deliverables from the inventory station (I.S.) to the human’s work station area (W.S.). A mindful robot will ensure that the human is never overworked.	58
4.2	Fluid Analogy for the Assistive Scenario.	62

4.3 Assistant Robot Simulation Results: For both (a) and (b), the robot worries more about the human's workload more than its own battery levels and productivity. Note that  $u_{pu} = u_{ipu} + u_{wpu}$  and  $u_{do} = u_{ido} + u_{wdo}$  to indicate the total deliverable pick up and drop off actions respectively. Also, the W.S. and I.S. are located at  $l_{ws} = 9$  and  $l_{is} = 1$  respectively. In (a), the robot initially drops off the deliverables it is carrying to give the human work and proceeds to charge its own batteries while slowly dropping off more work to the human. In (b), the robot notices that the human is overworked and proceeds to remove work from the human at the cost of the robot's own productivity until the human's workload becomes manageable. The robot also charges its low battery levels to remain operational. Then, the robot proceeds to slowly drop off work to the human at a manageable rate, which also makes the robot's perception of its own productivity to rise again. . . . .

## List of Algorithms

1	Human Detection Algorithm for MultiSense SL's Hokuyo Sensor	10
2	Decision Tree Learning . . . . .	21
3	Random Forest Learning . . . . .	23
4	Random Forest Classification . . . . .	24

# Chapter 1

## Introduction

Current high-performance industrial grade robots are unsafe to work with humans. These robots are pre-programmed to do tasks as quickly possible with minimal concern for its environment. They lack human-awareness and react on an out-of-context basis. This thesis claim that a successful collaborative robot must have the following capabilities:

1. An ability to detect and track the pose of fellow collaborators.
2. An ability to recognize the intent and actions and mental states of fellow collaborators
3. An ability to generate intelligent actions to accomplish task objectives and support fellow collaborators

### 1.1 Thesis Contributions

In chapter 2, a human detection algorithm is presented for the Multisense SL sensor used on the NASA Valkyrie robot. The Multisense SL has two primary sensors, a stereo camera which can output a colored point cloud data from the RGB-D data and a spinning Hokuyo lidar sensor which generates a 3D point cloud data with varying cloud density. The detection algorithm was created to handle the data points generated by the sensor's spinning Hokuyo

Lidar sensor since the stereo camera has limitations both in terms of visibility in regions where side-by-side interactions occur and distortion errors when an object is too close to the camera. The detection algorithm accepts any 3D point cloud data and outputs a set of 3D bounding boxes on regions where human points are present. The algorithm is able to detect humans under occlusions, varying point cloud sparsity, and in close proximity. However, due to features extracted from the point cloud, the approach has high recall and low precision classification. Thus, while human points are always extracted by the algorithm, non-human points are sometimes classified as humans.

The field of action and intent recognition is vast, and it is difficult to provide an all-purpose algorithm. One subset of general action recognition is gesture recognition. In chapter 3, a methodology for recognizing arm motion gesture recognition is addressed. Arm motion gestures are represented as a Dynamic Movement Primitive (DMP). Recognition is performed by extracting the basis weights of the gesture's DMP forcing function and then performing inference over a trained Gaussian Mixture Model (GMM) to identify the most likely gesture. Using DMPs and GMMs are common in literature, but this thesis tests many hypotheses to understand the discriminative power of DMPs and GMMs for gesture recognition. It was found that recognition rates are sensitive to the number of basis weights, DMPs can discriminate between two different linear motions, and that DMPs can be used for static gesture recognition.

Finally chapter 4 presents a methodology for generating intelligent, human-aware robot control policies using Model Predictive Control (MPC) with mixed integer constraints. Robot decision making has traditionally been framed as a either a motion planning problem or as a Markov Decision Prob-

lem. Here, this work presents a new method of representing Human-Robot-Interaction (HRI) scenarios as dynamical systems. Once represented as a dynamical system, advances in the field of control theory can be leveraged. The methodology is motivated by an assistive robot scenario in which the robot attempts to maximize productivity by delivering work to the human, but also ensures that the human is never overworked. Simulation results show that the approach is able to generate useful control behaviors from maximizing a cost function with a notion for productivity and human workload.

## Chapter 2

# Human Detection with the MultiSense SL Sensor on Valkyrie

In this chapter, an algorithm to detect humans in various poses and under significant occlusions for Valkyrie’s [43] Multisense SL (MSL) sensor [9] is presented. The MSL sensor is a multi-modal sensor capable of providing RGB-D data from its cameras and point cloud data from a spinning Hokuyo laser (lidar) sensor. Here, we describe why off-the-shelf solutions with the Kinect is not applicable, why the RGB-D data is not used, and how to semantically detect and extract human points from the MSL’s Hokuyo laser sensor.

### 2.1 Background and Motivation

For many side-by-side collaboration tasks, two or more humans working together naturally communicate via verbal cues and nonverbal cues. While detecting the presence of a human is not necessary for the former, the latter means of communication requires an ability to detect, identify, and track poses of human partners. Research has shown that nonverbal communication are important in human-robot collaboration [7] [20]. Thus, human detection is a necessary capability for a robot engaging in side-by-side collaboration with a human. Without this capability, a robot will not be able to naturally assist the human as another human partner would.

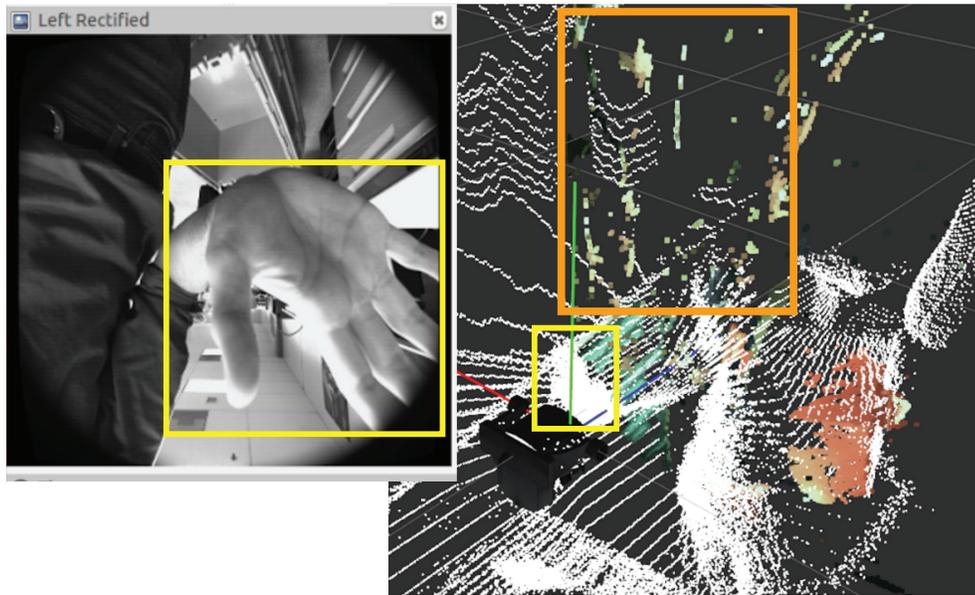


Figure 2.1: Multisense SL RGB-D distortion: The raw camera image (top left) shows a human placing his hand in front of the Multisense SL sensor. The yellow box indicates that the human’s hand is visible both in the raw camera image and the lidar point cloud data. However, the orange box shows that the RGB-D point cloud data is heavily distorted.

Furthermore, human detection is an essential component of many Human-Robot-Interaction (HRI) scenarios such as pedestrian detection [14] for autonomous driving, navigation with social-awareness [38], learning from demonstration [27], and others. Due to the extensive utility of human detection and tracking, many real-time algorithms exist for tackling this problem [29] [14] [55].

A potential solution for human tracking is to require the human to wear specialized markers so that the visual sensors can locate the markers on the human body and infer the human’s kinematic pose. However, using markers has issues as well. The markers may be sensitive to lighting and other

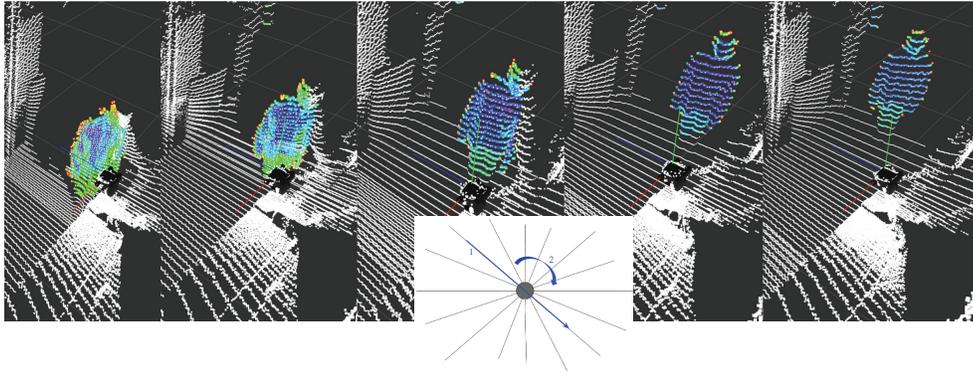


Figure 2.2: Multisense SL lidar data sparsity: Due to the scanning pattern of the Multisense SL’s hokuyo sensor (center image), the lidar data introduces sparsity as lidar speed, and distance and angle from the sensor increases (top image).

environmental conditions. Wearing multiple markers can also be cumbersome to the user.

Recently, with the advent of a cheap RGB-D sensor, the Microsoft Kinect [60], the rise of open-source robotics, the Robot Operating System (ROS) [42], and an open-sourced implementation of human pose detection and tracking, [36] [55], researchers have gained access to the necessary off-the-shelf hardware and software implementation to bootstrap their needs for reliably tracking and detecting human poses.

However, the Kinect’s implementation for human tracking has limited capability. First, due to hardware limitations, the human must be at a minimum distance away from the sensor to prevent point cloud distortion. Second, due to their feature selection, their algorithm requires that the humans unobstructed full-body must face the sensor. These two requirements are too restrictive for a robot engaging in side-by-side collaboration, as humans might be partially occluded when they are on the periphery of the robot’s vision.

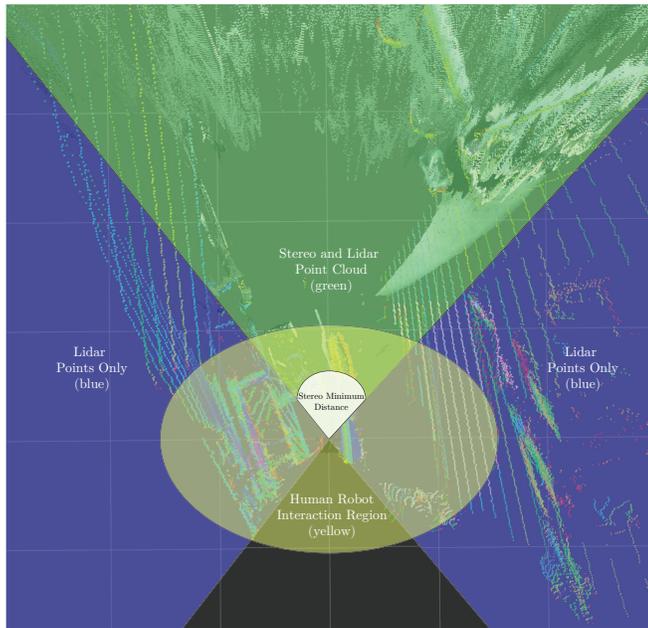


Figure 2.3: Multisense SL Overhead Visibility: The green region shows where both RGB-D and lidar point cloud data are visible, the blue region shows where lidar point cloud data is visible, and the yellow region is an approximation of where side-by-side collaboration occurs.

While it is possible to place multiple Kinects externally from the robot, robots with on-board sensing have more utility from a deployment point of view. It's also possible to place multiple kinects on the robot, however this approach introduces unnecessary costs and does not address the issue of the minimum distance requirement of the Kinect sensor.

Additionally, the Kinect's tracking algorithm is also not immediately usable for the Valkyrie robot [43] engaging in side-by-side collaboration with humans. Similar to the kinect's hardware limitations, the MSL's RGB-D requires a minimum distance to properly perform point cloud correspondence without introducing distortion (Figure 2.1) .

Note that the Kinect algorithm also assumes very dense point clouds to generate an appropriate depth map. Unfortunately, the algorithm can only work on the RGB-D data of the MSL sensor, as the lidar data introduce significant point cloud sparsity depending on the lidar speed and where the human is standing with respect to the sensor (Figure 2.2). Furthermore, the MSL’s RGB-D data has a limited viewing angle and will be blind to areas where physical HRI occurs and side-by-side collaboration is expected (Figure 2.3).

When comparing the point cloud data available with the MSL, data from the lidar shows that it has no minimum distance limitation (Figure 2.1) and has a wider viewing angle (Figure 2.4), which can enable human detection for side-by-side collaboration.

Given the limitations of the Kinect algorithm and the challenges that come with using the MSL sensor on Valkyrie, there is a need for creating a real-time human detection algorithm for the MSL using its lidar data. This is not immediately easy as in addition to the sparsity issue (Figure 2.2), the scanning pattern of the Hokuyo sensor introduces another problem. Namely, it needs to rotate 180 degrees before obtaining a full visual update. Thus, the hokuyo needs to be spun as fast as possible to obtain the latest update of its environment which further increases point cloud sparsity.

## 2.2 Required Capabilities

In order to make the human detection algorithm for the MSL practical, it is important to state its required capabilities. In essence, the algorithm must be able to detect humans under:

1. varying point cloud sparsity.

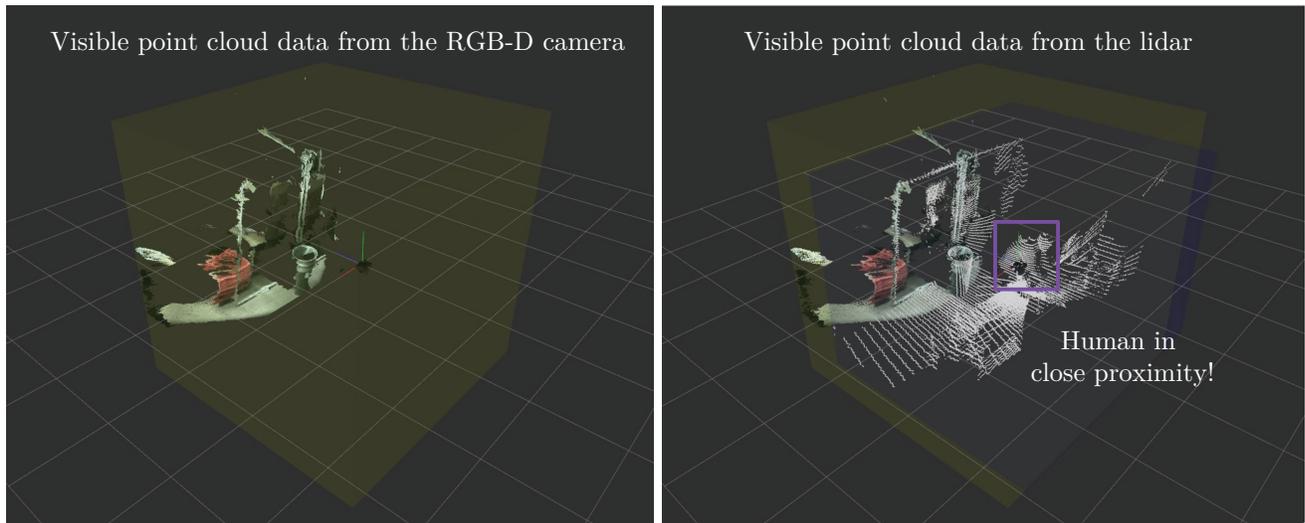


Figure 2.4: Visibility comparison between RGB-D point cloud data versus lidar data from Valkyrie’s Multisense SL sensor

2. regions where reliable RGB-D data is not available.
3. occlusions of up to 50 % of the body.
4. varying poses.
5. real-time constraints (less than 30ms [19]).

### 2.3 Technical Approach Overview

Algorithm 1 details the overview procedure for semantically extracting human point clouds from the MSL’s lidar data. Figure 2.5 gives a visualization of Algorithm 1.

The separation of point clouds into  $m$  different layers is inspired from [18] which used the same method to recognize occluded humans better. The algorithm presented here has a couple of notable differences. First, their im-

---

**Algorithm 1** Human Detection Algorithm for MultiSense SL’s Hokuyo Sensor

---

- 1: **procedure** DETECTHUMANS
  - 2:   Voxelize point cloud data
  - 3:   For every voxel, compute its local normal
  - 4:   Separate points into  $m$  layers along the z- axis
  - 5:   For each layer, cluster the points using a segmentation algorithm (eg: Euclidean distance clustering)
  - 6:   Calculate the features of each cluster and construct its feature vector.
  - 7:   Using the cluster’s feature vector, use a classification algorithm (eg: Random Forests) to classify whether the cluster is part of a human or not.
  - 8:   Clusters marked as humans are labeled as “Candidate Humans”
  - 9:   Cluster “Candidate Humans” points
  - 10:   Create a bounding box for each clustered candidate human points
  - 11:   Extract features of points inside the bounding box and classify the points as human or not.
  - 12:   Return the set of bounding boxes enclosing human points
- 

plementation relies on using RGB-D points from the Kinect. However, here we use sparser data from the MSL’s lidar sensor. Second, instead of using a histogram of Local Surface Normals (LSNs) that depend on the orientation of the point cloud with respect to the sensor origin, we instead use the Fast Point Feature Histogram (FPFH) [50] descriptors available from the Point Cloud Library [51] as our primary feature vector. Instead of creating a histogram for each normal axis direction  $(x,y,z)$ , the FPFH feature descriptor instead relies on angular differences between neighboring local normals. This makes the feature agnostic to the origin of the sensor. Finally, instead of using the Nearest Neighbors to reconstruct the human points, we instead use a clustering algorithm, which results to a similar performance, with potential extensions. At the naivest solution, a simple Euclidean clustering algorithm can reconstruct the human points. However, more sophisticated clustering algorithms that utilize probabilistic methods over temporal data and semantic data exist [19]

and can be substituted to the clustering algorithm used.

## 2.4 Feature Descriptions

In this section we describe the feature vector we use to semantically extract human points from non-human points for a given point cloud data. As algorithm 1 describes, we cluster points for every layer.

Our approach relies on classifying whether a given cluster,  $c$ , belongs to a candidate human or not. To perform this binary classification, we describe how to construct our feature vector,  $f(c)$ , of a given cluster.

### 2.4.1 Local Surface Normal Calculations

For a given radius around a point, the FPFH descriptor creates a histogram of the angular differences between neighboring local normals. Thus, the point’s local normal must first be computed. The computation of a point’s local surface normal utilizes Principal Component Analysis (PCA) [25], a standard statistical technique used for dimensionality reduction.

The procedure is as follows. For each point,  $p \in \mathbb{R}^3$ , its nearest neighbors within a specified spherical radius,  $r_p \in \mathbb{R}$  are extracted. The centroid  $\bar{p} \in \mathbb{R}^3$  of all the points within the radius is calculated.

$$\bar{p} = \frac{1}{k} \sum_{i=1}^k p_i \quad (2.1)$$

The  $3 \times 3$  covariance,  $C \in \mathbb{R}^{3 \times 3}$ , matrix of the spherical region is calculated using the following formula.

$$C = \frac{1}{k} \sum_{i=1}^k (p_i - \bar{p}) \cdot (p_i - \bar{p})^T \quad (2.2)$$

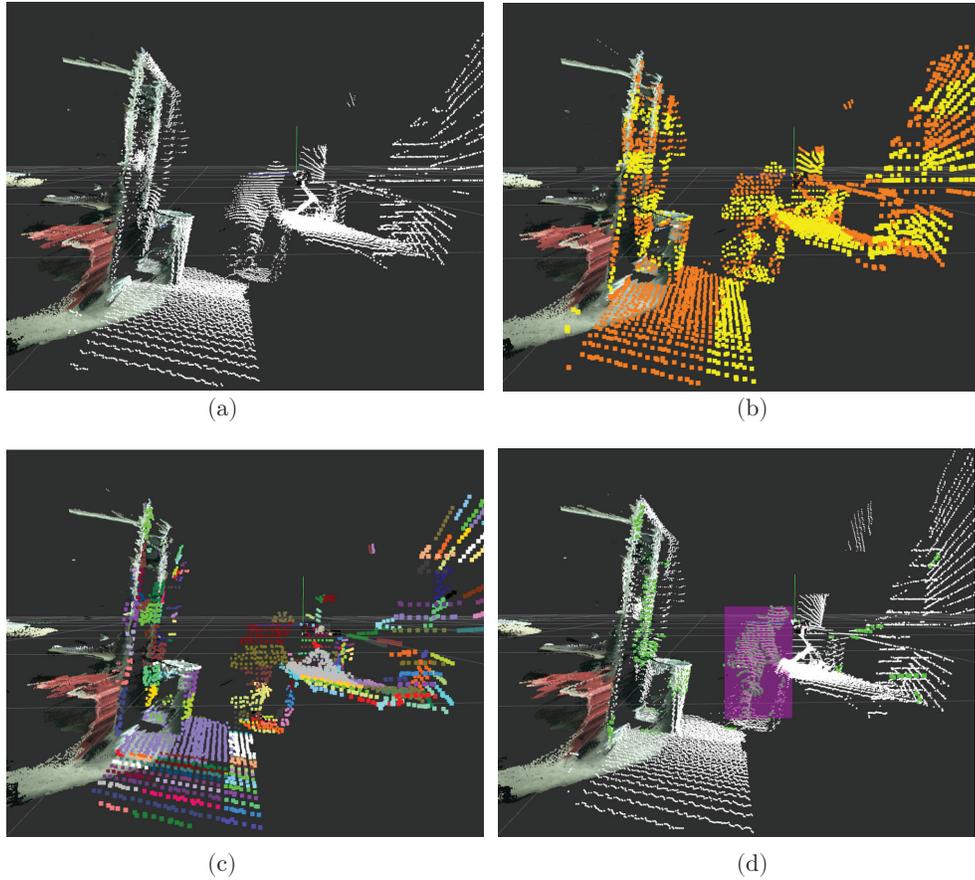


Figure 2.5: Visualizing how Human Points are extracted from Lidar data: (a) The original point cloud data is sub-sampled by voxelizing nearby points as one voxel. (b) The points are separated into  $m$  layers after extracting the local normals for each voxel point. (c) Euclidean clustering is used to cluster points together for each layer. (d) Features of each clustered points are extracted and classified as “Candidate Humans” (green) or not. Then, “Candidate Humans” are clustered again and semantic classification is performed. A bounding box is returned for all classified human points.

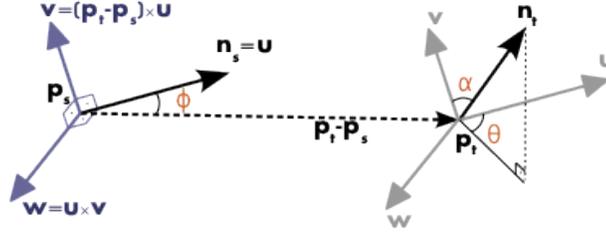


Figure 2.6: Extracting the angular difference between two local normals. This image is borrowed from [49].

Using Singular Value Decomposition, eigenvalues,  $\lambda_i$ , and eigenvectors  $v_i$  of  $C$  are calculated. The eigenvalues are ordered in increasing order. The eigenvector corresponding to the smallest eigenvalue is the local normal direction of the point  $p$ . This is intuitive as in PCA, the two largest eigenvectors describe the direction of greatest change. These two eigenvectors are perpendicular to each other. Since  $C \in \mathbb{R}^3$ , the last eigenvector must be perpendicular to the first two vectors and sufficiently describes the local normal of point  $p$ .

## 2.4.2 FPFH Cluster feature

The Fast Point Feature Histogram (FPFH) [50] is a histogram-based feature descriptor for point cloud data. The FPFH is an improvement over the Point Feature Histogram (PFH) [49]. The goal of FPFH and PFH is to capture the local curvature of a point using its local  $k$ -neighborhood of points.

FPFH is computed in two steps. First, for each point,  $p_q$ , compute the angular differences between the local normals of nearby points, expressed as three angles  $(\alpha, \phi, \theta)$ . The angular differences are binned into a histogram. This is referred to as the point's Simplified Point Feature Histogram (SPFH). In the second step, for each  $p_k$  neighbor of  $p_q$ , its neighbors are recomputed

and SPFH is performed on  $p_k$ . The SPFH of  $p_q$ 's neighbors are used to weigh the final histogram of  $p_q$ .

To compute the angular difference between two point's local normals, we first construct a fixed-coordinate frame on one of the points using the vector of the local normals and euclidean distance vector between the two points (See Figure 2.6). Let  $p_t$ , and  $p_s$  be two points with local normals  $n_t$  and  $n_s$  respectively. The angular difference of the local normals between these two points can be computed by defining a tuple  $(u, v, w)$ , where

$$u = n_s, \tag{2.3}$$

$$v = u \times \frac{(p_t - p_s)}{\|p_t - p_s\|_2}, \tag{2.4}$$

$$w = u \times v. \tag{2.5}$$

Then we compute the angular differences  $(\alpha, \phi, \theta)$  as

$$\alpha = v \cdot n_t, \tag{2.6}$$

$$\phi = u \cdot t \frac{(p_t - p_s)}{\|p_t - p_s\|_2}, \tag{2.7}$$

$$\theta = \text{atan}(w \cdot n_t, u \cdot n_t). \tag{2.8}$$

The SPFH is obtained by binning the angular differences into a histogram that is evenly spaced. With the SPFH properly defined, we can now define the FPFH of each query point  $p_q$  as follows.

$$FPFH(p_q) = SPFH(p_q) + \frac{1}{k} \sum_{i=1}^k \frac{1}{w_i} SPFH(p_i), \tag{2.9}$$

where  $w_i$  is a scalar weight defined as the euclidean distance between  $p_q$  and  $p_i$ . This weight is used to place bins more importance on points nearby  $p_q$  and

less on the neighbors of  $p_i$ . The Point Cloud Library’s FPFH implementation uses 33 bins, using 11 bins for each angle where each angle has a bin width of  $2\pi/11$ .

So far, only the FPFH feature of a point has been described. However, we need to classify whether a *cluster* is a candidate human. Thus, a feature descriptor for the cluster is needed.

To get the cluster’s FPFH feature, we sum all the FPFH feature histograms of each point  $p_j$  in the cluster. Formally, let  $S_c$  be the set of all points  $p$  belonging to cluster  $c$ ,  $f_1(c) \in \mathbb{R}^{33}$  be the cluster’s FPFH feature vector and let  $p_j \in S_c$  be a point inside the cluster. Then a cluster’s FPFH feature is

$$f_1(c) = \frac{1}{W} \sum_{p_j \in S_c} FPFH(p_j), \quad (2.10)$$

where  $W$  is a scaling factor such that the integral of the FPFH feature vector is unity. Other normalization schemes are possible such as requiring that the mean of all training examples for each bin is centered at 0. In Figure 2.7, normalization was performed using the maximum of each training example’s corresponding bin for illustration purposes.

### 2.4.3 Other Features

While the previous section described our most discriminative feature, here we construct other features used to describe cluster  $c$ . Table 2.4.3 summarizes all the features used.

The cluster covariance feature  $f_2(c) \in \mathbb{R}^9$  is inspired from its utility by other pedestrian detection algorithms [26]. To define  $f_2(c)$  We simply take the

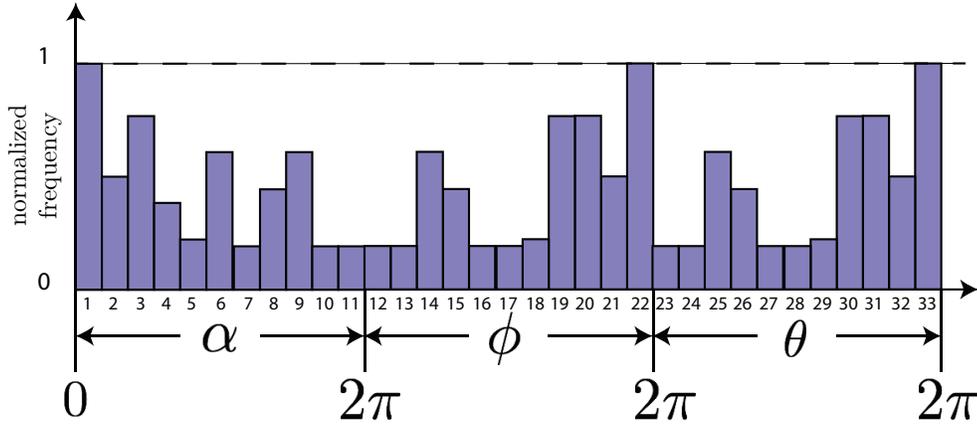


Figure 2.7: An illustration of a 33-bin FPFH histogram with each bin having a width of  $2\pi/11$ .

cluster's covariance,  $C(c) \in \mathbb{R}^{3 \times 3}$  and roll out its elements.

$$f_2(c) = [C_{11}(c), C_{12}(c), \dots, C_{33}(c)]^T, \quad (2.11)$$

with  $C_{ij}$  is the  $i$ th row and  $j$ th column of the matrix  $C(c)$ .

To calculate features  $f_i(c)$  from  $i = 3$  to  $i = 5$ , we first calculate the cluster's bounding box. The bounding box's  $x, y, z$  Cartesian positions are calculated by first initializing  $x_b^{\min}, y_b^{\min}, z_b^{\min} = \infty$  and  $x_b^{\max}, y_b^{\max}, z_b^{\max} = -\infty$ . These coordinates specify the axis-aligned corners of the bounding box. Then for each point  $p_j$  in cluster  $c$ , update  $\{x, y, z\}_b^{\{min, max\}}$  accordingly.

The length, width, and height of the cluster's bounding box can be extracted by specifying

$$(l_b^c, w_b^c, h_b^c) = \text{abs}(x_b^{\max} - x_b^{\min}, y_b^{\max} - y_b^{\min}, z_b^{\max} - z_b^{\min}). \quad (2.12)$$

	Features	Dimensions
$f_1$	FPFH (Fast Point Feature Histogram)	33
$f_2$	Cluster Covariance	9
$f_3$	Axis-Aligned Side Ratios	2
$f_4$	Axis-Aligned Side Area	1
$f_5$	Bounding Box Diagonal	1
$f_6$	Bounding Box Volume	1
$f_7$	Variance Ratios	2
$f_8$	Eigenvalue Ratio	1
$f_9$	x-y Best Fit Radius	1
$f_{10}$	x-y Distance from Sensor	1
$f_{11}$	x-y Angle from Sensor	1

Table 2.1: Table summarizing the features used for human segmentation

We now order the side lengths of the bounding box in increasing order. Define the tuple  $(l_\alpha^c, l_\beta^c, l_\gamma^c)$ , which indicate the side lengths of the axis aligned bounding box of cluster  $c$  in increasing order. This ordering is important to retain independence from the cluster’s position with respect to the sensor origin.

The cluster features  $f_3(c), f_4(c), f_5(c)$  that describe the axis-aligned bounding box side ratios, side area and diagonal can be defined as follows.

$$f_3(c) = \left[ \frac{l_\beta^c}{(l_\alpha^c + 1)}, \frac{l_\gamma^c}{(l_\alpha^c + 1)} \right] \quad (2.13)$$

$$f_4(c) = l_\alpha^c \cdot l_\beta^c \quad (2.14)$$

$$f_5(c) = \left\| [l_b^c, w_b^c, h_b^c] \right\|_2 \quad (2.15)$$

The bounding box volume,  $f_6(c)$  is defined as

$$f_6(c) = l_b^c \cdot w_b^c \cdot h_b^c. \quad (2.16)$$

Next, the variance and eigenvalue ratios which describe features  $f_7(c)$  and  $f_8(c)$  are an attempt towards modeling the distribution of the cluster's points, which have also been shown to be helpful with classifying pedestrians in a point cloud [26]. Similar to features  $f_3(c)$  and  $f_4(c)$ , we will order the variances and eigenvalues. The cluster's axis aligned variances  $(C_{11}(c), C_{22}(c), C_{33}(c))$  have already been calculated when extracting the cluster's covariance matrix. We order these variances in increasing order defined as  $(C_\alpha(c), C_\beta(c), C_\gamma(c))$ . Similarly, we extract the eigenvalues of the cluster's covariance matrix and order them in increasing order as  $(\lambda_\alpha(c), \lambda_\beta(c), \lambda_\gamma(c))$ .

We now define the cluster's Variance ratios and Eigenvalue ratios features,  $f_7(c)$  and  $f_8(c)$  respectively, as follows.

$$f_7(c) = \left[ \frac{C_\beta(c)}{(C_\alpha(c) + 1)}, \frac{C_\gamma(c)}{(C_\alpha(c) + 1)} \right] \quad (2.17)$$

$$f_8(c) = \frac{\lambda_\beta(c)}{(\lambda_\alpha(c) + 1)} \quad (2.18)$$

Next, the circularity of the cluster is extracted. Humans are mostly cylindrical in shape when standing, and one way to capture this cylindrical feature of humans is to estimate the best fitting  $x - y$  radius of the cluster. The  $x - y$  radius is the best fit circle when the cluster's points are projected to the  $x - y$  plane. Typically the  $x - y$  plane is parallel to the ground. To obtain this we first project all the points of cluster  $c$  to the  $x - y$  plane by ignoring the point's z-coordinate axis. As described in [1], the best fit circle can be extracted by parameterizing the problem with the following vector of unknowns,  $v(c)$ , for cluster  $c$ :

$$v(c) = [x_r(c), y_r(c), x_r^2(c) + y_r^2(c) - r_r^2(c)]^T \quad (2.19)$$

where  $x_r(c)$  and  $y_r(c)$  are the  $x$  and  $y$  positions of the center of the best fit circle respectively with radius  $r_r^2(c)$  for cluster  $c$ . We now construct the training matrices  $A$  and  $b$  which are functions defined by the points  $p_j \in S_c$  in cluster  $c$ . Let  $J = |S_c|$  be the total number of points in cluster  $c$  and  $p_j(x)$  and  $p_j(y)$  be indexing functions which indicate the  $j$ -th point's  $x$  and  $y$  coordinates. Thus, the training matrix  $A$  is defined as

$$A = \begin{bmatrix} -2p_1(x) & -2p_1(y) & 1 \\ -2p_2(x) & -2p_2(y) & 1 \\ \vdots & \vdots & \vdots \\ -2p_J(x) & -2p_J(y) & 1 \end{bmatrix}, \quad (2.20)$$

and the training vector  $b$  as

$$b = \begin{bmatrix} -p_1^2(x) - p_1^2(y) \\ -p_2^2(x) - p_2^2(y) \\ \vdots \\ -p_J^2(x) - p_J^2(y) \end{bmatrix}. \quad (2.21)$$

This establishes the over-determined system  $Av(c) = b$ . One way to find a solution is to minimize the following function

$$\min \|Av(c) - b\|_2^2. \quad (2.22)$$

This problem is equivalent to finding the least squares fit with linear parameters  $v(c)$ . The solution of this optimization problem is the pseudo-inverse

$$v(c) = (A^T A)^{-1} A^T b. \quad (2.23)$$

Now, we can extract the best-fit radius feature,  $f_9(c)$ , to be

$$f_9(c) = r_r(c) = \sqrt{(v(c, 1) + v(c, 2) - v(c, 3))} \quad (2.24)$$

where  $v(c, i)$  with  $i \in \{1, 2, 3\}$  indexes the elements of  $v(c)$ .

As previously mentioned, the lidar's scanning pattern varies the point cloud sparsity as distance and angle from the sensor changes (Figure 2.2). To capture the sparsity changes, we extract the features  $x - y$  distance from the sensor and  $x - y$  angle from the sensor. We use the cluster's centroid,  $\bar{p}(c)$ , and ignore its  $z$ -coordinate which projects this point to the  $x - y$  plane.

We define the indexing functions  $\bar{p}(c, x)$  and  $\bar{p}(c, y)$  which extracts the  $x, y$  positions of the cluster's centroid w.r.t the global frame. We also define  $p_s(x)$  and  $p_s(y)$  to be the sensor's  $x$  and  $y$  positions w.r.t the global frame. Thus,

$$\bar{p}(c, x, y) = [\bar{p}(c, x) - p_s(x), \bar{p}(c, y) - p_s(y)]^T, \quad (2.25)$$

is the vector from the sensor to the cluster's centroid. With these definitions, the  $x - y$  distance from the sensor,  $f_{10}(c)$  is defined as

$$f_{10}(c) = \|\bar{p}(c, x, y)\|_2 \quad (2.26)$$

Next, we define  $s_{dir} = [1, 0, 0]^T$  to be the unit vector describing sensor's forward facing direction in the local frame. Intuitively,  $s_{dir}$  states that the sensor is facing forward in the  $x$  direction. Let  $R$  be the rotation matrix from the local sensor frame to the global frame, then  $R \cdot s_{dir}$  is the unit vector describing the forward facing direction of the sensor in the global frame.

Using the dot product relationship,  $a \cdot b = \|a\|_2 \|b\|_2 \cos(\theta)$  for arbitrary vectors  $a$  and  $b$  we can extract the  $x - y$  angle from the sensor feature as

$$f_{11}(c) = \frac{\bar{p}(c, x, y) \cdot (R \cdot s_{dir})}{\|\bar{p}(c, x, y)\|_2 \cdot \|R \cdot s_{dir}\|_2} \quad (2.27)$$

## 2.5 Random Forest Classification Learning

---

### Algorithm 2 Decision Tree Learning

---

- 1: **procedure** DECISIONTREE( $d_t, m_s$ )
  - 2:   Let  $d_t$  be the maximum depth, and  $Q = m_s$  be the the set of  $x = (f, l)$  training example tuples with  $f$  and  $l$  being the feature vector and its label.
  - 3:   Propose a random set of splitting candidates  $\phi = \{(f_i, \tau_i)\}$  where  $i \in \{1, 2, \dots, A\}$ ,  $A = \log_2(|F|)$ ,  $F$  is the set of feature attributes,  $f_i \in F$  and  $\tau_i$  is a threshold.
  - 4:   At each split, partition the set of examples into left and right subsets:
  - 5:          $Q_l = \{x \mid f_i(x) \leq \tau_i\}$
  - 6:          $Q_r = Q \setminus Q_l$
  - 7:   For the current split, compute which  $\phi$  gives the largest gain in information by calculating the entropy
  - 8:          $\phi^* = \operatorname{argmax} G(\phi)$
  - 9:          $G(\phi) = H(T) - H(T|a)$  where  $H(\cdot)$  is the entropy and  $a$  is the current attribute selected
  - 10:          $H(T) = -\sum_{x_i \in Q} P(x_i) \log_2 P(x_i)$ .
  - 11:          $H(T|a) = -(\sum_{x_i \in Q_l} P(x_i) \log_2 P(x_i) + \sum_{x_i \in Q_r} P(x_i) \log_2 P(x_i))$
  - 12:         Define  $P(x_i)$  to be the number of examples in  $Q$  with labels the same as  $x_i$  over the size of  $Q$ .
  - 13:   If  $G(\phi)$  is maximized and the current split is not the maximum depth specified, recurse left and right.
  - 14:   At the terminal leaf, store the size of examples for each label represented.
  - 15:   return a DecisionTree
- 

Having specified the features for the cluster, a binary classifier must

be selected to specify whether a cluster belongs to a candidate human or not. The algorithm proposed does not restrict which classifier to be used. Here, the random forest classifier is utilized for a few reasons. In the seminal work of [55] which detects and tracks humans using the using only the depth data from the Kinect’s RGB-D sensor, multi-class classification was performed with high accuracy and efficiency with the random forest classifier.

The random forest classifier, is an ensemble method consisting of many decision trees. It uses bootstrapping technique both when selecting training examples and when selecting the best-feature per branch split. The basic idea for training a random forest follows. Create  $n_t$  number of decision trees with a maximum depth of  $d_t$ . Let  $m$  be the total number of training examples, where each example is an  $(f, l)$  tuple with  $f$  the feature vector of the example and  $l$  the label of the example. Each decision tree is built with a random subset of  $m/n_t$  training examples with replacement. (ie: Given two decision trees, it’s possible that they share the same training example. This bootstrapping technique is useful for reducing the classification variance). Then, the decision tree is trained using Algorithm 2. At every split in the tree, select the best-split feature from a random subset of all feature attributes. Typically, the size of the subset of feature attributes is of size  $\log_2(|F|)$  where  $F$  is set of feature attributes. Note that the feature selection is bootstrapped, that is, at every split possible, all features are still considered only that a subset is selected when identifying the best-split feature. This training algorithm is summarized in Algorithm 3.

The splitting technique utilized here is unlike regular decision trees. In regular decision trees, the the best-split feature is selected from all available feature attributes. Once an attribute is selected, it is no longer considered in

future splits.

---

**Algorithm 3** Random Forest Learning

---

- 1: **procedure** RANDOMFOREST( $n_t, d_t, m, F$ )
  - 2:   Initialize  $n_t$  decision trees
  - 3:   For each decision tree, train the decision tree with maximum depth  $d_t$  on a random subset of  $m$  examples of size  $m/n_t$ .
  - 4:   Return a Random Forest
- 

Now, given a feature vector  $f$ , the random forest performs classification as follows. Each decision tree outputs a classification with a probability attached to it. For example, if a decision tree has to classify between 3 classes ( $a, b, c$ ), the decision tree may output ( $p(a) = 0.3, p(b) = 0.6, p(c) = 0.1$ ). The random forest, accepts a feature vector  $f$  and passes it to the  $n_t$  decision trees. It then averages the probability result of each decision tree and returns label with the maximum probability (Algorithm 4).

The randomness both at the bootstrapped training example selection and the bootstrapped feature selection, the bias increases slightly. However, due to the number of trees and averaging, the variance also decreases. In the implementation of the human detection algorithm, we use Python’s scikit-learn random forest implementation [41], which follows the training and classification methodology detailed here.

## 2.6 Results and Discussion

### 2.6.1 Learning Curves, Precision/Recall, and Calculation Time

To evaluate the performance of the detection algorithm, Figure 2.8 shows the learning curve of the Random Forest classifier. To construct this graph, we hold out a random subset (typically 10 – 30%) of the training

---

**Algorithm 4** Random Forest Classification

---

- 1: **procedure** RANDOMFORESTCLASSIFY( $RF, f$ )
  - 2:   Let  $RF$  be a trained random forest and  $f$  be a feature vector to be classified.
  - 3:   Get the result from each decision tree,  $DT$  in  $RF$ , by calling DecisionTreeClassify( $DT, f$ ).
  - 4:   Average the probability result from all the decision trees in  $RF$ .
  - 5:   return the label with the maximum probability
  - 6:
  - 7: **procedure** DECISIONTREECLASSIFY( $DT, f$ )
  - 8:   Propagate  $f$  down the decision splits  $\phi$  in  $DT$  until a terminal leaf is reached.
  - 9:   At the terminal leaf, the size of the number of examples representing each label was previously stored.
  - 10:   Use the stored sizes to construct a probability vector  $p_l(f) \in \mathbb{R}^{|L|}$  with  $L$  being the set of all labels possible. The probability of each label is the frequency of the examples with a specific label over all stored sizes.
  - 11:   return  $p_l(f)$
- 

examples. The classifier is trained only on the remaining training examples. While the training error is calculated using the training examples used to train the classifier, the cross-validation error always test the classifier on the entire hold-out examples. Figure 2.8 shows that while the training error rate is always below 1%, the cross-validation plateaus at 10 – 13% after 2000 examples. This implies that more training examples will not necessarily help with making the classifier more discriminative.

Table 2.6.1 summarizes the overall performance of the classifier. For each label, the precision and recall is calculated. Define  $TP$ ,  $FP$ , and  $FN$  as true positives, false positives, and false negatives respectively. Then precision and recall are calculated as

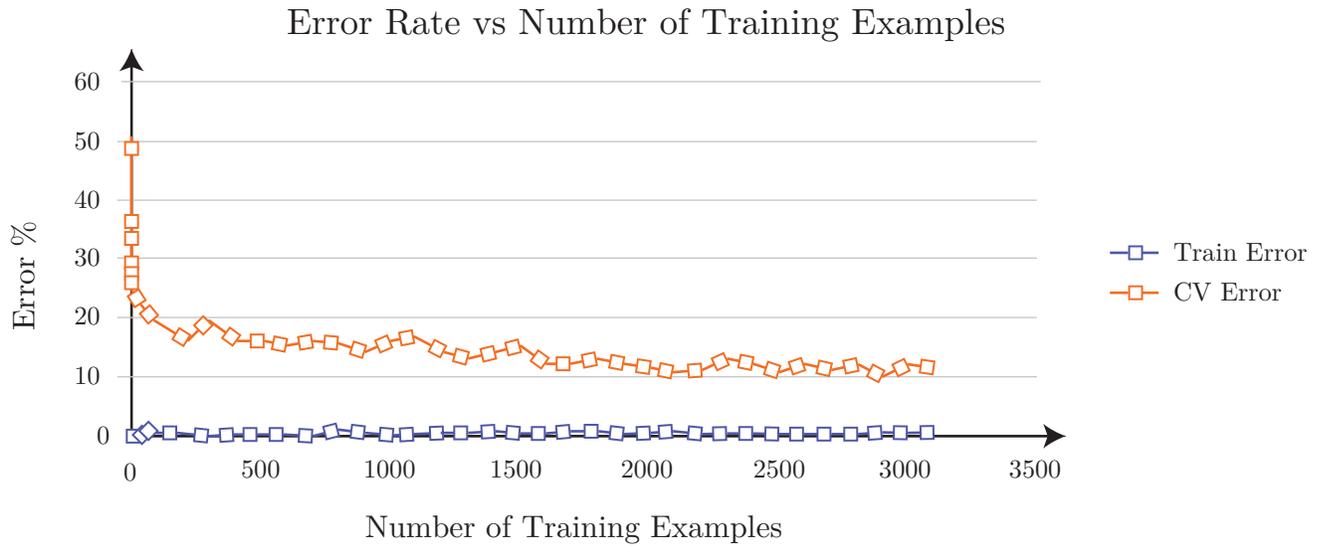


Figure 2.8: Random Forest Classifier: The candidate human cluster classification error rate vs number of training examples. The plot for the training error and cross-validation (CV) error are shown.

$$\text{Precision} = \frac{TP}{(TP + FP)}, \quad (2.28)$$

$$\text{Recall} = \frac{TP}{(TP + FN)}. \quad (2.29)$$

The average labeling accuracy is calculated simply as number of correct labels over total labels. Overall, Table 2.6.1 shows that the classifier has high recall but low precision. This implies that humans are always seen but objects are sometimes seen as humans. While not perfect, this performance is quite desirable. At the very least, the point cloud to process is significantly reduced. Moreover, the classified point cloud certainly contains human points.

	Precision	Recall
Label: Positive(Human)	86.16 $\pm$ 2.89	92.13 $\pm$ 1.07
Label: Negative(Non-Human)	91.58 $\pm$ 0.72	86.22 $\pm$ 3.41
Average Labeling Accuracy	88.43 $\pm$ 1.37	

Table 2.2: Table summarizing the classifier’s precision and recall performance for each label. The average labeling accuracy is the classifier’s performance on a k-fold cross-validation set. The recall performance for classifying a candidate human cluster is bolded to indicate that human clusters are extracted at a high rate even if non-human clusters are classified as humans.

Next, we evaluate the real time capability of the algorithm. When the algorithm was implemented on the real system, the algorithm was split between using C++ and Python. Figure 2.9 breaks down the processing time of the classification process. Note that the most computationally expensive operation, the FPFH feature calculation (since it uses nearest neighbors and histogram construction as part of the calculation) , finish consistently below 0.1ms independent of the number of voxels to process. The performance bottleneck comes from processing the data using Python. In particular, python’s append operation is very slow and scales linearly with the number of voxels to process. Moreover using python for other mundane feature calculation such as finding the area of the bounding box, causes significant slow downs. Thus, in the future, all processing and calculation should be done with C++.

### 2.6.2 Empirical Results

We recall that the detection algorithm is performed in two steps. The first marks whether a cluster is a candidate human or not. If so, the cluster is colored with green points. Next, the candidate human points are clustered together and a bounding box classifier decides whether or not the new clustered

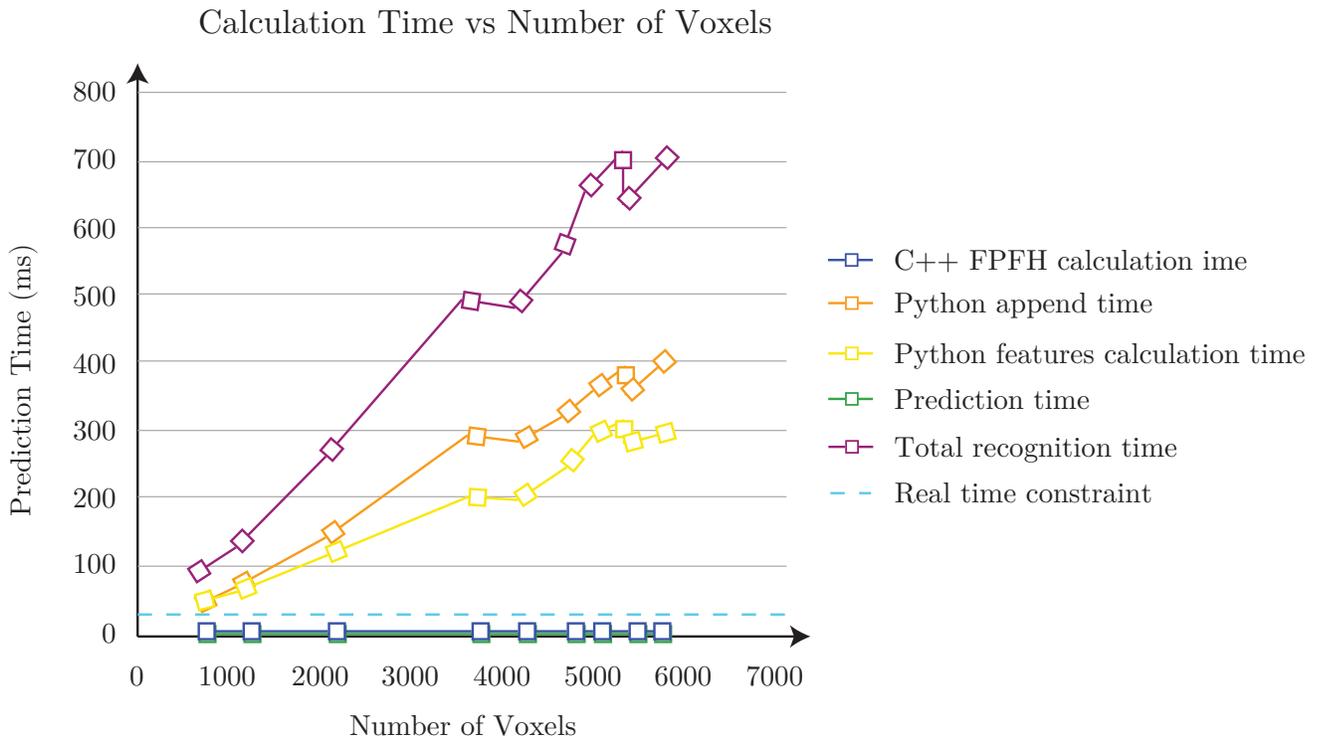


Figure 2.9: Calculation Time vs Number of Voxels

points is human or not. Here, the bounding box classifier simply asks whether the the bounding box formed by the clustered points is big enough to be a human. Thus, the final output of the algorithm are a set of bounding boxes enclosing candidate human points.

Figure 2.10 shows some of the empirical capabilities of the human detection algorithm. In (a), the basic human detection is shown. In (b), the human detection is still possible under lower body occlusions. This is made possible since the Euclidean space has been split into  $m$ -layers. In (c), the human is detected at a close proximity, which addresses the problem with

detecting humans with the stereo camera at close ranges (Figure 2.1 and 2.4). In (d) the algorithm is able to detect humans under varying sparsity, which is an issue with using the MSL’s lidar data (Figure 2.2). In (e), a human in a sitting pose is detected. Finally, in (f), a human (not represented in the training data) in a random pose is detected. Note that both (e) and (f) are possible because the algorithm uses local normal angular differences as the main feature, which enables similar shaped humans to be detected. Finally, the false positivity of the candidate human classifier is evident by the small green points in (a-f). However, the bounding box classifier acts as second filter to identify human points correctly.

## 2.7 Evaluation and Future work

This chapter presented a method for detecting humans using the Multi-sense SL’s Lidar sensor, which can be used to in areas where the stereo RGB-D sensor is not available or reliable. The algorithm provided can detect humans in partial occlusion, in close proximity, under varying sparsity and in random poses. The classification performance of the algorithm has high recall but comparatively low precision. When testing the performance of the algorithm in practice, this quantitative evaluation holds as human points are always extracted and classified as candidate humans, but sometimes objects are also classified as candidate humans. At the very least, the algorithm is capable of processing the entire point cloud input and always returning a reduced number of points with the guarantee that human points are present in the returned point cloud.

Further work remains to ensure that the algorithm is deployable in real systems. First, the implementation presented here has a computational time

bottleneck due to using Python for a subset of the routines. Porting the code to be completely C++ will ensure that the algorithm runs real-time. Second, the algorithm performs detection at every time step without considering temporal data. Third, the algorithm only detect humans and does not perform persistent tracking or skeleton matching. Finally, other human detection approaches using Convolutional Neural Networks are recently more appealing as it is able to automatically extract the necessary features to describe RGB-D data and point cloud data as well as run in real-time with a GPU [16] [44].

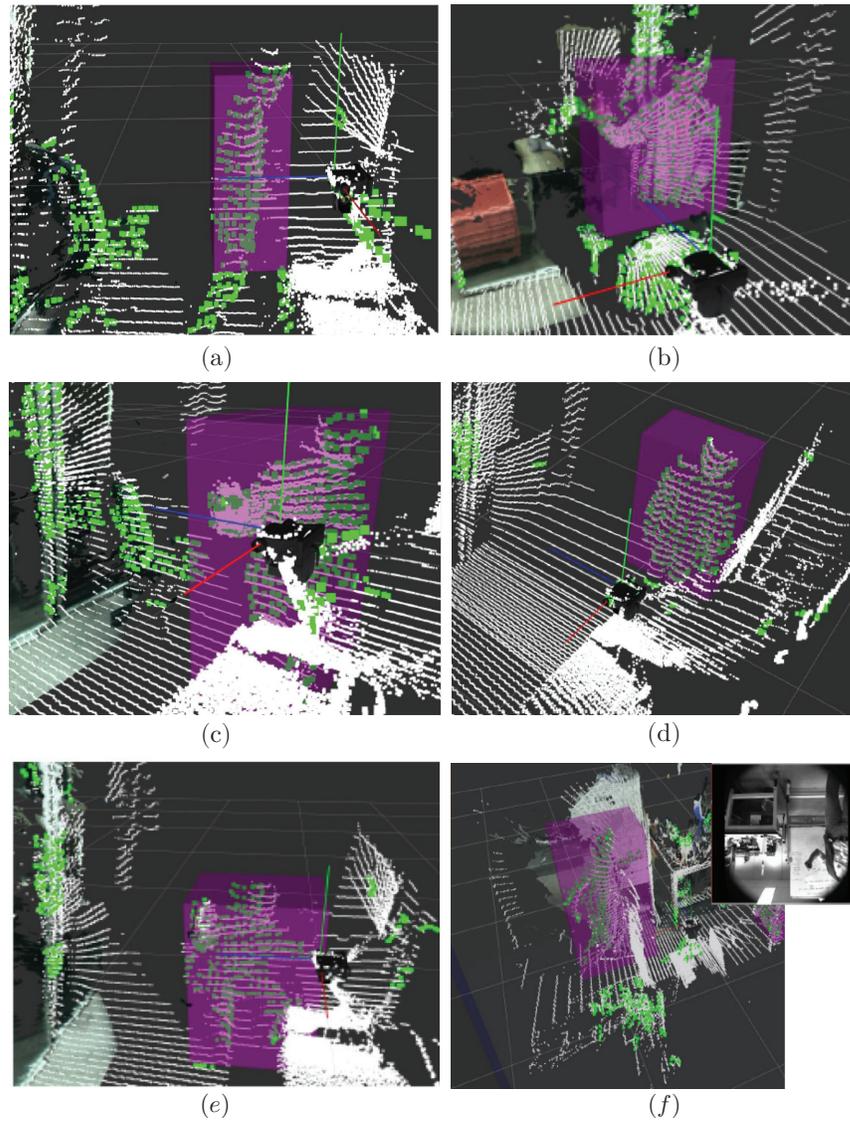


Figure 2.10: Human Detection Empirical Results. Green points indicate candidate human points and the purple bounding box indicates the region where a human is detected. Each sub-figure demonstrates the algorithm’s detection capability. See text for details.

## Chapter 3

# Action Recognition: Gesture Recognition with DMPs and GMMs

Human-aware robots must not only detect humans but also recognize their actions. Action recognition can take on many forms, but in this work, the focus will be placed on recognizing gesture-based actions only. Performing gestures is a key component in non-verbal communication and has been part a subject of interest in psychology [15].

In certain Human-Robot-Interaction(HRI) scenarios, recognizing human gestures is essential for efficient and safe human robot collaboration. Recognizing gestures is a key step to understanding the intent of a collaborative human, especially if there is a mapping between the provided movement gesture and the intent. For example, gestures combined with speech have been shown to enable joint-visual attention and spatial task completion for a human-robot collaborative scenario [8]. In general, the gesture-to-intent mapping is not necessarily a constant one-to-one mapping but can also vary with time and task dependency.

One approach towards action recognition is utilizing some form of feature extraction on some representation of the performed action and using a classifier to distinguish between different actions. This work models static, discrete, and rhythmic types of arm gestures as the forcing function of a Dynamic Movement Primitive (DMP) [23] representing the gesture, where the

basis weights of the forcing function were used as the gesture's features. Using Gaussian Mixture Models (GMMs) as the primary classification tool, different experiments were made to show the practicality of using DMPs for gesture recognition.

The purpose of this work is to identify the limits, practicality and intricacies of using DMPs with GMMs for movement recognition. While not exhaustive, exploring the short list of hypotheses to be tested, which are presented below, gives sufficient insight as the results and discussions sections will show.

The following hypotheses are addressed in this work:

1. An unsupervised learning algorithm such as an Expectation-Maximization (E-M) algorithm on GMMs can be used to automatically segment different static and discrete DMP demonstrations.
2. A supervised Gaussian Mixture Model (GMM) classifier can be used to classify different discrete DMP-based gestures.
3. A GMM classifier can distinguish between spatially different discrete DMP-based gestures
4. The classifier will fail to distinguish between two linear motions.
5. The GMM classifier will fail on classifying rhythmic gestures.
6. Using the discrete DMP formulation to represent all the gestures, the GMM classifier can classify static, discrete gestures but will fail to classify rhythmic gestures.

7. For a given set of data, there is an optimal number of weights that best represents the gestures.

To test the hypotheses, we perform eight types of arm motion gestures. We have one static gesture, five discrete gestures, two of which are linear, and two rhythmic gestures. Figure 3.1 gives a visualization of the gestures. The static gesture is simply constant in space. Two of the discrete gestures are letters U and S, and another two are linear motions with different starting and ending positions. The last discrete gesture is a triangle shape with very similar starting and ending goal positions to test the stability of similar start and end-goal states (see Section 3.2.1). The rhythmic gestures are a continuous circle motion and continuous waving motion.

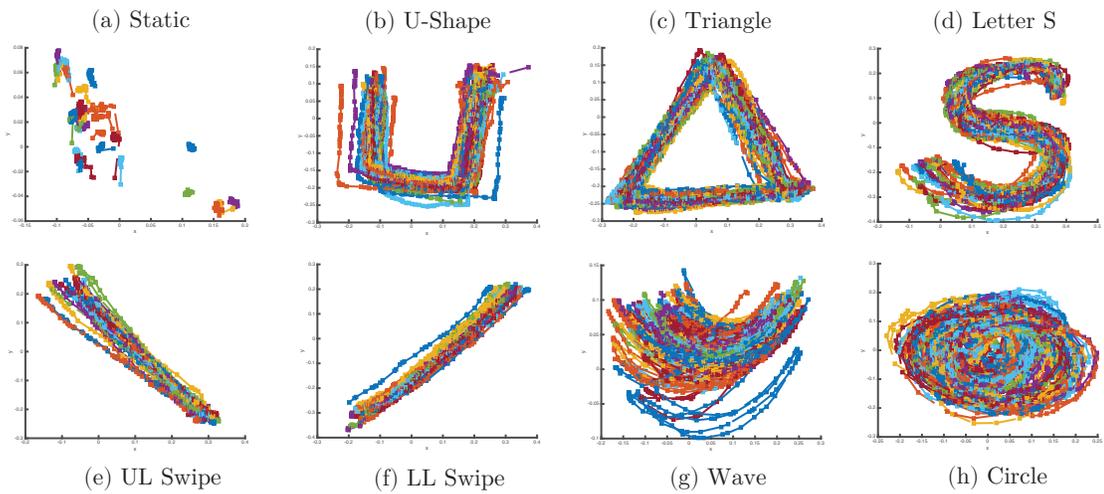


Figure 3.1: The eight types of demonstrated gestures are shown. The sub-figures indicate (a) a static gesture, (b)-(f) five discrete gestures, and (g) and (h) are two rhythmic gestures. The gestures were made using a Kinect that recognized the x-y-z position of the AR marker held by the demonstrator. Each gesture demonstration is plotted as a single color. The static gesture, (a), is a demonstration where the marker never moves. (b) and (d) are discrete letter-type gestures which is used in existing DMP literature to show movement recognition [23]. (c) is a triangle shape gesture to test the ability of the DMP to recognize gestures with almost equal starting and ending positions. (e) and (f) are linear gestures with different starting and ending positions to test if DMPs can discriminate between two spatially different discrete motions. Finally, (g) and (h) represent a continuous circular and waving motion respectively. For each sub-figure, each colored trajectory represents the trajectory of a single demonstration.

From these gestures, it was found that hypothesis 1 is possible, but unreliable, hypotheses 2, 3, 7 are true with high confidence, hypothesis 4 is false with high confidence, and hypotheses 5 and 6 are true with low confidence.

This work presents the following new findings: (a) currently, the community who use movement primitives for recognition do not discuss how their systems are tuned, but here a performance sensitivity analysis is discussed with respect to the number of basis weights used for recognition. (b) Previous recognition studies using DMPs do not try to recognize spatially linear/straight motions as the forcing function may appear similar, but the experiments presented here give evidence that it is possible to discriminate between two straight motions. (c) By accident, it was found that the two rhythmic gestures used in this study can be recognized using the discrete formulation of DMPs with unexpectedly high recognition rates. Finally, (d) DMPs can also represent static-type gestures by setting the goal position constant.

### 3.1 Related Works

The extensive work done in [23] is the most similar to this work. In [23], they detail the mechanics of using DMPs. In their work, they performed motion recognition of discrete movements. In particular, they focused on showing that different alphabetical letters will have a consistent similarity matrix, and so classification is possible. The difference between their work and this work is that GMMs were used to classify static, discrete, and rhythmic gestures using only the discrete formulation of DMPs. Additionally, this work shows that highly linear discrete motions can also be distinguished provided that the DMP parameters are specified properly. This work also shows that it is possible to recognize rhythmic motions despite being modeled with the

discrete formulation of DMPs.

In another work, the authors use a Hidden Markov Model (HMM) to automatically segment sequences of natural activities to automatically segment gestures and cluster them. After the primitive gestures are extracted, the gestures are represented as symbols and, the gestures' lexicon is extracted using their proposed algorithm [59]. Compared to their work, this work focuses on the gestures that are already automatically segmented and only classification of the gestures is needed.

Using nearest neighbors and an SVM classifier were shown to be effective at recognizing military-type gesture recognition [4]. In their work, they focused significantly on recognizing only static type and rhythmic type gestures. To handle noise and feature extraction, their implementation throws many data points away. Their implementation is also sensitive to temporal and spatial type of gestures. Here, the use of DMPs capture the entire motion.

In [2], different unsupervised algorithms were tried to automatically detect gestures and test the performance of various unsupervised clustering methods. However, the features used in their algorithm were not specified, and their features only looked at static and rhythmic motions.

As for human robot collaboration scenarios, [30] uses Probabilistic Movement Primitives (ProMPs) [39] to detect human intentions for assembly hand-over tasks and spatial mimicking of pointing tasks. Probabilistic movements use spatial information as part of learning the movement primitive, and therefore may not recognize similar looking gestures that are spatially different. Thus, ProMPs do not have the spatial invariant property of DMPs.

## 3.2 Technical Background

### 3.2.1 Dynamic Movement Primitives (DMPs) for Gesture Representation

The Dynamic Movement Primitive (DMP) framework [23] is a powerful tool that enables dynamic representation of discrete and rhythmic movements. Here, a biologically-inspired discrete formulation of DMPs given in [40] and [22] is used. As noted in [22], the primary difference is that the differential equations are based on a sequence of convergent acceleration fields instead of force. Practically, this is similar to the original formulation, but with additional benefits such as better stability when the goal and initial positions are similar, invariance under transformations, and better generalization to new movement targets. From this discussion, any one-dimensional movement can be represented as a converging spring-damper system perturbed by a nonlinear forcing function  $f(s)$ :

$$\tau \dot{v}(t) = K(g - x(t)) - Dv(t) - K(g - x_o)s + Kf(s), \quad (3.1)$$

$$\tau \dot{x}(t) = v(t), \quad (3.2)$$

$$\tau \dot{s}(t) = -\alpha s(t), \quad (3.3)$$

where  $x(t)$  and  $v(t)$  are the position and velocity of the movement;  $K$  and  $D$  are the spring and damper terms;  $g$  and  $x_o$  are the goal and start positions of the movement;  $\tau$  is the temporal scaling factor; and  $s$  is the phase variable that exponentially decreases from 1 to 0 with  $\alpha$  to control the convergence time.

While representing motion as a DMP has many favourable properties [23], this work takes advantage of its temporal and spatial invariant property.

In particular similar-looking motions can be demonstrated at varying durations with varying start and end goal positions. For example motion demonstrations can be spatially scaled and performed slowly but still have the same underlying DMP dynamics.

$\tau(s)$	$\alpha$	$K$	$D$
$\tau_{demo}$	$\ln(0.01)$	$400N/cm$	$2\sqrt{K}$

Table 3.1: DMP Learning Parameters

In this work, the parameters of the DMP are summarized in Table 3.2.1 . The spring term is set to be high, whose importance is described in Section 3.5 , and here it was set to  $K = 400N/cm$ . The damping term is critically damped with  $D = 2\sqrt{K}$ . The temporal scaling term is set to  $\tau = \tau_{demo}$ , where  $\tau_{demo}$  is the length of the movement demonstration. Finally,  $\alpha = \ln(0.01)$  to ensure that at  $t = \tau_{demo}$ ,  $s(t)$  is 99% converged.

To obtain the forcing function that represents the gesture, a demonstration trajectory,  $x_{demo}(t)$ , is recorded and differentiated twice to get  $v_{demo}(t)$  and  $\dot{v}_{demo}(t)$ , which is then substituted to the following equation:

$$f_{target}(s) = \frac{\tau\dot{v}(t) + Dv(t)}{K} - (g - x(t)) + (g - x_o)s, \quad (3.4)$$

with  $s(t) = \exp(\frac{\alpha}{\tau_{demo}}t)$  from solving Eq. 3.3 . Note that Eq. 3.4 is obtained by solving for  $f(s)$  from Eq. 3.1. The target function can then be approximated by minimizing the squared error between Eq. 3.4 and the  $w_i$  weights of the following non-linear function:

$$f(s) = \frac{\sum_{i=1}^n w_i \psi_i(s) s}{\sum_{i=1}^n \psi_i(s) s} \quad (3.5)$$

where  $\psi_i(s) = \exp(-h_i(s - c_i)^2)$  is the  $i$ -th Gaussian basis function centered at  $c_i$  with width  $h_i$ . [11] empirically determined that the width and centers of the Gaussian basis functions can be set to  $c_i = 1/n$  and  $h_i = \frac{n}{c_i}$ , where  $n$  is the number of basis weights used to approximate  $f(s)$ . Since all the parameters are fixed, local weighted regression of  $f(s)$  will give consistent  $w_i$  weights.

### 3.2.2 Gaussian Mixture Models (GMMs) for Gesture Recognition

Since DMPs are invariant to spatial and temporal motion demonstrations, it is reasonable to expect that the forcing function weights of the gestures will be clustered together in an  $n$ -dimensional plot [23]. This clustering may be oval in shape and so a multivariate gaussian is used to represent the cluster of the weights:

$$N(\mathbf{w}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{\exp(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{w} - \boldsymbol{\mu}_k))}{(2\pi)^{n/2} |\boldsymbol{\Sigma}_k|^{1/2}} \quad (3.6)$$

where  $n$  is the dimension of the multivariate distribution,  $\mathbf{w} \in \mathbb{R}^n$  is the input,  $\boldsymbol{\mu}_k \in \mathbb{R}^n$  is the mean, and  $\boldsymbol{\Sigma}_k \in \mathbb{R}^{n \times n}$  is the covariance. Now, a GMM [58] is defined to be

$$p(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{k=1}^K \pi_k N(\mathbf{w}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \quad (3.7)$$

where  $p(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$  is the probability of a particular feature,  $\mathbf{w}$ , given all the means,  $\boldsymbol{\mu}$ , and covariances  $\boldsymbol{\Sigma}$  of the combined gaussians. The variable  $\pi_k$  is the mixture component representing the fraction of elements belonging to a mixture  $k$  such that

$$\sum_{k=1}^K \pi_k = 1. \quad (3.8)$$

As an intuition, if there are  $k$  clusters with equal number of elements in each cluster, the mixture component is  $\pi_k = 1/k$ , a uniform distribution.

### 3.3 Experiment Methodology for DMP and GMMs

#### 3.3.1 Gesture Data Gathering

Eight gestures with 30 demonstrations each were recorded using the ROS package *ar track alvar* [34] to track a single AR marker with the Microsoft Kinect. There are five discrete gestures called "*U-shape*, *Letter-S*, *Triangle*, *LL-Swipe*, and *UL-Swipe*," one static gesture called "*Static*," and two rhythmic gestures called "*Wave* and *Circle*." The x-y plots of all the recorded gestures are shown in Figure 3.1.

Each gesture type served a purpose to maximize scientific findings. The *U-shape*, *Letter-S*, and *Triangle* discrete gestures have obvious descriptions. The *U-shape* and *Letter-S* gestures were provided as controls for hypothesis 2 since it has been previously show that they can be recognized with DMPs [23]. However, the *Triangle* gesture was selected since previous gesture recognition never dealt with motions that have almost identical start and goal positions. The *LL-Swipe* and *UL-Swipe* gestures are two discrete, linear-type gestures that starts from the lower-left corner and upper left corner respectively and ends in a corresponding opposite corner. The purpose of the discrete linear gestures is to test hypothesis 4 (that is, the linear DMP motions will be identical to each other and so any classifier will fail to distinguish the two gestures). Finally, two rhythmic gestures were added to test the hypothesis 5 (the discrete DMP formulation will fail recognizing rhythmic gesture). During the demonstration process, both rhythmic gestures *Wave* and *Circle* had no consistent starting and ending position. Sometimes it was difficult to manage the

frequency of the rhythmic gesture, and these inconsistencies are kept as part of training data.

Three additional types of discrete gestures were also gathered, but with only 5 demonstrations each. In particular, a spatially smaller versions of the discrete gestures *U-shape*, *Triangle*, and *Letter-S* were also provided as test data to test hypothesis 3.

These spatially different demonstrations can be viewed at Figure 3.3. Notice that the spatially different demonstrations are only represented in the training data from Figure 3.1 in terms of the overall shape of the gesture. This is important for testing hypothesis 3.

### 3.3.2 Gesture Feature Representation

After all the demonstrations were made, using the DMP formulation with the constants listed in Table 3.2.1, the data was pre-processed to calculate the 3-dimensional x,y,z forcing function of each gesture. We define  $n_b$  to be the number of basis weights on a particular dimension. Then, the  $n_b$  basis weights of each forcing function was extracted using local weighted regression, and the values of the weights were stored as vectors of  $w_x$ ,  $w_y$ ,  $w_z$ , where  $x$ ,  $y$ , and  $z$  indicates the particular Cartesian axis the weight represents. Finally, each gesture is represented as

$$\mathbf{w}_g = [\mathbf{w}_x^T, \mathbf{w}_y^T, \mathbf{w}_z^T]^T, \quad (3.9)$$

where  $\mathbf{w}_g$  is the concatenated vector of the forcing function's basis weights. Thus each gesture,  $\mathbf{w}_g$ , has  $n = 3n_b$  dimension features.

In order to visualize the relationship of the basis weights between any two gestures, the similarity function

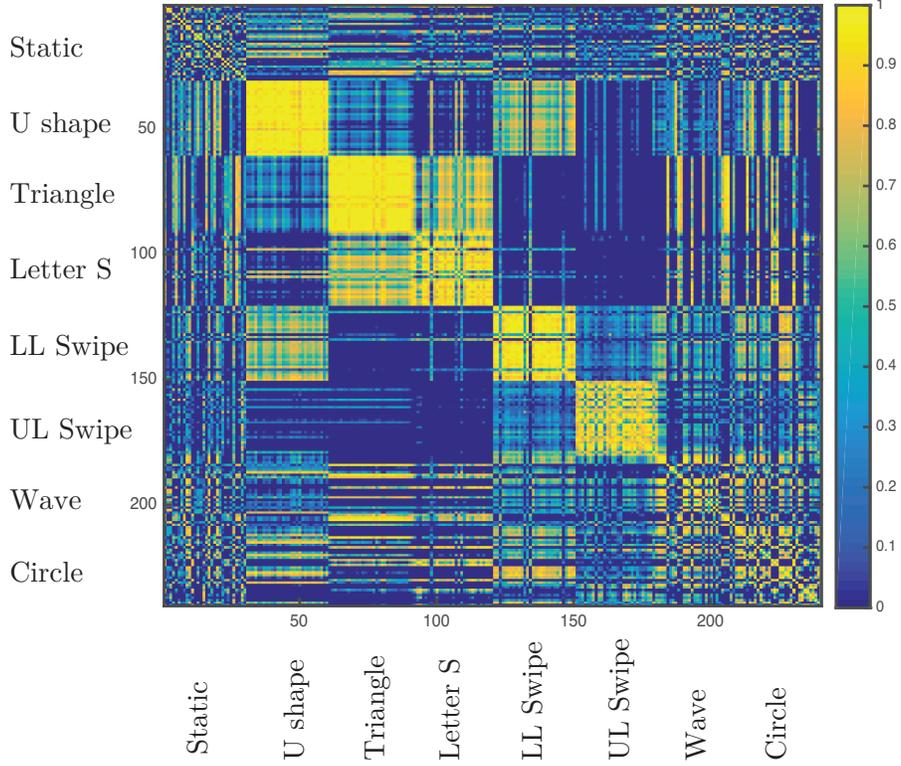


Figure 3.2: The similarity matrix of all the gestures is visualized as a colormap. Each cell represents the similarity between any two gestures where colors closer to 1 indicates high similarity and those below 0 have minimum similarity.

$$similarity = \frac{\mathbf{w}_{g_1}^T \mathbf{w}_{g_2}}{\|\mathbf{w}_{g_1}\| \cdot \|\mathbf{w}_{g_2}\|} \quad (3.10)$$

previously proposed by [23] is used. Note that Eq. 3.10 is 1 when two gestures are 100% similar and is 0 or below when there is minimum similarity. Figure 3.2 is a color map visualization of the similarity matrix between any two gestures, where each cell uses Eq. 3.10 with  $n_b = 5$  basis weights per dimension.

### 3.3.3 GMM Supervised Classification

To perform supervised classification, a finite  $K$  number of gaussian mixtures are trained. Each mixture  $k \in \{1, 2, \dots, K\}$ , representing one gesture, makes  $K$  total number of gestures to consider. For each  $\mathbf{w}_g$  gesture,  $D = 20$  random demonstrations were used as positive training examples. Then, for each  $k$  gaussian mixture, training is done by stacking the mean and covariance of the corresponding training examples:

$$\boldsymbol{\mu}_k = \text{mean}([\mathbf{w}_{g_1}, \dots, \mathbf{w}_{g_2}, \dots, \mathbf{w}_{g_D}]^T) \quad (3.11)$$

$$\boldsymbol{\Sigma}_k = \text{Cov}([\mathbf{w}_{g_1}, \dots, \mathbf{w}_{g_2}, \dots, \mathbf{w}_{g_D}]^T) \quad (3.12)$$

Now, given an unknown gesture,  $\mathbf{w}_g$ , the gesture's membership weight probability,  $r_k$ , is calculated for each  $k$  cluster using Baye's Rule. More specifically,  $r_k$  is the probability that the demonstration belongs to mixture  $k$  given a gesture demonstration,  $\mathbf{w}_g$ :

$$r_k = p(k|\mathbf{w}_g) = \frac{\pi_k N(\mathbf{w}_g, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{\bar{k}=1}^K \pi_{\bar{k}} N(\mathbf{w}_g, \boldsymbol{\mu}_{\bar{k}}, \boldsymbol{\Sigma}_{\bar{k}})}, \quad (3.13)$$

where  $p(k|\mathbf{w}_g)$  represents the cluster membership probability given a  $\mathbf{w}_g$  demonstration, and  $\pi_k$  is the  $k$ -th mixture weight representing that a randomly selected demonstration is part of the  $k$ -th mixture component. Note that  $\sum_{k=1}^K \pi_k = 1$ . Here,  $\pi_k = \frac{D}{D \cdot K} = \frac{1}{K}$  since each mixture component was trained with  $D$  demonstrations and there are  $D \cdot K$  total number of demonstrations. To identify the gesture, the cluster  $k$  that maximizes  $r_k$  is the gesture's classification. This is represented as:

$$k_{best} = \arg \max_k p(k|\mathbf{w}_g), \quad (3.14)$$

### 3.3.4 GMM Unsupervised Classification

For unsupervised classification, Gaussian Mixture Regression using an Expectation-Maximization [58] algorithm is performed on the static and discrete gestures data set and only the number of mixtures,  $K$ , is provided as input. If the mixture regression is 100% successful, it is expected that each mixture component  $\pi_k$  will reflect the true mixture. Note that it is known there are  $m_{per} = 30$  demonstrations for each gesture, and there are  $m = m_{per} \cdot K$  total demonstrations. Thus, it is enough to see if each cluster has identified exactly 30 components. Suppose a cluster has specified  $m_g(k)$  gestures to belong to cluster  $k$ . If  $m_g(k) \leq m_{per}$  then it is assumed that cluster  $k$  has found the correct  $m_g$  gestures. However, if  $m_g(k) > m_{per}$  then cluster  $k$  has  $m_{mistakes}(k) = m_g(k) - m_{per}$  mistakes since perfect clustering should contain  $m_{per}$  gestures for each cluster. Using this intuition, the following performance index is specified:

$$score = \frac{(m - m_{per} - \sum_{k=1}^K m_{mistakes}(k))}{(m - m_{per})}, \quad (3.15)$$

Note that a perfect score of 1 means that each gaussian mixture has exactly 30 gestures and a 0 means that all gestures are classified as a single cluster. It is possible that a score of 1 can be obtained while the clustered gestures are a mix of other gestures. However, in general this is unlikely to happen as different gestures will have different target functions and therefore have different basis weights.

Weights per Dimension	Discrete Gestures	Weights per Dimension	Discrete Gestures
1	(2.0 ± 6.3)%	25	(61.8 ± 13.6)%
3	(14.3 ± 13.8)%	30	(69.0 ± 12.5)%
5	(24.1 ± 15.9)%	35	(58.2 ± 10.6)%
10	(46.3 ± 10.2)%	40	(55.9 ± 7.7)%
15	(47.8 ± 10.5)%	50	(56.4 ± 9.2)%

Table 3.2: Unsupervised GMM

## 3.4 Experiment and Results

### 3.4.1 Unsupervised GMM Performance

The first experiment was to see how well unsupervised classification works on the entire static and discrete gestures data set. The number of basis weights  $n_b$  per dimension was changed as experiments consistently show that performance is sensitive to the number of weights used to represent the gesture. Matlab has a built in gaussian mixture model fitting function, called *fitgmdist*, that utilizes the E-M algorithm. Using the criteria described in Eq. 3.15, the performance of the unsupervised clustering was recorded in Table 3.4.1, where each cell in the table is the mean plus or minus the standard deviation of the score after 10 random trials.

The results indicate that 30 basis weights per dimension has the best unsupervised GMM performance with  $69\% \pm 12.5$  accuracy. However, as more basis weights are added to the dimension, the performance stagnates. Finally, the standard deviation for all the weights tested have high variance indicating unreliability due to its inconsistent performance. Thus, hypothesis 1 has potential but it is not reliable. The unsupervised GMM’s performance sensitivity to  $n_b$  also confirms hypothesis 7.

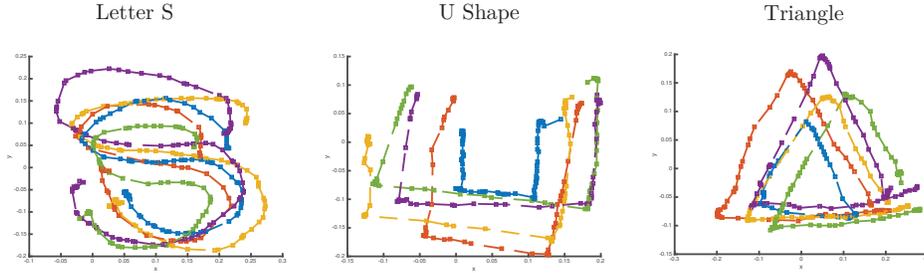


Figure 3.3: Spatially Different Demonstrations

Weights per Dimension	Discrete	Spatial	Rhythmic	Discrete and Rhythmic
1	$(78.7 \pm 0.7)\%$	$(54.7 \pm 4.2)\%$	$(31.5 \pm 8.5)\%$	$(62.8 \pm 1.2)\%$
3	$(98.3 \pm 0.6)\%$	$(73.7 \pm 7.1)\%$	$(93.2 \pm 4.0)\%$	$(96.3 \pm 1.7)\%$
5	$(98.6 \pm 1.2)\%$	$(88.0 \pm 5.3)\%$	$(97.0 \pm 2.2)\%$	$(95.1 \pm 7.1)\%$
10	$(89.3 \pm 1.5)\%$	$(43.3 \pm 5.7)\%$	$(82.7 \pm 2.9)\%$	$(86.1 \pm 1.3)\%$
15	$(71.6 \pm 3.1)\%$	$(11.3 \pm 3.2)\%$	$(58.8 \pm 11)\%$	$(62.7 \pm 2.8)\%$
25	$(78.7 \pm 2.5)\%$	$(33.3 \pm 6.3)\%$	$(76.3 \pm 4.0)\%$	$(77.0 \pm 2.4)\%$

Table 3.3: Supervised GMM on all data sets

### 3.4.2 Supervised GMM Performance

The next experiment was to test hypotheses (2-6) and further confirm hypothesis 7. Tables 3.4.2 and 3.4.2 summarizes the results. For all scenarios, each GMM was trained using 20 random gestures from a corresponding gesture type. Except for the "Spatial" columns, Table 3.4.2 tests the performance against the entire  $K_{test} \cdot 30$  gesture data set where  $K_{test} \in \{K_{discrete}, K_{rhythmic}, K_{spatialdiscrete}, K_{all}\}$  is the number of gestures being con-

Weights per Dimension	Discrete	Spatial	Rhythmic	Discrete and Rhythmic
1	$(77.2 \pm 3.7)\%$	$(51.3 \pm 3.2)\%$	$(36.0 \pm 12.4)\%$	$(60.9 \pm 2.5)\%$
3	$(97.3 \pm 1.4)\%$	$(66.7 \pm 7.1)\%$	$(81.1 \pm 9.1)\%$	$(88.6 \pm 3.6)\%$
5	$(96.2 \pm 2.5)\%$	$(65.3 \pm 8.2)\%$	$(88.5 \pm 8.8)\%$	$(92.5 \pm 2.6)\%$
10	$(86.5 \pm 1.1)\%$	$(40.7 \pm 4.9)\%$	$(68.0 \pm 8.5)\%$	$(73.9 \pm 8.6)\%$
15	$(57.2 \pm 4.3)\%$	$(10.0 \pm 3.5)\%$	$(61.2 \pm 7.5)\%$	$(45.5 \pm 6.9)\%$
25	$(50.17 \pm 9.2)\%$	$(26.0 \pm 4.9)\%$	$(77.1 \pm 4.0)\%$	$(36.5 \pm 5.7)\%$

Table 3.4: Supervised GMM on Cross Validation data set

sidered.

In this work, there are  $K_{rhythmic} = 2$  rhythmic gesture types,  $K_{discrete} = 5$  discrete and static gesture types,  $K_{spatialdiscrete} = 3$  spatially different discrete gestures and  $K_{all} = K_{rhythmic} + K_{discrete}$  discrete and rhythmic gesture types.

Recall that for each gesture,  $D = 20$  training data were used to train each mixture model. To ensure that the performance is not skewed by the trained data, Table 3.4.2 tests the performance only on the remaining unseen  $K_{test} \cdot 10$  gesture data set.

In the '*Discrete*' column, the supervised GMM was trained and tested only on the  $K_{discrete}$  static and discrete gestures. The '*Spatial*' column was also trained using the  $K_{discrete} \cdot 30$  static and discrete gestures but was tested using the  $K_{spatialdiscrete}$  spatially different discrete gesture set. The '*Rhythmic*' column was trained and compared only on the  $K_{rhythmic}$  rhythmic gestures. Finally, the '*Discrete and Rhythmic*' column was trained and tested on the  $K_{all}$  static, discrete, and rhythmic gestures without the spatially different gestures. For all types of tests, the number of basis weights per dimension were also changed to test hypothesis 7.

Tables 3.4.2 and 3.4.2 show that in general, there is high accuracy in the recognition performance of static and discrete gestures, which confirms hypothesis 2 and disproves hypothesis 4. In general, recognizing spatially similar static and discrete gestures performs very well, and the accuracy drops below 80% only when more basis weights per dimension are used due to over fitting.

The *Spatial* column confirms hypothesis 3. Concretely, spatially different discrete gestures can be recognized with basis weights of 3 and 5 per dimension.

sion. As a reminder, the training set for the *Spatial* has never seen spatially smaller demonstrations, which makes this result more meaningful and significant.

What is surprising is that the *Rhythmic* column shows that even with using the discrete formulation of DMPs to represent rhythmic gestures, the supervised GMM can distinguish between the rhythmic "Wave" and "Circle" gestures. It was expected that the rhythmic gestures would appear as noise and the GMM will fail to recognize the rhythmic gestures completely. However, as the result shows, the accuracy is better than guessing between two rhythmic gestures at random.

To test if the GMM classifier can discriminate between static, discrete, and rhythmic gestures, the *Discrete and Rhythmic* columns shows that the presence of rhythmic gestures did not affect recognition performance as it reflects similar values to the *Discrete* column. From this study, it is surprising that hypotheses 5 and 6 are both false as rhythmic gestures were classified successfully.

For all of the gesture recognition tests, it is evident that the number of weights used to represent the gesture affected the performance of the classifier, which confirms hypothesis 7 convincingly. Using too many basis weights causes over-fitting with high variance error, and not using enough basis weights (eg: when basis weights per dimension = 1) causes under-fitting with higher bias errors.

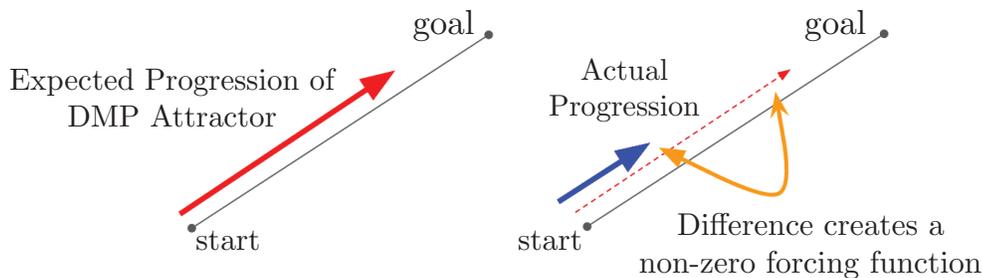


Figure 3.4: Linear Discrete Motion Gestures can be differentiated when  $K$  is high such that the DMP's attractor dynamics move faster than the actual demonstration making the forcing function non-zero.

### 3.5 Discussion

The results with regards to the ability of a supervised GMM to classify rhythmic gesture is strange and very unexpected. There are many possible explanations and some of are discussed here. It's possible that since there are only 2 rhythmic gestures, classifying between the two is easy as the GMM always return the best guess. The weights of each rhythmic gesture could also be sufficiently different in terms of forcing function noise, so fitting a GMM on two noise distributions was sufficient to discriminate between the two rhythmic gestures.

The hope was to show that rhythmic gestures will completely fail and using the rhythmic formulation of DMPs will be necessary. However, to even use the rhythmic DMP formulation for proper comparison, more rhythmic gesture types need to be recorded. Still, with the gestures used in this study, the static, discrete, and rhythmic gestures were classified successfully. Thus, until further study is conducted, hypotheses 5 and 6 are false but with low confidence.

The second surprising finding is that while the static and discrete ges-

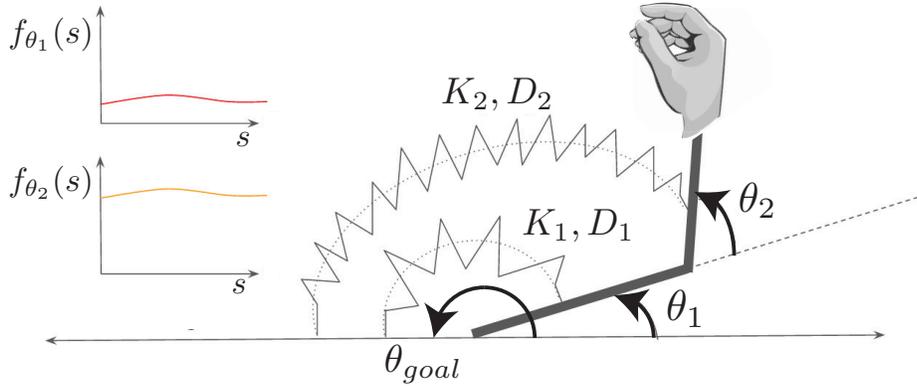


Figure 3.5: Recognizing static gestures is possible by setting the goal position away from the user and using features such as arm angle relative to the body of the user.

tures were classified successfully, confirming hypotheses 2 and 3, it did so while also classifying two different types of discrete linear gestures. The traditional thinking is that discrete linear gestures will have a 0 forcing function. This is why in [23] the motion gestures performed were all letters as trying to different linear motions could be problematic. However, here the results show that recognizing between two linear discrete gestures is possible. An intuitive explanation is provided in Figure 3.4. That is, if  $K$  of the DMP is set to be very high such that the attractor dynamics moves faster than the demonstration, the forcing function is non-zero and any type of linear motion in x-y-z can be classified.

In fact, this finding is predicted much earlier by looking at the similarity matrix between the two linear gestures in Figure 3.2. It is evident that they have no similarity at all.

This finding has an additional consequence. That is, it is also possible

to detect richer types of static gestures. For example, suppose that recognizing between two types of static arm gestures is necessary. The coordinates can be set to the angle formed by the upper arm to the shoulder and the angle formed by the elbow to the upper arm as shown in [4]. Then, for all static gestures, the goal position can be set away from the user as indicated in Figure 3.5. However, the additional complication is that the goal position is now different. Thus, to make this work with the framework, a higher level classifier is needed to distinguish between static and discrete gestures.

In this work the recognition of static, discrete, and rhythmic gestures were performed by using the discrete formulation of DMPs. In particular, the forcing function of the DMP was used to represent the gesture in which the weights obtained from local-weighted regression of equally-spaced gaussian functions were the features.

Using only GMMs for classification, it was found that unsupervised clustering can potentially be used to automatically learn different gesture types. However the high variability of the unsupervised GMM in the results shows that it will be unreliable.

On the other hand, using supervised GMM clustering provided an easy way to train a classifier while performing reliable recognition at a high accuracy especially when the number of basis weights are tuned. In particular, the classifier was able to distinguish between discrete and static gestures. Additionally, the classifier was also able to recognize different types of discrete linear motion under the DMP framework. This is an unexpected result as the DMPs of the two linear motions were expected to be different.

Finally, another unexpected result shows that the GMM can also classify rhythmic gestures even though the gestures were represented as discrete

motions. However, there are not enough rhythmic gestures in this data set to truly claim that the discrete DMP formulation can classify all types of rhythmic gestures.

Overall, this work demonstrates that using the new discrete formulation of DMPs is an effective method for recognizing spatially and temporally invariant movement gestures. Once the gestures are recognized, a mapping between the gesture to intention may be formulated.

### **3.6 Future Work**

In this work, only one static gesture was tested. Still, experiments with the discrete linear gestures resulted into a finding that DMPs can also represent richer static gesture types, but experimental validation remains. As a potential approach, identifying static gestures can be recognized with the current framework. Since it is static, the forcing function will be close to 0 as the goal and start positions are very close. Then after recognizing that the gesture is a static type, another GMM that classifies different type of static gestures can be used with the goal position explicitly specified.

Another future work is on the topic of rhythmic gestures. It is still not convincing that the discrete formulation of DMPs is enough to classify rhythmic gestures. In the future, two better ways of recognizing rhythmic gestures exist. The first is to use the rhythmic formulation for DMPs and use the learned basis weights for classification. Second, performing alignment on the data and approximating one period of the demonstration using a fourier transform can give consistent basis function weights.

Another problem with the current classification scheme is that it can-

not handle incorrect gestures as the current framework only assumes that all gesture demonstrations is represented by the GMM. Thus the classifier always returns the best maximum guess for any given gesture. This can be fixed by doing some threshold study after the best cluster membership is selected.

Finally, while using DMPs is invariant to different temporal demonstrations of similar gestures, the classifier will not be able to identify when the desired gesture has begun or ended. Thus, this will fail when a time series of data is given without some heuristics given to the system. An example heuristic for example could be detecting minimum velocity onset for both start and ending conditions [23]. However, this has the disadvantage that no gesture is ever given when the velocity is less than the specified threshold and gestures are assumed to be always given when the velocity is greater than the threshold. Perhaps a better approach to handle continuous time series is to use a change-point-detection algorithm [35].

## Chapter 4

# Action Generation: Decision Making with Model Predictive Control

The work presented in this chapter is part of a previous publication<sup>1</sup>. The author of this thesis was completely responsible for the content of the first half of the publication (Abstract to section 4.2 and section 6.1), which is presented here. While the author of this thesis made minor contributions to the theoretical development of Section 5 and 6.2, the second author, Orion Campbell, lead the development and writing of these sections. The third author, Travis Llado was responsible for the hardware implementation. The fourth author, Donghyun Kim provided important code for visualization. The fifth author was helpful during the initial development of section 5. Finally, Dr. Luis Sentis was the P.I of the paper. Now we discuss the contributions, methods and results of the first half of the publication.

A robot that can detect and track humans as well as understand human actions may be human-aware in the sense that it can identify where humans are and make statements about their actions. However, human detection and action recognition on their own are not necessarily useful. To be useful, a robot must generate actions and behaviors on its environment that are human-

---

<sup>1</sup>Jorgensen, S. J., Campbell, O., Llado, T., Kim, D., Ahn, J., and Sentis, L. (2017). Exploring Model Predictive Control to Generate Optimal Control Policies for HRI Dynamical Systems. arXiv preprint arXiv:1701.03839.

aware. Human-aware actions are important when the robot is under safety constraints. Furthermore, the robot’s performance is directly tied to its actions and behavior.

Since generating human-aware robot behaviors is important, how can a robot make such decisions that, for example, balance between safety and performance requirements? This decision making process is a fundamental problem in robotics [53]. Here we will discuss one approach towards human-aware action generation. The approach described here breaks the problem of generating human-aware actions into two components.

First, we model Human-Robot-Interaction (HRI) scenarios as linear dynamical systems which will describe the dynamic interplay between robot actions, human actions, and the environment. Second, we use Model Predictive Control (MPC) with mixed integer constraints to generate human-aware control policies over the dynamical system representation. The approach is motivated by presenting an assistive robot that aims to maximize productivity while minimizing the human’s workload, and the simulation results show that the robot generates useful behaviors as it finds control policies to minimize the specified cost function.

This work is inspired from studying Social Cognitive Theory (SCT) [3] which claims that human behavior is based on the dynamic interplay of personal, environmental, and behavioral influences. It was recently used to model the walking exercise behavior of humans as a linear dynamical system [31]. Among many other states that interplay with each other, their model included a measure of self-efficacy which increases as a result of exercise, thereby increasing the exercising behavior further. Subsequently, the authors also showed that a policy for behavior intervention can be generated

using Model Predictive Control [32].

In the same way, we ask similar questions: Can HRI scenarios be modeled as dynamical systems? If so, can the tools of control theory generate useful policies? Previous work on modeling HRI scenarios and generating appropriate behaviors include creating belief models of the robot and human [5, 6], probabilistic anticipatory action selection [21], collaborative agent planning [54], motion planning for navigation to maximize human comfort [56], fluent-turn taking using timed petri-nets [10], utilizing POMDPs for modeling cognition of an autonomous service robot [52], and many others. For all scenarios the robot’s cognitive model of the world and the human was necessary to generate appropriate actions to address the task at hand.

Here, we frame the cognitive modeling problem based on intuitive mechanical analogies. This technique leverages the power of feedback optimal control to generate useful interactive behaviors. In particular, this work explores how Model Predictive Control (MPC) with mixed-integer constraints can be used to solve HRI scenarios modeled with linear dynamics. In doing so, this methodology is an approach towards creating *cognitive feedback controllers*. The modeling and policy generation technique here is high level. Therefore, it is assumed that the low level control of the robot, such as its joint controllers or torque controllers [37], is given. Other low-level modules needed to accomplish the task such as perception modules to process the environment, and motion primitives to accomplish sub-tasks are also assumed to be given.

As the name suggests, MPC contains a model of the system and can simulate how its control policies can affect the model in the future [46]. Since MPC can “see” a finite horizon into the future using its model of the world, it can identify locally what the best control policies are to minimize some

objective function.

To describe the usefulness of MPC for extracting useful control policies in HRI, we consider an assistive robot that helps a human accomplish his work by bringing the human the necessary deliverables from an inventory station (Fig. 4.1). The assistive robot must (1a) ensure it has enough battery to remain operational, (1b) continue being productive by delivering work to the human, and (1c) ensure that the human is never overworked.

Note that the HRI scenario contain a number of if-then statements which activate or deactivate boolean variables to specify if certain control inputs are available for the robot. For instance, the robot can only move *if* it has enough battery power or it can only pick up deliverables when it is near the inventory station or the human’s workstation. Such if-then constraints must be incorporated in the optimization routine mathematically via reformulating the statements as inequality constraints. To incorporate these constraints, we use Mixed Integer MPC, which is an optimization routine that minimizes some objective function subject to both real-valued and integer constraints. This optimization framework is called Mixed-Integer Programming (MIP) [57].

Interestingly, while our objective function only contains battery levels, productivity, and human workload the robot is able to find the correct control policies to satisfy the end objective.

To demonstrate the approach, simulations were performed using a Python interface called CVXPY [12], a convex optimization library with mixed-integer programming capability with an academic license of the Gurobi [17] solver. The code is available at [https://github.com/hrianon/mpc\\_hri](https://github.com/hrianon/mpc_hri).

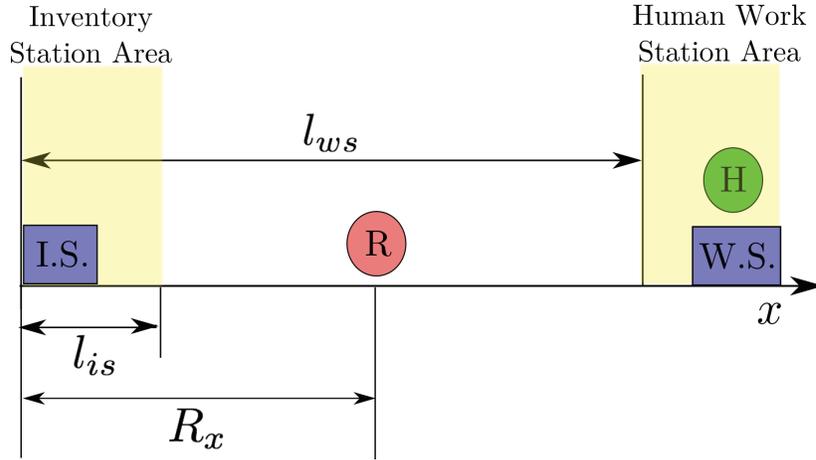


Figure 4.1: Assistive HRI Scenario: An assistive robot must bring deliverables from the inventory station (I.S.) to the human’s work station area (W.S.). A mindful robot will ensure that the human is never overworked.

#### 4.1 Related Works

MIP was previously used for planning spacecraft trajectories [47] and using integer constraints to model obstacle avoidance. MIP has also been used for generating optimal paths for manipulators [13]. Martin et al. [31] used a fluid-tank analogy and a corresponding linear dynamic model to characterize human mental states that influence daily walking behavior. With a simplified version of the model, they controlled the system using Hybrid Model Predictive Control with integer and boolean constraints to achieve a desired goal [32].

In addition to previously mentioned works on modeling HRI and generating behaviors [5, 6, 21, 54, 56, 10, 52], kinodynamic planning with RRT can also be used to solve search problems with dynamic constraints [28]. However, as with most planning algorithms, this requires specifying a desired goal state that may not be reachable. On the other hand, an MPC formulation

performs an optimization routine to find the best control policy to minimize an objective function over the given time horizon.

## 4.2 HRI as Linear Dynamical Systems

To take advantage of the techniques found in the controls community, we model HRI scenarios as linear dynamical systems which have the form

$$\frac{dx}{dt} = \dot{x} = Ax + Bu, \quad (4.1)$$

where  $A \in \mathbb{R}^{n \times n}$  and  $B \in \mathbb{R}^{n \times m}$  describe the state changes due to the  $n$  world states,  $x \in \mathbb{R}^n$ , and  $m$  control inputs,  $u \in \mathbb{R}^m$  respectively. In this work, we discretize the dynamics by  $\Delta t$ . Note that by Euler integration, an estimate of the state vector after  $\Delta t$  can be obtained using Equation 4.1 as

$$x(k+1) = \dot{x}(k)\Delta t + x(k), \quad (4.2)$$

where  $x(k)$  denote the state at time step  $k$ . Next, by expanding  $\dot{x}(k)$ , the state evolution can be described by known variables

$$x(k+1) = (Ax(k) + Bu(k))\Delta t + x(k), \quad (4.3)$$

$$= (A\Delta t + I)x(k) + Bu(k)\Delta t. \quad (4.4)$$

where  $I \in \mathbb{R}^{n \times n}$  is identity and  $u(k)$  denote the input at time step  $k$ .

## 4.3 Policy Generation via Model Predictive Control

Given some desired robot behavior  $y^{ref} \in \mathbb{R}^{n_y}$  with  $n_y$  behaviors, e.g. we want the robot to be 100% productive and minimize the human workload to 25%, a standard quadratic cost function is used to quantify how well the

decision vectors,  $u_0, u_1, \dots, u_{p-1}$ , bring a state output  $y$  to  $y_{ref}$  over a finite horizon  $p$  time steps. The cost function is then defined as

$$J = \sum_{i=1}^p (y(k+i) - y^{ref})^T Q_y (y(k+i) - y^{ref}), \quad (4.5)$$

where  $Q_y \in \mathbb{R}^{n_y \times n_y}$  is a matrix describing the quadratic weights of the desired behavior.

The mixed-integer MPC problem can now be formulated as follows. In general, the MPC problem attempts to minimize a cost function  $J$  subject to dynamic constraints and inequality constraints:

$$\begin{aligned} & \underset{\{[u(k+i)]_{i=0}^{p-1}, [\delta(k+i)]_{i=0}^{p-1}\}}{\text{argmin}} && J && (4.6) \\ & \text{s.t.} && && \\ & x(k+i) = (A\Delta t + I)x(k) + Bu(k)\Delta t, && && \\ & y(k+i) = Cx(k+i), && && \\ & E_1\delta(k) \leq E_2 + E_3x(k) + E_4u(k) + E_5z(k), && && \end{aligned}$$

where  $\delta(k) \in \{0, 1\}^{n_{bool}}$  are  $n_{bool}$  boolean variables used in the problem,  $z(k) \in \mathbb{R}^{n_o}$  are  $n_o$  floating variables, and  $E_{\{1, \dots, 5\}}$ , are matrices used to compactly specify the constraints of the problem. However, to be very clear about how constraints are specified in the MPC problem, we will describe each inequality constraint used for the scenario.

## 4.4 Assistant Robot Scenario as a Linear Dynamical System

### 4.4.1 World State and Actions

We use a fluid-flow analogy to describe the linear dynamics of the scenario (Fig. 4.2) which enables easy visualization of how the different states of the world are affected by the robot's actions and other world states. We model the state of the world as a vector  $x_R \in \mathbb{R}^5$ ,

$$x_R = [R_x, R_b, R_d, R_p, H_l]^T, \quad (4.7)$$

where  $R_x$ ,  $R_b$ ,  $R_d$ ,  $R_p$ , and  $H_l$ , are the robot's x-coordinate position, battery levels, amount of deliverables being carried, self-perceived productivity, and perceived human workload respectively. In general, the system has control inputs  $u \in \mathbb{R}^6$  defined as

$$u_R = [u_{move}, u_{charge}, u_{ipu}, u_{ido}, u_{wpu}, u_{wdo}]^T. \quad (4.8)$$

More specifically,  $u_{move}$  lets the robot move left and right,  $u_{charge}$  specifies how the robot's batteries change,  $u_{ipu}$  and  $u_{wpu}$  denote the act of picking up deliverables from the inventory station and the human's work station respectively, and  $u_{ido}$  and  $u_{wdo}$  denote the act of dropping off deliverables to the inventory station and the human's work station respectively.

### 4.4.2 State Transition Matrix

Referring to Fig. (4.2), at every time step, the robot's battery decreases by  $-1/\tau_b$  and is further decreased as it moves ( $|u_{move}| > 0$ ). The battery can be charged by  $u_{charge}$  when the robot is near the inventory station.

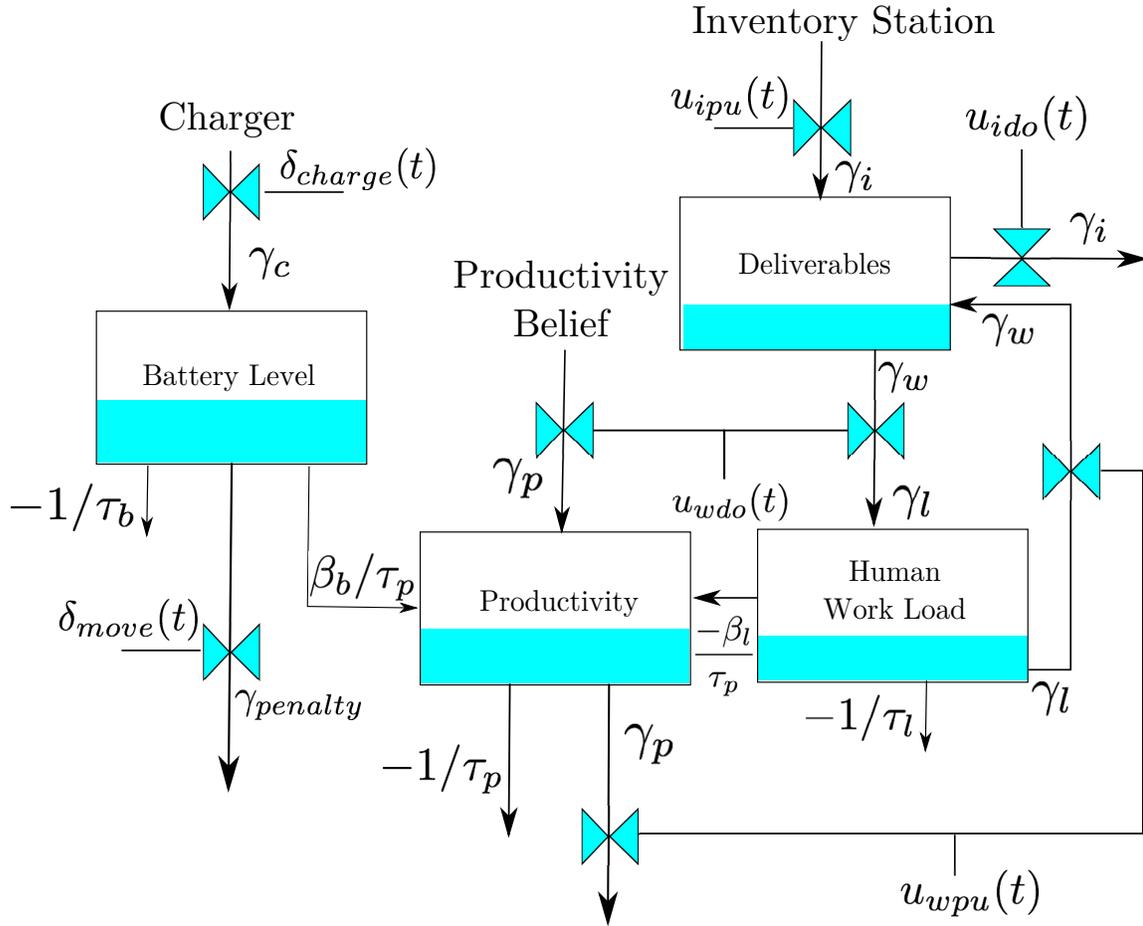


Figure 4.2: Fluid Analogy for the Assistive Scenario.

The robot feels productive whenever it has high battery levels,  $\beta_B/\tau_p$ , and whenever it drops off deliverables,  $u_{wdo}$  to the human, but feels less productive whenever it takes work,  $u_{wpo}$ , away from the human and whenever the robot perceives high levels of human workload,  $-\beta_l/\tau_p$ . As with battery levels, the robot's perception of its own productivity decreases by  $-1/\tau_p$  with time.

The robot's capacity to carry deliverables is modeled by the state  $R_d$ . Whenever the robot performs a deliverable pick up action ( $u_{ipu}$  and  $u_{wpu}$ ),

this state increases. Similarly, when the robot performs a deliverable drop off action ( $u_{ido}$  and  $u_{wdo}$ ) this state decreases.

Finally, the robot has a model of the human's workload,  $H_l$ . Namely, deliverables dropped off to the human's work station is analogous to increasing the human workload. The human, working at his own pace, reduces his workload by  $-1/\tau_l$ .

We note that Fig. (4.2) contains gamma,  $\gamma$ , variables. These variables are constants to describe how much the state changes,  $\frac{dx}{dt}$ , as a function of time due to the input  $u$ . These variables are useful since the units for  $u_{move}$  is different from  $R_b$ , but battery levels are affected by moving nonetheless. They also capture how the robot's deliverables  $R_d$  gets converted to human workload,  $H_l$ , and robot productivity  $R_p$  through an action  $u_{wdo}$ . However while  $\gamma$  might be different, the exchange between two states uses the same constant. Thus, if a robot drops off deliverables to the human's work station, which increases  $H_l$  and  $R_p$ , and decreases  $R_d$ , the robot can pick up the same amount of deliverables, decreasing  $H_l$  and  $R_p$  and increasing  $R_d$  back to its original values. This abstraction enables the model to evolve later if more states are added to model human workload or robot productivity.

Thus, the world state evolves with the following  $A$  and  $B$  matrices

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1/\tau_b & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \beta_b/\tau_p & -\beta_w/\tau_p & -1/\tau_p & 0 \\ 0 & 0 & 0 & 0 & -1/\tau_l \end{bmatrix}, \quad (4.9)$$

$$B = \begin{bmatrix} \gamma_m & 0 & 0 & 0 & 0 & 0 \\ 0 & \gamma_c & 0 & 0 & 0 & 0 \\ 0 & 0 & \gamma_i & -\gamma_i & -\gamma_w & -\gamma_w \\ 0 & 0 & 0 & 0 & -\gamma_p & \gamma_p \\ 0 & 0 & 0 & 0 & -\gamma_l & \gamma_l \end{bmatrix}, \quad (4.10)$$

where the  $\gamma$  variables are constants that describe how the state changes due to the input  $u$ .

The current state transition matrix that describes the HRI scenario does not incorporate certain constraints of the problem. For example, the robot may only pick up deliverables whenever it is near the inventory station or when it is near the human work station. The next section describes how such constraints are automatically incorporated in the MPC problem statement.

## 4.5 World Constraints formulated as Mixed-Integer Constraints for Model Predictive Control

To maximize battery and productivity, and minimize human workload, we define the observation vector  $y \in \mathbb{R}^3$  at time step  $k + 1$  to be  $y(k + 1) = Cx(k + 1)$

where  $C$  is

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad (4.11)$$

We set  $y^{ref} = [1, 1, 0.25]^T$  which tells the robot to aim for 100% battery level, 100% productivity, and a human workload of 25%. The cost function weights are set to be  $Q_y = \text{diag}\{w_b, w_p, w_l\}$  with  $w_b$ ,  $w_p$ , and  $w_l$  denote the weights for the battery, robot productivity, and human workload respectively.

### 4.5.1 Scenario Constraints

The HRI scenario presented contains a number of constraints and the optimization routine must be restricted to a set of allowable actions depending on the world states. The robot has the following constraints:

1. It can move only if it has enough battery.
2. Its batteries are charged only if it is near the inventory station.
3. It can pick up and drop off deliverables only if it is near either station.
4. It cannot pick up deliverables beyond its capacity.
5. It cannot drop off deliverables if it has no deliverables.
6. It loses more battery as it moves.

Such if-then constraints must be converted to inequality constraints in order to frame the problem as a mixed-integer MPC problem. To specify constraint 1, we introduce a boolean variable,  $\delta_{bat}$  that indicates if the robot has enough battery to move. Namely, it is only true if and only if the robot's battery is above a threshold,  $b_{thresh}$ .

$$\delta_{bat} = 1 \Leftrightarrow R_b(k) \geq b_{thresh} \quad (4.12)$$

The following inequalities express this if-then constraint from Eq. 4.12 as a mixed-integer constraint.

$$R_b(k) - b_{thresh} \leq \delta_{bat}(k)(R_b^{max} - R_b^{min}), \quad (4.13)$$

$$R_b(k) - b_{thresh} \geq (1 - \delta_{bat}(k))(R_b^{min} - R_b^{max}), \quad (4.14)$$

where  $R_b^{max}$  and  $R_b^{min}$  are upper and lower bounds of the battery level. Next, we specify that the robot can only move if it has enough battery

$$u_{move}(k) \geq \delta_{bat}(k)u_{move}^{min}, \quad (4.15)$$

$$u_{move}(k) \leq \delta_{bat}(k)u_{move}^{max}, \quad (4.16)$$

where  $u_{move}^{min}$  and  $u_{move}^{max}$  specify the maximum movement effort the robot can use at every time step.

Next, to specify constraints 2, and 3, we introduce two booleans,  $\delta_{is}$  and  $\delta_{ws}$  to indicate whether the robot is at the inventory station or the human work station. The desired location constraint is described as

$$\delta_{is} = 1 \Leftrightarrow R_x \leq l_{is} \quad (4.17)$$

$$\delta_{ws} = 1 \Leftrightarrow R_x \geq l_{ws} \quad (4.18)$$

where  $l_{is}$  specifies the location of the inventory station and  $l_{ws}$  specifies the location of the human's work station. Similar to the battery level constraint, the location constraints can be expressed as a mixed-integer constraint using the following inequality constraints.

$$(R_x(k) - l_{is}) \leq (1 - \delta_{is}(k))R_x^{max}, \quad (4.19)$$

$$(R_x(k) - l_{is}) \geq \delta_{is}(k)(-R_x^{max}), \quad (4.20)$$

$$(R_x(k) - l_{ws}) \leq \delta_{ws}(k)R_x^{max}, \quad (4.21)$$

$$(R_x(k) - l_{ws}) \geq (1 - \delta_{ws}(k))(-R_x^{max}). \quad (4.22)$$

Having location constraints, we can now constrain the robot to only pick up and drop off deliverables whenever it is near the inventory station or

the human work station:

$$0 \leq u_{ipu}(k) \leq \delta_{is}, \quad (4.23)$$

$$0 \leq u_{wpu}(k) \leq \delta_{ws}, \quad (4.24)$$

$$0 \leq u_{ido}(k) \leq \delta_{is}, \quad (4.25)$$

$$0 \leq u_{wdo}(k) \leq \delta_{ws}, \quad (4.26)$$

Next, to specify capacity constraints 4 and 5, we simply state that the robot cannot take actions that will cause it to exceed its carrying capacity of 100% or to drop off deliverables when it doesn't have any.

$$0 \leq R_d \leq 1.0 \quad (4.27)$$

To specify constraint 6, we introduce two boolean variables,  $\delta_{mp}$  and  $\delta_{mn}$  which are true if the robot exerts a positive effort and negative effort respectively. This is necessary because taking absolute values does not satisfy the Disciplined Convex Programming (DCP) [12] ruleset.

$$\delta_{mp} = 1 \Leftrightarrow u_{move} > 0, \delta_{mn} = 1 \Leftrightarrow u_{move} < 0. \quad (4.28)$$

These are again specified as inequality constraints

$$(1 - \delta_{mp}(k))u_{move}^{min} < u_{move}(k) < \delta_{mp}(k)u_{move}^{max}, \quad (4.29)$$

$$\delta_{mn}(k)u_{move}^{min} < u_{move}(k) < (1 - \delta_{mn}(k))u_{move}^{max}. \quad (4.30)$$

Finally, to encode the overall change in battery due to robot movement and charging, we specify

$$u_{charge} = \delta_{charge} - (\delta_{move})\gamma_{movepenalty} \quad (4.31)$$

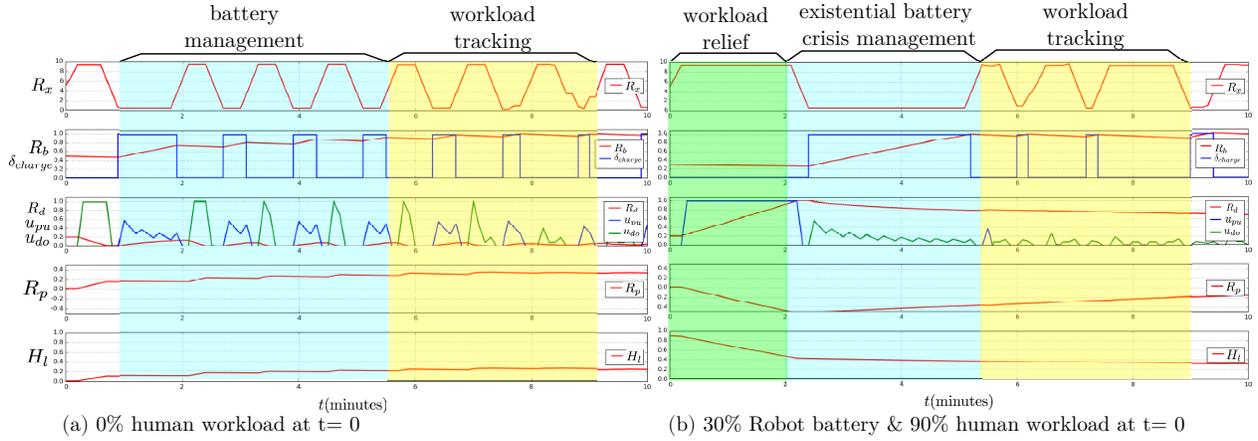


Figure 4.3: Assistant Robot Simulation Results: For both (a) and (b), the robot worries more about the human’s workload more than its own battery levels and productivity. Note that  $u_{pu} = u_{ipu} + u_{wpu}$  and  $u_{do} = u_{ido} + u_{wdo}$  to indicate the total deliverable pick up and drop off actions respectively. Also, the W.S. and I.S. are located at  $l_{ws} = 9$  and  $l_{is} = 1$  respectively. In (a), the robot initially drops off the deliverables it is carrying to give the human work and proceeds to charge its own batteries while slowly dropping off more work to the human. In (b), the robot notices that the human is overworked and proceeds to remove work from the human at the cost of the robot’s own productivity until the human’s workload becomes manageable. The robot also charges its low battery levels to remain operational. Then, the robot proceeds to slowly drop off work to the human at a manageable rate, which also makes the robot’s perception of its own productivity to rise again.

where  $\delta_{charge} = \delta_{is}$ , and  $\delta_{move} = (\delta_{mp} + \delta_{mn})$ . Additionally, to be consistent with the constants’ effects as specified in Fig. 4.2 and Eq. 4.10,  $\gamma_{movepenalty} = \gamma_{penalty}/\gamma_c$ . That is, the battery is charged whenever it is near the inventory station (constraint 2) and the battery is decreased whenever the robot moves (constraint 6).

## 4.6 Results

We provide two test cases to the robot. For both cases, the optimization routine is set to maximize robot battery and productivity, and target a 25% human workload ( $y^{ref} = [1; 1; 0.25]$ ). The following weights  $w_b = 1$ ,  $w_p = 1$ , and  $w_l = 10$  were used. That is, the robot cares more about ensuring the human is never overworked over its own battery and productivity levels. The simulation parameters are available in the linked code repository.

In the first case (Fig. 4.3a), the robot starts between the inventory station and the human work station and the human starts out with no workload. The robot first drops off its deliverables to the human and proceeds to charge its batteries. Then it moves back and forth to bring just enough deliverables to ensure that the human has a manageable workload (always at 25%).

In the second case (Fig. 4.3b), all the parameters and initial conditions are the same except that the human starts out with 90% workload and the robot starts out with 30% battery. Despite having low battery, the robot rushes to the human and removes the workload from the human. This causes the robot's productivity to become negative as per the definition of robot productivity. The robot understands that the human being overworked is the more important issue. When the human's workload is at 40%, the robot charges its battery to remain operational. Then, the robot returns to a behavior which ensures the human has a manageable workload

## 4.7 Discussion

Potential future work includes testing the assistive robot MPC model as well as further improvements on modeling human-robot connection dynam-

ics. By taking inspiration from SCT models found in the exercise behavior intervention community, this work explored the possibility of treating HRI as controllable dynamical systems in which state-of-the-art techniques from the controls community can be leveraged.

## Chapter 5

### Summary and Future Outlook

This thesis presented methodologies for detecting humans, recognizing arm motion gestures, and generating human-aware control policies as an attempt to address the necessary capabilities of a human-aware robot. Chapter 2 presented an algorithm for detecting humans given a 3D point cloud data. Chapter 3 tested many hypotheses when arm motion gestures are represented as Dynamic Movement Primitives and classified with Gaussian Mixture Models. Finally, Chapter 4 presented a new novel method for representing HRI scenarios as a dynamical system and generate human-aware control policies using Model Predictive Control with mixed-integer constraints.

Plenty of work remains for the future as this thesis only addressed a subset of the technical challenges needed to be solved to have a human-aware robot.

Half of the problem with creating a truly autonomous human-aware robot comes from creating deployable algorithms that can extract useful information from sensor data. For example, the problem of Human detection, tracking, and pose estimation as well as human action and intent recognition have been framed as supervised machine learning problems. Recent advances in the field of Convolutional Neural Networks (CNNs) are defeating well engineered supervised classification problems. At least in the very near future,

CNNs have the most promise in creating general detection, tracking, and recognition modules for human-aware robots.

The other half of the problem comes from how to generate safe, high-performance, human-aware robot policies. Here, using CNNs has not been shown to be effective or desirable as it is difficult to state or reason about performance guarantees due to its black box nature. At the very least, the approach of modeling HRI as dynamical systems and using MPC to generate control policies enables the engineer to design and reason about the robot's actions. While not perfect, having a clear understanding of the robot's actions make it deployable. Thus, designing a framework for autonomous behaviors remains to be an important research avenue. Representing HRI as dynamical systems and using MPC is one approach and must be tested on a real system.

## Bibliography

- [1] Kai O. Arras, Óscar Martínez Mozos, and Wolfram Burgard. Using boosted features for the detection of people in 2D range data. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3402–3407, 2007.
- [2] Adrian Ball, David Rye, Fabio Ramos, and Mari Velonaki. A comparison of unsupervised learning algorithms for gesture clustering. *Proceedings of the 6th international conference on Human-robot interaction - HRI '11*, page 111, 2011.
- [3] Albert Bandura. *Social Foundations of Thought and Action: A Social Cognitive Theory*. Prentice Hall, 1 edition, 1985.
- [4] Garrett Bernstein, Nyk Lotocky, and Dan Gallagher. Robot Recognition of Military Gestures CS 4758 Term Project. 2012.
- [5] Cynthia Breazeal, Matt Berlin, Andrew Brooks, Jesse Gray, and Andrea L. Thomaz. Using perspective taking to learn from ambiguous demonstrations. *Robotics and Autonomous Systems*, 54(5):385–393, 2006.
- [6] Cynthia Breazeal, Jesse Gray, and Matt Berlin. An Embodied Cognition Approach to Mindreading Skills for Socially Intelligent Robots. *The International Journal of Robotics Research*, 28(5):656–680, 2009.
- [7] Cynthia Breazeal, Cory D Kidd, Andrea Lockerd Thomaz, Guy Hoffman, and Matt Berlin. Effects of Nonverbal Communication on Efficiency

- and Robustness of Human-Robot Teamwork.pdf. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 708–713, 2005.
- [8] Andrew G Brooks and Cynthia Breazeal. Working with Robots and Objects: Revisiting Deictic Reference for Achieving Spatial Common Ground. *Gesture*, pages 297–304, 2006.
- [9] Carnegie Robotics. MultiSense SL. [http://files.carnegierobotics.com/products/MultiSense\\_SL/MultiSense\\_SL\\_brochure.pdf](http://files.carnegierobotics.com/products/MultiSense_SL/MultiSense_SL_brochure.pdf), 2014.
- [10] Crystal Chao and Andrea Lockerd Thomaz. Timed Petri nets for fluent turn-taking over multimodal interaction resources in human-robot collaboration. *The International Journal of Robotics Research*, page 0278364915627291, 2016.
- [11] Travis Dewolf. Dynamic movement primitives: The basics part 1. <http://tinyurl.com/z49c7g5>. Last Accessed: 2015-12-04.
- [12] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.
- [13] Hao Ding, Gunther Reißig, Dominic Groß, and Olaf Stursberg. Mixed-Integer Programming for Optimal Path Planning of Robotic Manipulators. *Int. Conference on Automation Science and Engineering*, pages 133–138, 2011.
- [14] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4):743–761, 2012.

- [15] Paul Ekman and Wallace V Friesen. The Repertoire of Nonverbal Behavior: Categories, Origins, Usage, and Coding. *Semiotica*, 1(1):49–98, 1969.
- [16] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3Deep: Fast Object Detection in 3D Point Clouds Using Efficient Convolutional Neural Networks. 2016.
- [17] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2015.
- [18] Frederik Hegger, Nico Hochgeschwender, Gerhard K Kraetzschmar, and Paul G Ploeger. People Detection in 3D Point Clouds using Local Surface Normals. *RoboCup 2012: Robot Soccer World Cup XVI*, pages 154–165, 2013.
- [19] David Held, Devin Guillory, Brice Rebsamen, Sebastian Thrun, and Silvio Savarese. A Probabilistic Framework for Real-time 3D Segmentation using Spatial, Temporal, and Semantic Cues. *Proceedings of Robotics: Science and Systems*, 2016.
- [20] Guy Hoffman and Cynthia Breazeal. Collaboration in Human-Robot Teams. *Proc. of the AIAA 1st Intelligent Systems Technical Conference, Chicago, IL, USA*, pages 1–18, 2004.
- [21] Guy Hoffman and Cynthia Breazeal. Cost-Based Anticipatory Action Selection for Human-Robot Fluency. *IEEE Transactions on Robotics*, 23(5):952–961, 2007.
- [22] Heiko Hoffmann, Peter Pastor, Dae-Hyung Park, and Stefan Schaal. Biologically inspired dynamical systems for movement generation: Automatic

- real-time goal adaptation and obstacle avoidance. *2009 IEEE International Conference on Robotics and Automation*, pages 2587–2592, 2009.
- [23] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–73, 2013.
- [24] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, 2(May):1–6, 2002.
- [25] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.
- [26] Kiyosumi Kidono, Takeo Miyasaka, Akihiro Watanabe, Takashi Naito, and Jun Miura. Pedestrian Recognition Using High-definition LIDAR. *IEEE Intelligent Vehicles Symposium*, pages 405–410, 2011.
- [27] Hema Swetha Koppula, Rudhir Gupta, and Ashutosh Saxena. Learning human activities and object affordances from RGB-D videos. *The International Journal of Robotics Research*, 32(8):951–970, 2013.
- [28] Steven M. LaValle and James J Kuffner. Randomized Kinodynamic Planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.
- [29] Xia Lu, Chen Chia-Chih, and Jake K Aggarwal. Human detection using depth information by Kinect. *CVPR 2011 WORKSHOPS*, pages 15–22, 2011.

- [30] Guilherme Maeda, Marco Ewerton, Rudolf Lioutikov, Heni Ben Amor, Jan Peters, and Gerhard Neumann. Learning Interaction for Collaborative Tasks with Probabilistic Movement Primitives. *International Conference on Humanoid Robots*, pages 527–534, 2014.
- [31] César A Martín, Daniel E Rivera, William T Riley, Eric B Hekler, Matthew P Buman, Marc A Adams, and Abby C King. A Dynamical Systems Model of Social Cognitive Theory. In *American Control Conference(ACC)*, pages 2407–2412, 2014.
- [32] Cysar A Martyn, Daniel E Rivera, and Eric B Hekler. A Decision Framework for an Adaptive Behavioral Intervention for Physical Activity Using Hybrid Model Predictive Control. In *American Control Conference(ACC)*, pages 3576–3581, 2016.
- [33] Robin R Murphy. A Survey of Social Gaze Categories and Subject Descriptors. *ACM/IEEE International Conference on Human-Robot Interaction*, pages 253–254, 2011.
- [34] Scott Niekum. Ar tracker alvar ros package. [http://wiki.ros.org/ar\\_track\\_alvar](http://wiki.ros.org/ar_track_alvar). Last Accessed: 2015-12-04.
- [35] Scott Niekum, Sarah Osentoski, Christopher G Atkeson, and Andrew G Barto. Online Bayesian Changepoint Detection for Articulated Motion Models. *2015 IEEE International Conference on Robotics and Automation*, 2015.
- [36] OpenNI. openni\_tracker ROS package. [http://wiki.ros.org/openni\\_tracker](http://wiki.ros.org/openni_tracker), 2013.

- [37] Nicholas Paine, Joshua S Mehling, James Holley, Nicolaus A Radford, Gwendolyn Johnson, Chien-Liang Fok, and Luis Sentis. Actuator control for the nasa-jsc valkyrie humanoid robot: A decoupled dynamics approach for torque control of series elastic robots. *Journal of Field Robotics*, 32(3):378–396, 2015.
- [38] Panagiotis Papadakis, Patrick Rives, Anne Spalanzani, Panagiotis Papadakis, Patrick Rives, Anne Spalanzani, Adaptive Spacing, Panagiotis Papadakis, Patrick Rives, and Anne Spalanzani. Adaptive Spacing in Human-Robot Interactions. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2627–2632, 2014.
- [39] Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. Probabilistic Movement Primitives. *Neural Information Processing Systems*, pages 1–9, 2013.
- [40] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. *2009 IEEE International Conference on Robotics and Automation*, pages 763–768, 2009.
- [41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [42] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Mg. ROS: an

- open-source Robot Operating System. *ICRA workshop on open source software*, 3(3.2):5, 2009.
- [43] Nicolaus A Radford, Philip Strawser, Kimberly Hambuchen, Joshua S Mehling, William K Verdeyen, A Stuart Donnan, James Holley, Jairo Sanchez, Vienny Nguyen, Lyndon Bridgwater, et al. Valkyrie: Nasa’s first bipedal humanoid robot. *Journal of Field Robotics*, 32(3):397–419, 2015.
- [44] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. 794:185–192, jun 2015.
- [45] Charles Rich, Brett Ponsler, Aaron Holroyd, and Candace L. Sidner. Recognizing engagement in human-robot interaction. *Human-Robot Interaction (HRI), 2010 5th ACM/IEEE International Conference on*, pages 375–382, 2010.
- [46] Arthur Richards and Jonathan How. Mixed-integer programming for control. *Proceedings of the 2005, American Control Conference, 2005.*, pages 2676–2683, 2005.
- [47] Arthur Richards, Tom Schouwenaars, Jonathan P. How, and Eric Feron. Spacecraft Trajectory Planning with Avoidance Constraints Using Mixed-Integer Linear Programming. *Journal of Guidance, Control, and Dynamics*, 25(4):755–764, 2002.
- [48] Laurel D. Riek. Wizard of Oz Studies in HRI: A Systematic Review and New Reporting Guidelines. *Journal of Human-Robot Interaction*, 1(1):119–136, 2012.

- [49] Radu Bogdan Rusu. Semantic 3d object maps for everyday manipulation in human living environments. *KI-Künstliche Intelligenz*, 24(4):345–348, 2010.
- [50] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 3212–3217. IEEE, 2009.
- [51] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.
- [52] Sven R. Schmidt-Rohr, Martin Lösch, and Rüdiger Dillmann. Human and robot behavior modeling for probabilistic cognition of an autonomous service robot. *Proceedings of the 17th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN*, pages 635–640, 2008.
- [53] Luis Sentis. Lecture notes. The University of Texas at Austin, 2016.
- [54] Julie Shah, James Wiken, Brian Williams, and Cynthia Breazeal. Improved human-robot team performance using Chaski, a Human-Inspired Plan Execution System. *Proceedings of the 6th international conference on Human-robot interaction*, pages 29–36, 2011.
- [55] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time Human Pose Recognition in Parts from Single Depth Images. *Communications of the ACM*, 56(1):116–124, 2013.

- [56] Emrah Akin Sisbot, Kuis F. Marin-Urias, Rachid Alami, and Thierry Siméon. A human aware mobile robot motion planner. *IEEE Transactions on Robotics*, 23(5):874–883, 2007.
- [57] J Cole Smith and Z Caner Taskin. A Tutorial Guide to Mixed Integer Programming Models and Solution Techniques. *Optimization in Medicine and Biology*, pages 1–23, 2008.
- [58] Padhraic Smyth. The EM Algorithm for Gaussian Mixtures The EM Algorithm for Gaussian Mixture Models. <http://www.ics.uci.edu/~smyth/courses/cs274/notes/EMnotes.pdf>. Last Accessed: 2015-12-04.
- [59] Tian-Shu Wang, Heung-Yeung Shum, Ying-Qing Xu, and Nan-Ning Zheng. Unsupervised analysis of human gestures. *Advances in Multimedia Information Processing. Lecture Notes in Computer Science*, 2195:174–181, 2001.
- [60] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE Multimedia*, 19(2):4–10, 2012.

## Vita

Steven Jens Mangubat Jorgensen was born in Manila, Philippines on April 1st, 1992. He's the son of Robert Jens M. Jorgensen and Marissa M. Jorgensen. He received the Bachelor of Science degree in Engineering from the Massachusetts Institute of Technology (MIT) on June 2014. He decided to pursue his graduate studies as an effort towards understanding how to control robots more intelligently. He applied to The University of Texas at Austin to join Dr. Luis Sentis's Human-Centered Robotics Lab (HCRL). From 2014-2015 he was funded by PI Electronics to work on designing humanoids with Electrorheological-Fluid Actuators. In August 2015, his NASA Space Technology Research Fellowship (NSTRF 2015) began, which enabled him to start focusing on high-performance, safe, human-aware robots. With the support of his NASA fellowship, this thesis is a summary of the subset of work performed from Fall 2015 to Fall 2016.

Permanent address: [stevenjensj@gmail.com](mailto:stevenjensj@gmail.com)

This thesis was typeset with  $\text{\LaTeX}^\dagger$  by the author.

---

<sup>†</sup> $\text{\LaTeX}$  is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's  $\text{\TeX}$  Program.